2019-ENAM-0016

École doctorale n° 432 : Sciences des Métiers de l'ingénieur

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

l'École Nationale Supérieure d'Arts et Métiers

Spécialité "Conception"

présentée et soutenue publiquement par

Yohann AUDOUX

le 14 juin 2019

Développement d'une nouvelle méthode de réduction de modèle basée sur les hypersurfaces NURBS (Non-Uniform Rational Basis Splines)

Directeur de thèse : **Jérôme PAILHÈS**
Co-directeur de la thèse : **Marco MONTEMURRO**

Jury

M. Laurent GALLIMARD, PU, LEME, Université Paris Nanterre	Président
M. Ludovic CHAMOIN, PU, LMT, ENS Paris-Saclay	Rapporteur
M. Elias CUETO, Professeur, Université de Saragosse	Rapporteur
M. Emmanuelle ABISSET, PU, I2M, UMR 5295, Ecole Nationale Supérieure d'Arts et Métiers	Examinateur
M. Angela VINCENTI, MCF, Institut d'Alembert, Sorbonne Université	Examinateur
M. Jérôme Pailhès, PU, I2M, UMR 5295, Ecole Nationale Supérieure d'Arts et Métiers	Examinateur
M. Marco Montemurro, MCF HDR, I2M, UMR 5295, Ecole Nationale Supérieure d'Arts et Métiers	Examinateur

T H È S E

Arts et Métiers ParisTech - Campus de Bordeaux
Institut de Mécanique et d'Ingénierie – I2M, UMR CNRS 5295

A ma fiancée Mélissande,

Remerciements

L'écriture de ces quelques lignes marque la fin d'une aventure dont je n'ai, bien sûr, pas été le seul acteur. A ce titre, je souhaiterais tout d'abord remercier mon directeur de thèse Jérôme PAILHES ainsi que mon co-directeur Marco MONTEMURRO pour le temps qu'ils ont consacré à mon encadrement ainsi que les précieux et nombreux conseils qu'ils m'ont prodigués. Au delà de leurs compétences scientifiques, le recul qu'apporte Jérôme sur les problèmes considérés, associé à la grande rigueur de Marco m'ont permis de m'épanouir et surtout de m'améliorer tout au long de mon doctorat. Leurs avis favorables sur ma capacité à poursuivre une carrière d'enseignant-chercheur est une marque de confiance qui me va droit au cœur. Je souhaite également remercier le Laboratoire I2M pour m'avoir accueilli ainsi que le ministère de l'enseignement supérieur et de la recherche et l'Ecole Normale Supérieure Paris-Saclay pour l'octroi du financement spécifique pour normalien qui a permis de réaliser ces travaux de thèses.

La reconnaissance du travail accompli par ses pairs est le fondement même d'un doctorat et de la carrière d'un jeune chercheur. Je souhaite donc adresser mes sincères remerciement aux rapporteurs de cette thèse, Ludovic CHAMOIN et Elias CUETO, ainsi qu'aux examinateurs, Laurent Gallimard, Angela Vincenti et Emmanuelle ABISSET-CHAVANNE, pour l'intérêt qu'ils ont porté à mon travail et les discussions intéressantes qu'ils ont amenées lors de la soutenance.

Ce travail n'aurait pas pu être mené à bien, sans l'ambiance de convivialité qui règne entre les doctorants et je dois bien évidemment adresser ma toute gratitude à Giulio. Nous avons commencé ensemble et partagé bien plus qu'un bureau et des NURBS. Je profite de ces remerciements pour te confirmer que le sentiment d'amitié que tu me portes est pleinement réciproque. Laura et toi êtes de très belles personnes que je suis fier de compté parmi mes amis. J'ai également une pensée pour notre auto-proclamé meilleur stagiaire, Michele Iacopo, que je considère également comme un ami, et je sais déjà que nos balades à moto seront des plus agréables. Je n'oublie pas non plus tous ceux qui participe, ou ont participé, à cette superbe ambiance : Jeremy, Hugo, Anton, Enrico, Lorenzo, Pietro, Rachel, Giulia, Thibaut Rn, Thibaut Rd, Marco D, Thomas, Marco P, Soukaina, Alexandre, Alberto, Grace, Jorge, Jose ... L'I2M a la chance d'avoir une pluralité culturelle, bonne pour la recherche, mais également pour l'enrichissement personnel. J'ai beaucoup appris de vous tous et je vous en remercie, vous avez contribué à mon épanouissement.

Si une bonne ambiance de travail est primordiale pour réussir une thèse, le soutient de ses amis et de sa famille est indispensable. C'est pourquoi j'adresse mes remerciements les plus sincères à Pierre et Manon, vous êtes bien plus que de simples amis d'enfance, vous êtes une seconde famille. Je remercie également mes beaux-parents, Laurence et Philippe, ainsi que mes deux beaux-frères, Médéric et Kellian. Vous m'avez accompagné dans les bons comme dans les

mauvais moments me faisant ressentir votre fierté à chaque instant.

J'ai une pensée particulière pour mon père, Dominique, qui nous a quitté trop tôt et j'ose espérer qu'il serait fier s'il était encore des nôtres. Je souhaite remercier personnellement Olivier. Tu as permis à ma mère de retrouver un équilibre, tu lui as apporté beaucoup et, même si j'espère que tu le sais déjà, je te considère comme un membre à part entière de ma famille. Je remercie bien évidemment ma mère, Corinne, pour l'amour inconditionnel qu'elle m'apporte et la fierté non dissimulée qu'elle communique aux gens lorsqu'elle évoque mon parcours ainsi que mon petit frère, David, pour son amour fraternel et nos débats vifs et animés. Enfin, je remercie ma grand mère, Claudine, qui a toujours été un soutient indéfectible pour moi.

Mon dernier merci va à toi, Mélissande, celle qui partage ma vie depuis plus de dix ans maintenant. Te dédier cette thèse n'est bien sûr pas suffisant à côté des sacrifices qui m'ont permis de la réaliser. Ton soutient était essentiel ces dernières années et l'est toujours autant. Lorsque je doute, il me suffit de m'observer à travers ton regard pour me sentir mieux. Je ne peux pas me contenter de te dire simplement merci, ces quelques lignes sont là pour te déclarer mon amour et t'assurer que tu es la femme avec qui je veux passer le reste de ma vie.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction, contexte et objectifs de la thèse

1.1 Contexte de la thèse

Malgré les progrès incontestables dans les sciences informatiques durant les dernières décennies, un certain nombre de problèmes restent difficiles à traiter, soit par leur complexité numérique intrinsèque, soit en raison de contraintes spécifiques telles que la nécessité de traitement en temps réel (ou quasi-réel). Sans être exhaustif, on peut citer, par exemple, les problèmes de caractérisation inverse et d'optimisation qui nécessitent de calculer la ou les sorties d'un modèle un grand nombre de fois, ou le déploiement de plateformes dont le traitement doit être effectué en temps réel, telles que des applications en réalité virtuelle, réalité augmentée ou bien encore le contrôle actif de systèmes complexes. Ces problèmes ne peuvent pas être traités par les simples performances informatiques brutes et nécessitent des techniques de résolutions appropriées. Dans ce contexte, un nombre important de méthodes de réduction de modèle existent et permettent de diminuer la complexité d'un modèle afin d'en réduire le coût de calcul tout en garantissant une certaine précision.

Ces méthodes sont ainsi capables de réduire les temps de calcul de simulations multi-champs et/ou multi-échelles complexes afin de les intégrer, par exemple, dans des processus d'optimisation. Le processus de réduction de modèle peut être décrit comme la recherche du modèle réduit (ou métamodèle) nécessitant moins de *ressources* pour être évalué que le modèle complexe duquel il a été obtenu. La définition de *ressources* dépend du problème considéré. Par exemple, un problème de réduction d'image aura pour objectif de réduire *le nombre de données* nécessaires à l'évaluation du métamodèle tandis qu'un problème d'optimisation se focalisera sur la réduction du *temps d'évaluation* du modèle réduit (i.e. le temps pour obtenir les sorties du métamodèle à partir des entrées).

Il est important de noter que toutes les méthodes actuelles ont besoin d'une étape de calibration qui n'est pas considérée dans le temps d'évaluation du métamodèle. Ce temps correspond à une phase *off-line* d'utilisation et le temps nécessaire à cette étape est bien souvent supérieur au temps d'évaluation du métamodèle qui correspond, lui, à l'utilisation *on-line*. L'objectif de cette étape est de trouver les paramètres du modèle réduit permettant de garantir une certaine

précision.

La conception de systèmes est de plus en plus pilotée par les données, avec des temps de cycles réduits, de moins en moins de prototypage et une augmentation de la complexité des systèmes (systèmes pluridisciplinaires par exemple). Dans ce cadre, les techniques de réduction de modèle offrent un énorme potentiel par leur capacité à encapsuler des données obtenues par de multiples simulations (ou expériences) dans un modèle mathématique unique. Ce modèle est, généralement, beaucoup plus simple et plus rapide d'utilisation que les modèles ou les expérimentations ayant permis de l'obtenir. L'emploi d'un modèle réduit est particulièrement attrayant dans les cas où : (i) la modélisation initiale d'un système limite le nombre de données pouvant être générées ; (ii) le coût expérimental est trop élevé pour effectuer beaucoup de cas ; (iii) la réponse dépend de plusieurs modèles ; (iv) le modèle doit être utilisé dans une boucle d'optimisation ; (v) un nombre important de données sont disponibles mais sont difficilement exploitables de manière directe ; (vi) l'espace de stockage des données doit être réduit sans pertes d'information.

1.2 La réduction de modèle

Pour illustrer le concept de la réduction de modèle, considérons un système réel de type MIMO (*Multiple Inputs Multiple Outputs*) caractérisé par N entrées, M sorties et une fonction de transfert $\mathbf{z}$ telle que :

$$\begin{array}{rccc} & \mathbb{R}^N & \rightarrow & \mathbb{R}^M, \\ \mathbf{z}: & \mathbf{X} & \rightarrow & \mathbf{z}(\mathbf{X}), \end{array} \tag{1.1}$$

avec $\mathbf{X} = \left(X^{(1)}, \ldots, X^{(N)}\right)$ le vecteur contenant les N entrées du système et $\mathbf{z}(\mathbf{X}) = (z^{(1)}(\mathbf{X}),$ ... , $z^{(M)}(\mathbf{X}))$ le vecteur contenant les M sorties de la fonction de transfert du système MIMO. Dans certains cas, cette fonction peut être complètement connue de manière analytique mais c'est rarement le cas lorsqu'il s'agit de problèmes industriels. De plus, il n'est pas rare que cette fonction doive être approximée par des expérimentations ou par le biais de méthodes de calcul par éléments finis couteuses. L'objectif de la réduction de modèle est de trouver la fonction $\hat{\mathbf{z}}(\mathbf{X})$ nécessitant moins de ressources que $\mathbf{z}(\mathbf{X})$ pour être évaluée de la forme suivante :

$$\begin{array}{rccc} & \mathbb{R}^N & \rightarrow & \mathbb{R}^M, \\ \hat{\mathbf{z}}: & \mathbf{X} & \rightarrow & \hat{\mathbf{z}}(\mathbf{X}) = \mathbf{z}(\mathbf{X}) + \boldsymbol{\epsilon}(\mathbf{X}), \end{array} \tag{1.2}$$

où $\hat{\mathbf{z}}(\mathbf{X})$ est la fonction approximant la vraie fonction de transfert $\mathbf{z}(\mathbf{X})$ et $\boldsymbol{\epsilon}(\mathbf{X})$ est l'erreur d'approximation au point $\mathbf{X}$. La fonction $\boldsymbol{\epsilon}(\mathbf{X})$ est une fonction bornée dont les bornes sont directement liées à la précision d'approximation du métamodèle. Comme $\hat{\mathbf{z}}(\mathbf{X})$ n'est pas connue dans la majorité des cas, les bornes de la fonction $\boldsymbol{\epsilon}(\mathbf{X})$ sont en général estimées sur un ensemble représentatif de points de validation.

La plupart des méthodes existantes actuellement sont des techniques dites *a posteriori* car la phase de calibration de ces métamodèles nécessite l'acquisition d'un ensemble de points obtenus expérimentalement ou par le biais d'un modèle complet (i.e. couteux) du système réel. La seule méthode dite *a priori* connue à ce jour est la méthode de décomposition aux valeurs propres généralisée ou PGD (*Proper Generalised Decomposition*) [4] qui construit le modèle réduit sans avoir recours à un ensemble de points préalablement calculés. Cette méthode est basée sur une

représentation séparée de l'espace itérativement enrichie par l'ajout de *modes* jusqu'à atteindre un certain seuil de précision. Chaque mode ajouté donne lieu à la résolution d'un problème non-linéaire. Cette méthode s'est illustrée notamment dans la résolution de problèmes de chimie quantique insolvables jusqu'alors [5]. Elle présente cependant quelques limitations dont notamment le choix du pas de discrétisation, fixé par l'utilisateur, qui peut avoir un fort impact sur les résultats obtenus. De plus, une technique d'interpolation (ou d'approximation) doit être utilisée pour obtenir les valeurs en dehors des pas de discrétisation avec en général une plus mauvaise précision pour ceux-ci. Cela conduit également à des résultats moins précis pour des problèmes non-linéaires.

Parmi les méthodes de réduction de modèle *a posteriori*, la décomposition orthogonale aux valeurs propres ou POD (*Proper Orthogonal Decomposition*) [6,7] et le krigeage [8,9] sont les plus communément utilisées. La première méthode a pour objectif de diminuer *l'ordre* du problème considéré d'une taille K (correspondant au degrés de liberté pour un calcul par éléments finis par exemple) à la taille K' avec $K' \ll K$. Cette réduction est obtenue par la résolution d'un problème aux valeurs propres servant de base de projection orthogonale pour le problème considéré. La réduction est obtenue en ne conservant que les valeurs propres les plus importantes et en obtenant ainsi une base de projection réduite. Bien que fortement utilisée, dans le domaine des calculs dynamiques de mécanique des fluides [6], cette méthode nécessite, dans sa phase d'utilisation *on-line*, la résolution d'un système linéaire d'ordre K' qui peut s'avérer problématique pour des applications nécessitant des temps de réponse très court.

De son côté, le krigeage, qui a été développé dans un cadre géostatistique, est connu comme le meilleur estimateur linéaire non biaisé. Cette technique est basée sur la prise en compte de la structure de dépendance spatiale des données. Dans le cadre de la réduction de modèle, la méthode se focalise sur le calcul de la covariance grâce à l'utilisation d'une fonction de corrélation spatiale définie par l'utilisateur. Connu pour être un estimateur non biaisé, car aucun biais n'est introduit dans le calcul, il est important de noter que du biais est introduit par le choix de la fonction de corrélation spatiale. Il existe deux techniques permettant de déterminer les paramètres de cette fonction, la validation croisée et le maximum de vraisemblance, qui augmentent significativement le temps nécessaires pour la phase de calibration du métamodèle. De plus, cette approche ne permet pas de réduire le nombre de données nécessaire à l'évaluation du modèle réduit et l'utilisation *on-line* s'appuie sur l'inversion d'une matrice dont la taille est directement liée au nombre de points connus. Ces limitations rendent la méthode adaptée au cas où peu de données sont disponibles.

Parmi toutes les méthodes existantes, les réseaux de neurones artificiels, ou *Artificial Neural Networks* (ANNs) en anglais, méritent une attention particulière car elles sont actuellement les seules méthodes capables de traiter des problèmes caractérisés par des milliers d'entrées [10,11]. Mais ce grand potentiel est associé à une complexité qui les rend inaccessibles à un utilisateur non formé à leur utilisation et surtout à leur paramétrisation. Cette difficulté s'accompagne souvent d'une phase d'essai/erreur pour le choix des paramètres. Il n'existe pas de méthode permettant de savoir si les données utilisées pour la phase de calibration, nommée *apprentissage* dans le cas des réseaux de neurones, sont appropriées et il est en général difficile de savoir si le métamodèle obtenu ne souffre pas d'*over-fitting*.

Afin de répondre à ces limitations, la méthode que nous proposons utilise une hypersurface

NURBS (*Non-Uniform Rational Basis Splines*) de dimension M caractérisée par un espace paramétrique de dimension N dont les paramètres sont automatiquement déterminés sans nécessiter une approche essai/erreur. La méthode de réduction de modèle que nous proposons est une méthode *a posteriori* qui s'appuie sur un ensemble de points connus de la fonction de transfert $\mathbf{z}(\mathbf{X})$ d'un système MIMO. Ces points sont nommés points cibles $\{\mathbf{Q}_s = \mathbf{z}(\mathbf{X}_s)\}$ où la fonction $\mathbf{z}$ a été évaluée de manière exacte ou approchée pour le vecteur d'entrée $\mathbf{X}_s$. Dans cette thèse, on approxime la fonction $\mathbf{z}(\mathbf{X})$ par une hypersurface NURBS :

$$\begin{array}{rccc} & [0,1]^N & \rightarrow & \mathbb{R}^M, \\ \mathbf{H}: & \mathbf{u} & \rightarrow & \mathbf{H}(\mathbf{u}), \end{array} \tag{1.3}$$

où $\mathbf{H}(\mathbf{u}) \in \mathbb{R}^M$ est le point de l'hypersurface NURBS aux coordonnées dans l'espace paramétrique $\mathbf{u} = \left(u^{(1)}, \ldots, u^{(N)}\right) \in \mathbb{R}^N$. Les différentes coordonnées de $\mathbf{H}(\mathbf{u})$ représentent les M différentes sorties du système tandis que les coordonnées paramétriques $\mathbf{u}$ représentent les N entrées. En lien avec les définitions qui seront introduites au chapitre 3, et les pratiques usuelles du cadre des courbes et surfaces NURBS [2], les valeurs de $u^{(i)}$, $i = 1, \ldots, N$, sont bornées dans l'intervalle $[0,1]$. L'espace de départ de la fonction $\mathbf{z}(\mathbf{X})$ étant un sous espace $E \subset \mathbb{R}^N \neq [0,1]^N$, on détermine une fonction *bijective* $\mathbf{f}$ telle que :

$$\begin{array}{rccc} & E & \rightarrow & [0,1]^N, \\ \mathbf{f}: & \mathbf{X} & \rightarrow & \mathbf{f}(\mathbf{X}) = \mathbf{u}. \end{array} \tag{1.4}$$

Cette fonction permet de lier le vecteur des variables d'entrée $\mathbf{X}$ au vecteur des variables d'entrée sans dimensions $\mathbf{u}$ de manière unique.

1.3 Objectifs de la thèse

Cette thèse n'a pas vocation à proposer une méthode qui soit la *meilleure* pour *tous* les problèmes, car une telle méthode relève de l'utopie. En revanche, l'objectif des travaux présentés dans ce manuscrit est de proposer une méthode de réduction de modèle très versatile (i.e. capable de s'adapter à une grande variété de problèmes), ne nécessitant pas un paramétrage complexe, à fixer par l'utilisateur, dont l'impact sur le métamodèle obtenu n'est pas négligeable et proposant, s'il n'est pas le meilleur, un *bon* métamodèle du problème considéré.

Pour cela, nous avons développé une méthode de réduction de modèle basée sur l'utilisation des hypersurfaces NURBS. Le problème de réduction de modèle est ramené à un problème d'*hypersurface fitting* d'une hypersurface NURBS sur un ensemble de points cibles préalablement obtenus. Ce type d'approche n'est pas complètement nouveau (cf. [1] par exemple) mais l'originalité de ce travail réside dans l'absence d'hypothèses simplificatrices permettant de déterminer les paramètres de l'hypersurface NURBS considérée. En effet, les principales méthodes existantes aujourd'hui déterminent une grande partie des paramètres de l'hypersurface NURBS utilisée par des règles établies dans le cadre des courbes et surfaces NURBS [2]. Ces règles ont déjà été remises en question mais seulement dans le cas des courbes et surface NURBS [12, 13]. De plus les approches proposées se focalisent souvent sur l'optimisation des poids ou des *knot-vectors* mais rarement tous les paramètres en même temps. Les travaux présentés ici se démarquent des

démarches existantes en incluant tous les paramètres de l'hypersurface NURBS dans le processus d'*hypersurface fitting*.

Ce processus d'*hypersurface fitting* peut être modélisé par un problème d'optimisation dont les variables sont les paramètres indépendants de l'hypersurface NURBS. Cependant le principal inconvénient, lorsque tous les paramètres de l'hypersurface font partie des variables d'optimisation, est que le nombre de variables d'optimisation n'est pas constant. Ce nombre est en fait lié à certains des paramètres de l'hypersurface et, en particulier, les dimensions des *knot-vectors* dépendent directement du nombre de points de contrôle et des degrés. Ce problème a nécessité le développement d'une stratégie d'optimisation originale qui fait suite aux travaux de Montemurro [14].

Dans ces travaux, Montemurro [14] a introduit une nouvelle méthode d'optimisation de conception de structures complexes : l'optimisation de *systèmes modulaires*. Un système modulaire peut être défini comme un système composé par des "unités élémentaires" (les *modules*) et dont chaque module est caractérisé par le même vecteur d'inconnues (les variables de conception de chaque module) mais pas nécessairement les mêmes valeurs. Ainsi, les modules composant le système partagent le même vecteur général d'inconnues mais ils peuvent être définis par des valeurs différentes de ces inconnues. Dans le cas des structures, les systèmes modulaires sont massivement utilisés industriellement et, plus particulièrement, dans les domaines de l'aéronautique et l'automobile. Nous pouvons citer deux exemples classiques de systèmes modulaires :

- les stratifiés composés de n_{ply} plis élémentaires, où chaque pli représente un module caractérisé par différents paramètres comme, par exemple, les propriétés du matériau, l'épaisseur ou encore l'orientation des fibres ;
- les parties structurelles d'un avion, à savoir, les panneaux raidi composant le fuselage et les ailes, où chaque panneau peut être vu comme un système modulaire dans lequel l'élément unitaire est le raidisseur.

L'optimisation de systèmes modulaires est souvent une tâche difficile que l'on peut formaliser mathématiquement par un problème d'optimisation non-conventionnel. Le but de ce genre de problème est d'optimiser, d'une part, le nombre de modules du système et, d'autre part, les valeurs des paramètres de chaque module. D'un point de vue mathématique, cela signifie de rechercher la configuration optimale globale du système dans un espace de dimension N_{var} variable, où N_{var} est le nombre total de variables de conception qui dépend directement du nombre de modules composant le système. De plus, les variables peuvent être de différentes natures : continues, discrètes, etc.

En considérant ces deux aspects (i.e. nombre de variables de conception variable et de différentes natures), associé au fait que les problèmes d'optimisation de systèmes modulaires sont souvent non-linéaires et non-convexes, la stratégie développée par Montemurro s'inscrit dans le cadre des métaheuristiques et plus précisément dans le contexte des algorithmes génétiques (AGs). Cependant, les AGs standard ne sont pas capables de traiter des problèmes d'optimisation de systèmes modulaires, principalement parce qu'il ne peuvent pas résoudre des problèmes d'optimisation définis sur des espaces de dimension variable. Pour surmonter cette difficulté, le concept d'*espèces* et de nouveaux opérateurs génétiques, autorisant la reproduction entre des individus issus d'espèces différentes, ont été introduits dans l'AG BIANCA (*Biologically Inspired ANalysis of Composite Assemblages*) [15]. En particulier, les phases de *croisement* et de *muta-*

tion, dont la combinaison des deux est la phase de reproduction (le cœur même d'un AG) ont été modifiées significativement pour autoriser la reproduction d'individus appartenant à des espèces différentes. Cela se traduit par le fait que l'AG BIANCA ne suit plus seulement l'évolution des individus mais également celle des espèces. De cette manière, les nouveaux opérateurs génétiques sont indépendant du problème traité puisqu'ils sont seulement liés au concept d'espèces et transcendent ainsi la nature physique du problème considéré.

Cependant, même si BIANCA a démontré sa capacité à traiter de nombreux problèmes [16–18], cet AG reste limité par le fait qu'il ne peut traiter des problèmes d'optimisation de systèmes modulaires ne possédant qu'un seul type de module. Or, le problème d'*hypersurface fitting*, relatif au problème de réduction de modèle que nous devons traiter, peut être mis sous la forme d'un problème d'optimisation d'un système modulaire possédant N composants modulaires, N étant le nombre d'entrée du modèle réduit. Le premier point d'originalité du travail proposé est, donc, la généralisation du concept d'espèce proposé par Montemurro [14]. En effet, la structure des individus a été généralisée et étendue afin de contenir un nombre N_{mod} quelconque de composants modulaires, chacun caractérisé par un nombre propre de chromosomes, et indépendants les uns des autres. Le concept d'espèce n'est donc plus seulement lié au nombre de chromosomes d'un individu mais aux nombres de chromosomes de chacun des composants modulaires qui le composent. Le résultat de cette généralisation est une nouvelle métaheuristique, l'AG ERASMUS (*EvolutionaRy Algorithm for optimiSation of ModUlar Systems*) capable de traiter des problèmes d'optimisation de systèmes modulaires à nombre quelconque de composants modulaires. Ce cas est fréquent dans les problèmes industriels et on peut citer par exemple :

- les systèmes composés de plusieurs stratifiés, où chaque stratifié représente un composant modulaire, chacun composé de n_{ply}^{strat} plis élémentaires, où chaque pli représente un module caractérisé par différents paramètres comme, par exemple, propriétés du matériau, épaisseur, orientation des fibres ;
- les parties structurelles d'un avion, comme par exemple, la section d'un fuselage, où les lisses et les cadres représentent chacun un type de composant modulaire, eux mêmes caractérisés par leurs propres paramètres de conception.

Le second point d'originalité du travail proposé dans ce manuscrit est l'utilisation d'hypersurface NURBS pour la réduction de modèle sans introduire d'hypothèses simplificatrices sur les différents paramètres de l'hypersurface. En effet, le développement de la métaheuristique ERASMUS, couplé au développement d'une stratégie d'optimisation hybride baptisée HERO (*Hybrid EvolutionaRy Optimisation*), utilisant à la fois l'AG ERASMUS et une méthode déterministe pour la résolution de problèmes d'optimisation, permet de déterminer l'ensemble des paramètres de l'hypersurface sans aucunes hypothèses et de manière indépendante du problème considéré. En particulier, l'utilisation d'ERASMUS permet d'obtenir le nombre de variables d'optimisation ainsi que les valeurs des paramètres entiers régissant l'hypersurface, tandis que la méthode déterministe permet d'atteindre l'optimum le plus proche de la solution renvoyée par ERASMUS. La stratégie de réduction de modèle que nous avons développée a été utilisée sur des benchmarks de nature thermique et mécanique ainsi que sur un problème complexe concernant la charge critique de flambage d'un panneau raidi. Dans ce dernier cas, l'AG ERASMUS, couplé au métamodèle obtenu, a été utilisé pour résoudre un problème de minimisation de la masse du panneau, soumis à diverses contraintes dont une sur la charge critique de flambage.

1.4 Organisation du manuscrit

Ce manuscrit est organisé de la manière suivante :

- Le présent chapitre introduit le contexte et les objectifs de la thèse.
- Le chapitre 2 dresse un état de l'art des différentes méthodes de réduction de modèle existantes. Après une présentation des principes mathématiques de la réduction de modèle, les méthodes sont classées en trois grandes familles. La première famille correspond aux méthodes d'interpolation qui permettent d'obtenir un métamodèle à partir d'un certain nombre de données nommées points cibles. Leur particularité est que le métamodèle généré renvoi la valeur exacte de la fonction approximée au niveau des points cibles. La deuxième famille correspond aux méthodes d'approximation qui, comme dans le cas de l'interpolation, permettent d'obtenir un métamodèle à partir de données disponibles. La différence principale de ces méthodes est que le métamodèle obtenu ne renvoie pas la valeur exacte de la fonction approximée au niveau des points cibles mais une valeur approximée. Enfin la dernière famille de méthodes est celle de la réduction d'ordre. Ces méthodes ont pour but de diminuer la complexité du modèle afin d'en diminuer le temps de calcul.
- Le chapitre 3 présente les outils et méthodes utilisés dans ces travaux de thèse. Tout d'abord le formalisme mathématique des courbes et surfaces NURBS est présenté, puis le formalisme est généralisé au cas des hypersurfaces NURBS de dimensions quelconques. Les règles empiriques classiquement utilisées pour fixer certains paramètres des entités NURBS sont présentées et des explications sur le non-fondement de ces règles sont évoquées. Dans la seconde partie de ce chapitre, les concepts de base de l'optimisation ainsi que les fondements mathématiques des AGs sont exposés. L'AG BIANCA est présenté et la nouvelle métaheuristique ERASMUS que nous avons développée est introduite. En particulier, nous détaillons les nouveaux opérateurs génétiques permettant la phase de reproduction entre individus de différentes espèces. Ces opérateurs correspondent à une généralisation des opérateurs de l'AG BIANCA. Nous détaillons également la stratégie de pénalisation particulière d'ERASMUS, à savoir l'ADP (*Automtic Dynamic Penalisation*), héritée de l'AG BIANCA. Enfin, nous présentons les généralités et les fondements mathématiques des méthodes de résolution déterministes (i.e. basées sur le gradient de la fonction objectif).
- Le chapitre 4, après un brève introduction, expose la formulation mathématique du problème de réduction de modèle. Pour cela, la liste des variables d'optimisation, correspondant aux paramètres de l'hypersurface NURBS, est dressée et le problème de réduction de modèle est formulé comme un problème d'optimisation non-linéaire sous contraintes. En particulier, nous montrons que ce problème d'optimisation peut être vu comme un problème d'optimisation d'un système modulaire possédant N types de composants modulaires, N étant le nombre d'entrées du métamodèle. La seconde partie de ce chapitre présente les stratégies permettant d'obtenir les coordonnées des points de contrôle. En particulier, les stratégies utilisées dans le cadre des surfaces NURBS sont généralisées aux cas des hypersurfaces. Ces stratégies ramènent le problème d'*hypersurface fitting* à une succession de problèmes de *curve fitting* mais nécessitent des conditions particulières pour être appliquées. C'est pourquoi nous présentons également une formulation hybride de ces stratégies. Enfin, la dernière partie présente la stratégie de résolution HERO que nous avons

développée. Elle est définie dans un cadre général (i.e. utilisable pour un grand nombre de problèmes) et appliquée au cas de l'*hypersurface fitting* par une entité NURBS.

- Le chapitre 5 présente les résultats obtenus par notre méthode de réduction de modèle sur trois benchmarks. Le premier traite un problème d'approximation de champ de température d'une plaque mince soumise à divers flux thermiques. Ce problème admet quatre paramètres d'entrée et une sortie. La particularité de ce benchmark réside dans le fait que les conditions limites appliquées à la plaque rendent le champ de température indépendant d'un des paramètres d'entrée. Le deuxième benchmark traite un problème d'approximation du champ de déplacement d'une plaque mince encastrée à une extrémité et soumise à un effort à son autre extrémité. Ce problème admet quatre entrées et deux sorties. Sa particularité est due au fait que la norme de l'effort appliquée fait partie des paramètres d'entrée du métamodèle. Enfin, le dernier benchmark traite un problème d'approximation d'un champ de déplacement d'une plaque mince trouée. Ce problème admet lui aussi quatre entrées et deux sorties. Sa particularité est que la topologie du domaine, et donc la distribution des points cibles, change pour chaque combinaison des paramètres d'entrée : il en résulte un problème d'*hypersurface fitting* fortement non-convexe. Les résultats de notre méthode de réduction de modèle sont comparés à une méthode basée sur les hypersurfaces NURBS issue de la littérature et utilisant les règles empiriques classiques pour fixer certains paramètres de l'hypersurface, à savoir, les *knot vectors* et les degrés.
- Le chapitre 6 présente un cas d'application de complexité industrielle. Le problème à traiter est un problème d'optimisation d'un panneau raidi réalisé par deux stratifiés, l'un pour la peau et l'autre pour le raidisseur. Le but de ce problème d'optimisation est de minimiser la masse du panneau, tout en respectant une contrainte sur la charge critique du premier mode de flambage. Le calcul de la masse étant réalisé par le biais d'une formule analytique, le problème nécessitant une réduction de modèle est l'estimation de la charge critique de flambage. Dans ce cas, le métamodèle à réaliser admet six entrées et une sortie. Dans un premier temps, la stratégie d'optimisation MS2L (*Multi-Scale Two-Level*) permettant l'optimisation de la structure est présentée. Cette stratégie résout deux problèmes d'optimisation interdépendants. Dans le premier, les paramètres géométriques et mécaniques des stratifiés composant la peau et le raidisseur sont déterminés. Les paramètres mécaniques utilisent le formalisme polaire qui est introduit brièvement. Dans le second problème, les orientations des plis composant les stratifiés sont déterminées de manière à satisfaire les valeurs des paramètres fournis par la résolution du premier problème. Le calcul de la charge critique de flambage n'intervient que dans le problème de premier niveau. C'est pourquoi, après avoir présenté les résultats obtenus par notre méthode de réduction de modèle, le métamodèle obtenu est utilisé pour résoudre le problème de premier niveau.
- Le dernier chapitre présente les conclusions générales ainsi que les perspectives de ces travaux de thèse.

Chapitre 2

État de l'art

2.1 Introduction

Le domaine de la réduction de modèle, ou *metamodeling* en anglais, représente un ensemble de méthodes et de techniques dont l'objectif principal est de générer le modèle d'un modèle appelé *modèle réduit* ou encore *métamodèle*. La conception de système est de plus en plus pilotée par les données, avec des temps de cycles réduits, de moins en moins de prototypage et une augmentation de la complexité des systèmes (systèmes pluridisciplinaires par exemple). Dans ce cadre, les techniques de réduction de modèle offrent un énorme potentiel par leur capacité à encapsuler des données obtenues par de multiples simulations (ou expériences) dans un modèle mathématique unique. Ce modèle est généralement beaucoup plus simple et plus rapide d'utilisation que les modèles ou les expérimentations ayant permis de l'obtenir. Nous pouvons d'ailleurs lister les différentes caractéristiques attendues d'un bon métamodèle :

- Minimiser le nombre de paramètres devant être fixés *a priori* par l'utilisateur,
- Ne pas nécessiter d'expertise de la part de l'utilisateur pour construire le métamodèle,
- Être plus rapide que le modèle dont il est issu en conservant la robustesse et la précision de celui-ci,
- Minimiser le nombre de données nécessaires à son utilisation.

Notons que la génération d'un modèle réduit passe, quelle que soit la méthode utilisée, par deux phases de calculs très différentes. Ces deux étapes ont des temps de calculs qui peuvent différer de quelques ordres de grandeurs. Dans un premier temps, les paramètres du métamodèle sont évalués à partir de données connues, sauf dans le cas de la méthode PGD (Proper Generalized Decomposition) qui ne nécessite pas systématiquement de calcul *a priori*. Cette étape est souvent appelée phase de *calibration* ou d'*apprentissage* (on utilise souvent apprentissage dans le contexte des *machines learning*). Le métamodèle obtenu peut ensuite être utilisé et le temps de calcul nécessaire à l'obtention des sorties du modèle réduit est généralement très inférieur au temps nécessaire à sa calibration. Il est donc important de noter que deux temps distincts de calcul interviennent lorsque l'on parle de réduction de modèle, *le temps de calibration* (ou d'*apprentissage*) et le *temps de restitution* (ou d'*utilisation*). L'emploi d'un modèle réduit est particulièrement attrayant dans les cas où :

- la modélisation initiale d'un système limite le nombre de données pouvant être générées,
- le coût expérimental est également trop élevé pour effectuer beaucoup de cas,
- la réponse dépend de plusieurs modèles,
- le modèle doit être utilisé dans une boucle d'optimisation.

Ce sont des cas fréquents dans le domaine de l'ingénierie et, en particulier, dans le domaine aéronautique, où le chargement de la structure d'un avion est obtenu à partir d'analyses multi-physiques et/ou d'expérimentations à échelle réelle qui engendrent des coûts importants. Les enjeux de minimisation de la masse jouant un rôle prépondérant dans la conception d'un avion, il paraît naturel, dans ce genre de situation, d'adopter un métamodèle qui encapsule les coûteuses données obtenues à des fins d'optimisation. Il existe d'ailleurs un grand nombre d'exemples dans lesquels des modèles réduits sont utilisés dans un but d'optimisation, dans le domaine aéronautique [19–22], mais également dans d'autres domaines [9, 23–29].

Ce chapitre dresse une liste non-exhaustive des principales méthodes connues et utilisées. Toutes présentent des avantages et des limites et il paraît difficile, voire même impossible, qu'*une seule* méthode puisse prétendre être supérieure à *toutes* les autres pour *tous* les cas d'applications. On peut en revanche établir, qu'en fonction de la nature des données à modéliser, certains métamodèles sont plus adaptés que d'autres. Le problème est qu'un ingénieur ne connaît pas *a priori* le comportement de l'espace de conception et il est donc nécessaire d'utiliser une méthode susceptible de donner une bonne approximation de cet espace malgré ce manque d'information. Le but est donc d'identifier une technique de réduction de modèle qui produit de bons résultats, sans être obligatoirement les meilleurs, mais sur un grand nombre de problèmes différents. La méthode développée dans ces travaux est basée sur l'utilisation des entités géométriques nommées splines de bases rationnelles non-uniforme mieux connues sous leur acronyme anglais NURBS (*Non-Uniform Rational Basis Splines*). Initialement développées dans le cadre de la modélisation géométrique des courbes et surfaces, leur formalisme est extensible au cas de dimensions quelconques et une méthode de réduction de modèle les utilisant a récemment été publiée par Turner [1] et utilisée dans un problème de caractérisation inverse des paramètres d'un matériau composite [30]. Elle présentait toutefois l'inconvénient de ne pas tirer parti de tous les paramètres régissant les entités NURBS, même si l'utilisateur n'avait pas de paramètres à fixer *a priori*, un grand nombre de paramètres ont été fixés à l'avance par des règles empiriques. Notre approche vise à éliminer ces règles en exploitant tous les paramètres régissant l'hypersurface NURBS sans introduire des hypothèses simplificatrices sur le choix de certaines variables de l'entité.

Les techniques de réduction de modèle présentées dans les prochaines sections, peuvent être classées selon trois catégories d'utilisation :

1. méthodes d'interpolation ;
2. méthodes d'approximation ;
3. méthodes de réduction d'ordre.

Les *méthodes d'interpolation* et les *méthodes d'approximation* permettent d'obtenir, à partir d'un ensemble de points connus $\{\mathbf{X}_i\}$, les valeurs en tous points non mesurés de l'espace $\mathbf{X}_0$. La différence principale entre ces deux méthodes est qu'en *interpolation*, la valeur en un point connu est la valeur réelle, ou mesurée, alors qu'en *approximation* la valeur en un point connu est

approchée. Les *méthodes de réduction d'ordre* permettent, quant à elles, d'obtenir une complexité de calcul inférieure à la complexité initiale.

Certaines des méthodes proposées sont capables de traiter les trois cas d'utilisation tandis que d'autres sont spécifiques à un domaine. Par exemple, les méthodes barycentriques sont des méthodes d'interpolation car elles ont été définies pour approximer la valeur de la fonction $\mathbf{z}$ en un point $\mathbf{X}_0$ non connu de l'espace, la valeur en un point connu étant directement la valeur mesurée [31,32]. Les méthodes de *krigeage* et les méthodes utilisant des *fonctions à base radiale* sont également des méthodes d'interpolation mais il est possible de les utiliser en approximation pour s'adapter à des données bruitées [9]. A l'inverse, les méthodes de *régression* sont souvent utilisées en approximation car il y a généralement plus de données que de coefficients à calculer [32]. Les méthodes basées sur l'apprentissage comme les *réseaux de neurones* ou les *machines à vecteurs de support* font partie des méthodes d'approximation car la phase d'apprentissage ne garantit pas l'interpolation des données [11, 33]. Les méthodes basées sur l'utilisation des *Splines* et plus généralement des *NURBS* sont, quant à elles, capables de traiter des problèmes d'interpolation et d'approximation [2] mais peuvent également être utilisées comme des méthodes de calcul directes, dans le cas de calculs isogéométriques par exemple [34]. En ce qui concerne les méthodes de réduction d'ordre, elles ont été développées dans le but de réduire la complexité de calcul, en générant une base de projection *a posteriori* de la solution pour la *décomposition en valeurs propres orthogonales* [35], et en utilisant une formulation à variables séparées pour la *décomposition propre généralisée* [36] générant ainsi une approximation de la solution pour un coût de calcul inférieur au modèle original. Il est donc naturel que ces méthodes puissent également être utilisées dans le cadre de l'approximation de données connues [7].

2.2 Méthodes d'interpolation et d'approximation

L'*interpolation* est une opération consistant à déterminer, à partir d'une série statistique succincte aux valeurs trop espacées, de nouvelles valeurs correspondant à un caractère intermédiaire pour lequel aucune mesure n'a été effectuée (Larousse).

Dans le domaine de l'ingénierie mécanique, les *méthodes d'interpolation* ont pour but d'obtenir la valeur $\mathbf{z}(\mathbf{X}_0)$ de la fonction $\mathbf{z}(\mathbf{X})$, $\mathbf{X} = \left(X^{(1)}, \ldots, X^{(N)}\right) \in \mathbb{R}^N$ en un point $\mathbf{X}_0$ à partir d'un ensemble de n valeurs connues de cette fonction $\{\mathbf{z}(\mathbf{X}_1), ..., \mathbf{z}(\mathbf{X}_n)\}$. La dimension de l'espace vectoriel contenant $\mathbf{X}$ est N. On note $\hat{\mathbf{z}}\left(\mathbf{X}_0\right)$ la valeur de la fonction qui interpole la valeur au point $\mathbf{X}_0$ et répondant à l'équation suivante :

$$\begin{cases} \hat{\mathbf{z}}\left(\mathbf{X}_0\right) = \mathbf{f}\left(\mathbf{z}(\mathbf{X}_1), ..., \mathbf{z}(\mathbf{X}_n)\right), \ \forall \mathbf{X}_0 \neq \{\mathbf{X}_1, ..., \mathbf{X}_n\}, \\ \hat{\mathbf{z}}\left(\mathbf{X}_i\right) = \mathbf{z}\left(\mathbf{X}_i\right), \ i = 1, ..., n. \end{cases} \tag{2.1}$$

La fonction $\mathbf{z}(\mathbf{X})$, $\mathbf{X} \in \mathbb{R}^N$ peut être une fonction scalaire ou vectorielle et, dans ce cas, nous notons M la dimension de cet espace vectoriel. Lorsque c'est le cas, nous considérons la fonction $\mathbf{z}$ comme une application $\mathbf{z} : \mathbb{R}^N \to \mathbb{R}^M$.

L'approximation est basée sur le même principe à la différence près que la fonction $\hat{z}(\mathbf{X})$ ne passe pas par les points $z(\mathbf{X}_i)$, $i = 1, ..., n$, mais à proximité de sorte que :

$$\hat{\mathbf{z}}\left(\mathbf{X}_0\right) = \mathbf{f}\left(\mathbf{z}(\mathbf{X}_1), ..., z(\mathbf{X}_n)\right), \ \forall \mathbf{X}_0 \in \mathbb{R}^N \tag{2.2}$$

En général, le but de l'approximation est de diminuer le nombre de paramètres nécessaires au calcul de $\hat{\mathbf{z}}$ en conservant un écart entre la valeur approximée $\hat{\mathbf{z}}(\mathbf{X}_i)$ et la valeur mesurée $\mathbf{z}(\mathbf{X}_i)$, $i = 1, ..., n$, *assez faible*, i.e. correspondant assez généralement à une erreur maximale d'approximation autorisée par l'utilisateur du métamodèle. Dans la littérature il est également possible de trouver les termes *méthode exacte* et *méthode approchée* pour parler des méthodes d'interpolation et d'approximation, respectivement.

Mathématiquement parlant, utiliser une méthode d'approximation pour un ensemble de données déterministes est discutable, pour ne pas dire incorrect, les erreurs étant dues à la précision du moyen ayant permis d'obtenir ces données. Par exemple, dans le cas d'une simulation numérique par éléments finis, ces erreurs sont déterminées par le nombre d'éléments du modèle. Elles sont systémiques et non pas aléatoires comme dans un ensemble de données non déterministes. Certains chercheurs considèrent donc que seules les méthodes d'interpolation sont appropriées pour modéliser des données déterministes [37]. Il existe cependant beaucoup de cas d'étude où l'utilisation d'un métamodèle d'approximation a démontré son utilité [9, 27, 38–42] même s'il modélisait des données déterministes. En effet, les simulations réalisées dans le domaine de l'ingénierie sont des approximations de la réalité et un métamodèle de ces simulations peut être interprété comme une approximation de l'approximation du système réel à l'étude. Par extension, il s'agit toujours de l'approximation du système réel [1]. Une méthode d'interpolation, utilisée pour modéliser une simulation du réel reste donc une approximation du système réel, au même titre qu'une méthode d'approximation. Il est donc plus pertinent de déterminer quelle approximation représente au mieux le système réel.

Toutes ces méthodes peuvent être classées en deux catégories principales : les méthodes *déterministes* et les méthodes *stochastiques* qui modélisent la fonction $\mathbf{z}(\mathbf{X})$, $\mathbf{X} \in \mathbb{R}^N$ comme la réalisation d'une fonction aléatoire :

$$\mathbf{Z}(\mathbf{X}) = \boldsymbol{\mu}(\mathbf{X}) + \boldsymbol{\varepsilon}(\mathbf{X}),\ \mathbf{X} \in \mathbb{R}^N, \tag{2.3}$$

où $\boldsymbol{\mu}(\mathbf{X})$ est la structure déterministe correspondant à l'espérance $E[\mathbf{Z}(\mathbf{X})]$, $\boldsymbol{\varepsilon}(\mathbf{X})$ est la structure aléatoire dont les propriétés dépendent de la méthode employée et $\mathbf{Z}(\mathbf{X})$ est la fonction aléatoire dont la réalisation est $\mathbf{z}(\mathbf{X})$ [32, 43, 44].

2.2.1 Méthodes déterministes barycentriques

Les méthodes d'interpolation de type barycentrique [45], aussi appelées *moyennes mobiles* [32] ou encore *approximation de Kernel* [31, 46] font partie des méthodes les plus intuitives. En effet, la valeur en un point inconnu $\mathbf{X}_0$ est prédite par une moyenne pondérée de l'ensemble des valeurs connues $\{\mathbf{z}(\mathbf{X}_1), ..., \mathbf{z}(\mathbf{X}_n)\}$:

$$\hat{\mathbf{z}}(\mathbf{X}_0) = \sum_{i=1}^{n} \lambda_i \mathbf{z}(\mathbf{X}_i), \tag{2.4}$$

avec λ_i le poids affecté à la valeur mesurée $\mathbf{z}(\mathbf{X}_i)$ respectant l'équation suivante :

$$\sum_{i=1}^{n} \lambda_i = 1, \tag{2.5}$$

afin de ne pas générer de distorsion par rapport à la valeur réelle. Le choix des pondérations varie d'une méthode à l'autre mais elles font pour la plupart intervenir la distance euclidienne $\|\mathbf{X}_i - \mathbf{X}_0\|$ entre le point $\mathbf{X}_i$ où la valeur de la fonction est connue et le point $\mathbf{X}_0$ de façon à accorder un poids λ_i plus grand aux points proches de $\mathbf{X}_0$. Le choix le plus classique pour la détermination de l'ensemble des $\{\lambda_i\}$ est de faire intervenir l'inverse de la distance euclidienne :

$$\lambda_i = \frac{\dfrac{1}{\|\mathbf{X}_i - \mathbf{X}_0\|^d}}{\displaystyle\sum_{j=1}^{n} \frac{1}{\|\mathbf{X}_j - \mathbf{X}_0\|^d}}, \quad d > 0, \tag{2.6}$$

avec d un paramètre à fixer a priori. Il permet de diminuer l'effet des valeurs éloignées de $\mathbf{X}_0$ sur son approximation. Il est également possible d'affecter un poids nul aux points les plus éloignés de $\mathbf{X}_0$. Dans ce cas, seul les points dans un certain voisinage $V(\mathbf{X}_0)$ sont pris en compte dans le calcul des coefficients λ_i et l'éq. (2.6) devient :

$$\lambda_i = \frac{\dfrac{1}{\|\mathbf{X}_i - \mathbf{X}_0\|^d}}{\displaystyle\sum_{j \in V(\mathbf{X}_0)} \frac{1}{\|\mathbf{X}_j - \mathbf{X}_0\|^d}}, \ \mathbf{X}_i \in V(\mathbf{X}_0). \tag{2.7}$$

En pratique, il est possible de choisir la taille du voisinage $V(\mathbf{X}_0)$ de deux manières. Soit un nombre de point n_0 est fixé *a priori* et limite le voisinage $V(\mathbf{X}_0)$ aux n_0 points les plus proches, au sens de la norme euclidienne, de $\mathbf{X}_0$. Soit un rayon r_0 est fixé lui aussi *a priori* et seul les points contenus dans l'hypersphère centrée en $\mathbf{X}_0$ et de rayon r_0 appartiennent au voisinage $V(\mathbf{X}_0)$. Les références [47–49] ont étudié les différents taux de convergence en fonction de la taille du voisinage et du nombre de points cibles disponibles de manière à choisir judicieusement ces deux paramètres. De part sa nature, l'interpolation de points $\mathbf{X}_0$ proches des bords est plus biaisée qu'à l'intérieur du domaine. Pour traiter ce problème, une méthode a été proposée par [50] permettant ainsi de conserver l'ordre de grandeur de biais et de variance de l'intérieur du domaine au niveau des bornes.

Bien qu'intuitive, cette méthode est surtout utilisée dans des applications dont l'espace des paramètres N est faible ($N < 3$) tels que l'interpolation spatiale [31], le traitement d'images [51] ou encore l'analyse de scènes dynamiques [52]. La raison est simple, le calcul points à points de la distance euclidienne peut s'avérer très coûteux lorsque la dimension N de l'espace ou le nombre de points mesurés n augmentent, de même que la recherche du voisinage $V(\mathbf{X}_0)$.

2.2.2 Méthodes déterministes d'interpolation par partitionnement de l'espace

Ces méthodes forment un cas particulier des méthodes barycentriques [32, 45] mais elles se distinguent par l'utilisation d'un découpage de l'espace permettant de déterminer à la fois les poids λ_i et le voisinage $V(\mathbf{X}_0)$. Il existe principalement deux méthodes de division de l'espace : par *polytopes* et par *simplexes*. Ces méthodes ayant été initialement développées dans le cadre de l'interpolation spatiale ($N = 2, 3$), il est beaucoup plus fréquent de trouver dans la littérature les

termes de découpage par *polygones* et *triangles* ($N = 2$) ou *polyèdres* et *tétraèdres* ($N = 3$). Le partitionnement par polytopes porte d'ailleurs différents noms dont les principaux sont *polygones de Thiessen*, *diagramme de Voronoi* et *tessellation de Dirichlet*. Le principe de ce découpage de l'espace est de définir pour chaque point connu $\mathbf{X}_i$ un polytope dans lequel tous les points sont plus proches de $\mathbf{X}_i$ que de n'importe quel autre point $\mathbf{X}_j$, $j \neq i$. La fig. 2.1 présente un exemple dans un cas de dimension $N = 2$ où les polygones de Thiessen en trait plein ont été obtenus à partir d'un ensemble de quinze points cibles. Par opposition, le découpage en simplexes partitionne l'espace par des N-simplexes dont les sommets sont les points connus $\mathbf{X}_i$. Il existe plusieurs méthodes pour sélectionner les sommets d'un même simplexe mais le plus connu est le critère de Delaunay qui utilise un partitionnement par polytope. Les points cibles partageant un hyperplan de leurs polytopes sont liés et forment une arrête du simplexe. La fig. 2.1 montre le découpage obtenu par le critère de Delaunay (trait pointillé) en dimension $N = 2$ à partir des polygones de Thiessen (trait plein) et de leurs quinze points cibles $\mathbf{X}_i$ associés. Comme les deux méthodes sont basées sur la tesselation par polytopes, il existe un grand nombre de références traitant de l'implémentation algorithmique du calcul de ce partitionnement, en dimension $N \leq 3$ [53–57] mais également en dimension quelconque [58,59]. Pour un approfondissement, le lecteur intéressé est renvoyé vers la référence [60].

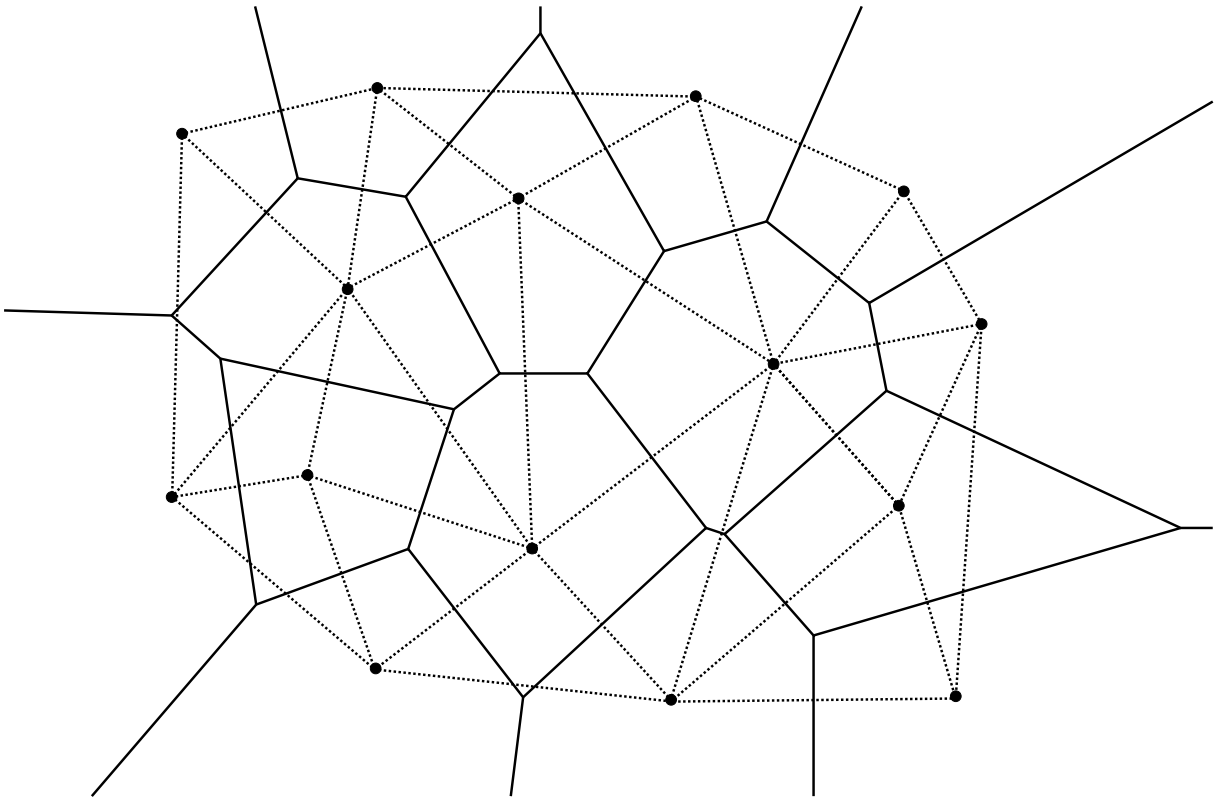


FIGURE 2.1 – Exemple de polygones de Thiessen (trait plein) et triangulation de Delaunay (trait pointillé)

Il existe plusieurs méthodes d'interpolation basées sur un partitionnement de l'espace dont la plus simple est celle du *plus proche voisin* qui affecte à l'ensemble des points $\mathbf{X}_0$ appartenant à un polytope, la valeurs du point connu $\mathbf{X}_i$ qui lui est associé. Elle est cependant peu utilisée en raison des problèmes de discontinuité qu'elle génère aux points appartenant aux hyperplans des polytopes. On lui préfère donc souvent la méthode *d'interpolation par voisinage naturel* formalisée par Sibson [61,62] et dont une revue récente des différentes variantes est disponible dans la référence [63]. Le principe de calcul des poids λ_i ainsi que du voisinage $V(\mathbf{X}_0)$ se fait en deux temps. L'espace est partitionné une première fois avec l'ensemble des points connus $\{\mathbf{X}_i\,,\ i = 1, ..., n\}$ et n'a ensuite plus besoin d'être recalculée. L'espace est ensuite de nouveau partitionné en incluant le point $\mathbf{X}_0$ dans cette tesselation, ce qui implique d'effectuer le découpage pour chaque point $\mathbf{X}_0$ que l'on souhaite calculer. Les deux partitionnements sont ensuite "superposés" afin de faire apparaître les intersections entre le polytope associé au point $\mathbf{X}_0$ et les polytopes de l'ensemble des points $\{\mathbf{X}_i\,,\ i = 1, ..., n\}$. Le volume de l'intersection entre le polytope associé à $\mathbf{X}_0$ et le polytope associé à $\mathbf{X}_i$ permet de calculer la pondération λ_i qui lui est associée de la manière suivante :

$$\lambda_i = \frac{\mathrm{Vol}\left(P_{\mathbf{X}_i} \cap P_{\mathbf{X}_0}\right)}{\sum_{j=1}^{n} \mathrm{Vol}\left(P_{\mathbf{X}_j} \cap P_{\mathbf{X}_0}\right)}, \tag{2.8}$$

où $\mathrm{Vol}\left(P_{\mathbf{X}_i} \cap P_{\mathbf{X}_0}\right)$ représente le volume de l'intersection entre le polytope $P_{\mathbf{X}_i}$ associé au point $\mathbf{X}_i$ et le polytope $P_{\mathbf{X}_0}$ associé au point $\mathbf{X}_0$. On comprend donc qu'un poids nul est affecté aux points $\mathbf{X}_i$ trop éloignés de $\mathbf{X}_0$, i.e. dont l'intersection $P_{\mathbf{X}_i} \cap P_{\mathbf{X}_0}$ est vide, sans avoir à fixer de paramètre *a priori* pour le calcul du voisinage $V\left(\mathbf{X}_0\right)$. La fig. 2.2 présente un exemple en dimension $N = 2$ où le polygone associé au point $\mathbf{X}_0$ (trait pointillé) a été superposé aux polygones de Thiessen (trait plein) afin de faire apparaître les aires a_i de leurs intersections. Dans cet exemple, la valeur $\hat{\mathbf{z}}\left(\mathbf{X}_0\right)$ en un point de l'espace est donnée par le relation suivante :

$$\hat{\mathbf{z}}\left(\mathbf{X}_0\right) = \sum_{i=1}^{n} \frac{a_i}{\sum_{j=1}^{n} a_j} \mathbf{z}\left(\mathbf{X}_i\right), \tag{2.9}$$

avec a_i l'aire de l'intersection entre le polygone associé au point $\mathbf{X}_0$ et le polygone associé au point $\mathbf{X}_i$.

Dans le cas d'un partitionnement par *simplexe*, le voisinage $V\left(\mathbf{X}_0\right)$ est déterminé par le simplexe $S_{\mathbf{X}_0}$ dans lequel il se trouve. En effet, seuls les points de ce simplexe interviennent dans le calcul des pondérations λ_i par la relation :

$$\lambda_i = \frac{\mathrm{Vol}\left(S_i\right)}{\sum_{j \in S_{\mathbf{X}_0}} \mathrm{Vol}\left(S_j\right)}, \tag{2.10}$$

où $\mathrm{Vol}\left(S_i\right)$ est le volume du simplexe S_i dont les sommets sont l'ensemble des points $\{\mathbf{X}_j\,,\ j \in S_{\mathbf{X}_0},\ j \neq i\}$ et le point $\mathbf{X}_0$. La fig. 2.3 présente un exemple en dimension $N = 2$ d'interpolation par voisinage naturel à partir d'une triangulation de Delaunay. Dans cet exemple, la valeur $\hat{\mathbf{z}}\left(\mathbf{X}_0\right)$ est donnée par la relation suivante :

$$\hat{\mathbf{z}}\left(\mathbf{X}_0\right) = \sum_{i=1}^{n} \frac{a_i}{\sum_{j=1}^{n} a_j} \mathbf{z}\left(\mathbf{X}_i\right), \tag{2.11}$$

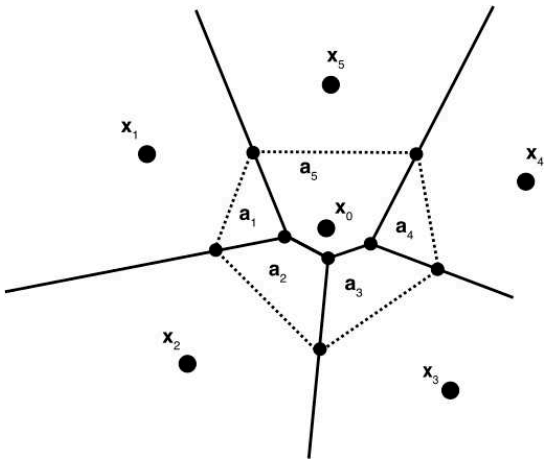


FIGURE 2.2 – Exemple de pondération par voisinage naturel en dimension $N = 2$ avec des polygones de Thiessen

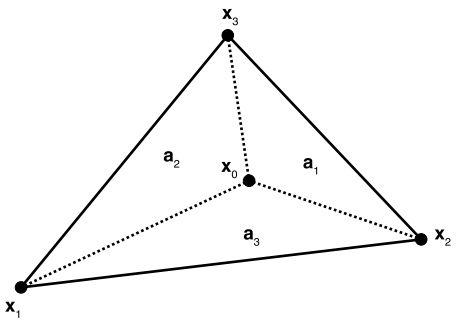


FIGURE 2.3 – Exemple de pondération par voisinage naturel en dimension $N = 2$ avec une triangulation de Delaunay

avec a_i l'aire du triangle dont les sommets sont les points $\{\mathbf{X}_0\,,\,\mathbf{X}_j,\ j \neq i\}$.

De même que pour la famille complète des méthodes barycentriques, les techniques de partitionnement sont essentiellement utilisées en interpolation spatiale, dont notamment le traitement de données météorologiques [64, 65] ou de données géostatistiques [66, 67]. En effet, malgré la simplicité de ces méthodes, la complexité du calcul de la tesselation croît grandement avec la dimension N et certaines méthodes impliquent de calculer un nouveau découpage pour chaque valeur $\mathbf{z}\,(\mathbf{X}_0)$ à interpoler. Notons cependant que l'avantage principal de ces méthodes réside dans l'absence de paramètre à fixer *a priori* par l'utilisateur.

2.2.3 Méthode déterministe : régression

Les méthodes de régression, aussi connues sous le nom de *surfaces de réponses*, *surfaces de tendances* [32] ou *Response Surface Models* en anglais, sont directement issues des techniques

de plans d'expérience et furent initialement introduites comme des méthodes d'ajustement de courbes (*curve fitting*). La méthode la plus classique utilise une forme polynomiale des variables indépendantes :

$$\hat{z}\left(\mathbf{X}\right) = \sum_{i_1+\ldots+i_N \leq p} \beta_{i_1,\ldots,i_N} \left(x^{(1)}\right)^{i_1} \times \ldots \times \left(x^{(N)}\right)^{i_N}, \tag{2.12}$$

où p est le degré maximale du polynôme et l'ensemble des $\{\beta_{i_1,\ldots,i_N}\}$ sont les coefficients du polynôme à déterminer. Le degré p doit être fixé *a priori* par l'utilisateur et celui-ci détermine le nombre de coefficients à calculer et donc le nombre minimal de points nécessaires. Il existe plusieurs méthodes permettant d'obtenir les coefficients $\beta_{i_1,\ldots,i_N}$ comme le maximum de vraisemblance [68–70] mais la plus utilisée est la méthode des moindres carrés. Il est d'ailleurs aisé de mettre l'éq. (2.12) sous la forme matricielle suivante :

$$\mathbf{Z} = \mathbf{M}\boldsymbol{\beta}, \tag{2.13}$$

avec $\mathbf{Z}$ un vecteur de dimension $n \times 1$ des valeurs de la fonction $z\left(\mathbf{X}_i\right)$, $i = 1, \ldots, n$, $\boldsymbol{\beta}$ est un vecteur de dimension $P \times 1$ des coefficients $\beta_{i_1,\ldots,i_N}$ et $\mathbf{M}$ est une matrice de dimension $n \times P$ contenant les termes des différentes séries de puissance $\prod_{i_1+\ldots+i_N \leq p} \left(x^{(1)}\right)^{i_1} \times \ldots \times \left(x^{(N)}\right)^{i_N}$ avec P le nombre de coefficients à calculer défini par la relation suivante :

$$P = (p+1)^N. \tag{2.14}$$

Dans le cas où le nombre de points mesurés n est égal au nombre de coefficients P à déterminer, la matrice $\mathbf{M}$ est carrée et potentiellement inversible, ce qui permet d'obtenir une interpolation des points cibles. En général, n est supérieur à P et il est alors nécessaire d'inverser $\mathbf{M}$ à l'aide de la *pseudo-inverse*. Ceci revient à calculer les coefficients $\beta_{i_1,\ldots,i_N}$ par la méthode des moindres carrés. Le polynôme obtenu n'interpolera alors pas les points cibles mais les approximera.

Du point de vue de l'utilisateur du métamodèle, le seul paramètre à fixer *a priori* est le degré maximal des polynômes p mais celui-ci pilote en grande partie le coût de calcul de la pseudo-inverse de $\mathbf{M}$. En effet, l'inversion de la matrice est généralement considérée comme étant $\mathcal{O}\left(P^3\right)$ [71] et la dimension N de l'espace n'étant pas une variable, seul p pilote la complexité du calcul. Il est également important de noter que dans le cas de sorties multiples, i.e. une fonction $\mathbf{z}(\mathbf{X})$ vectorielle, il est nécessaire de calculer les P coefficients $\beta_{i_1,\ldots,i_N}$ pour chaque sortie. Notons également que les surfaces de réponse sont capables d'extrapoler les valeurs en dehors des limites fixées par l'ensemble des points cibles avec des aptitudes de prédiction correctes [72]. Il est rare de trouver des applications où p est supérieur à 2 à cause de l'effet *vaguelette* généré par des polynômes de degrés supérieurs. A ceci s'ajoute le fait que tous les points cibles ont la même influence sur tout le champ. Une conséquence directe est que cette méthode n'est pas efficace dans le cas où la fonction $z(\mathbf{X})$ est fortement non-linéaire localement. Des approches locales ont été développées pour résoudre ces problèmes [73, 74], basées sur l'utilisation de multiples surfaces de réponse locales, mais conserver la continuité et la dérivabilité entre les différents morceaux est loin d'être trivial. De plus, le métamodèle obtenu est dépendant du système de coordonnées utilisé, ce qui peut conduire à des résultats incohérents en changeant le système de coordonnées ou son orientation.

2.2.4 Méthodes stochastiques de Krigeage

Cette méthode doit son nom à l'ingenieur minier sud-africain Krige [75] qui en est le précurseur. Le terme *krigeage* et le formalisme de la méthode sont toutefois dus au français Matheron [76, 77] qui en a assuré le développement. Le fondement de cette méthode est de prendre en considération la structure de dépendance spatiale des données. Elle est d'ailleurs la première à en avoir tenu compte. La fonction $\hat{z}(\mathbf{X})$ est donnée par la relation suivante :

$$\hat{z}(\mathbf{X}_0) = a + \sum_{i=1}^{n} \lambda_i z(\mathbf{X}_i), \tag{2.15}$$

où les coefficients a et λ_i sont obtenus de manière à ce que la prédiction ne soit pas biaisée avec une variance minimale. Comme il s'agit d'une méthode stochastique, la fonction $z(\mathbf{X})$ est vue comme la réalisation d'une variable aléatoire régie par l'éq. (2.3) où $\varepsilon(\mathbf{X})$ est la structure aléatoire stationnaire, d'espérance nulle et possédant une structure de dépendance à déterminer. Il existe plusieurs formes de krigeage, dépendant de la forme donnée à la structure déterministe $\mu(\mathbf{X})$, dont principalement :

- Le krigeage simple où $\mu(\mathbf{X}) = k$ est une constante connue.
- Le krigeage ordinaire où $\mu(\mathbf{X}) = \mu$ est une constante inconnue.
- Le krigeage universel où $\mu(\mathbf{X}) = \sum_j \beta_j f_j(\mathbf{X})$ est une combinaison linéaire de coefficients β_j de fonctions $f_j(\mathbf{X})$ de la position.

Initialement développée dans un cadre géostatistique [44, 78], l'intérêt pour cette méthode a vu le développement d'une branche spécifique à la réduction de modèle, connue sous le nom de DACE [8, 79, 80] (*Design and Analysis of Computer Experiments*), dont la formulation bifurque légèrement du cadre initial [9]. Nous présentons bien évidemment ici le formalisme utilisé dans le cadre de la réduction de modèle.

La modélisation par krigeage, passe par l'étude de la structure de dépendance de la fonction aléatoire $\varepsilon(\mathbf{X})$. Pour cela, la covariance de la variable aléatoire $Z(\mathbf{X})$ est définie de la manière suivante :

$$\mathrm{Cov}\left(Z(\mathbf{X}_i), Z(\mathbf{X}_0)\right) = \sigma_z^2 R(\mathbf{X}_i, \mathbf{X}_0), \tag{2.16}$$

où σ_z^2 est un facteur d'échelle, appelé variance du processus, qui peut s'adapter aux données traitées et $R(\mathbf{X}_i, \mathbf{X}_0)$ la fonction de corrélation spatiale, ou SFC (*Spatial Correlation Function*), entre les points $\mathbf{X}_i$ et $\mathbf{X}_0$. Le choix de la SFC détermine la manière dont les données vont être interpolées et il en existe un grand nombre comme le modèle *pépitique*, le modèle *linéaire à palier*, le modèle *exponentiel* ou encore le modèle *gaussien* [43, 44]. Cependant, dans le domaine de l'ingénierie, le modèle le plus utilisé pour le cas où $N = 1$ est [1, 9] :

$$R(X_i, X_0) = e^{-\theta |X_i - X_0|^p}, \tag{2.17}$$

avec $\theta > 0$, un paramètre définissant le rayon d'influence des données et $0 < p \leq 2$ un paramètre liée au *lissage* du modèle. On remarque, d'après cette équation, que plus les points X_i et X_0 sont éloignés l'un de l'autre, plus l'influence de X_i sur la prédiction de X_0 est diminuée. Le paramètre θ permet de personnaliser la rapidité avec laquelle l'influence décroit en fonction de la distance.

Pour de petites valeurs de θ, même les points éloignés continuent à influencer la prédiction car ils restent très corrélés. A l'inverse, pour des grandes valeurs de θ, seuls les points très proches de la valeur prédite auront de l'influence. Concernant le paramètre p, plus sa valeur est élevée, plus la réponse obtenue sera lisse. Avec cette formulation, $R(X_i, X_i) = 1$, ce qui implique que le métamodèle obtenu réalise une interpolation des données. Il existe toutefois des techniques permettant de ne pas passer par les points mesurés afin d'approximer des données bruitées par exemple [9]. L'éq. (2.17) est valable pour le cas où $N = 1$. Pour étendre cette formulation à une dimension N quelconque, une pratique courante est de multiplier les fonctions de corrélation de chaque dimension. Cette règle, connue sous le nom de règle du produit de corrélations ou *product correlation rule*, exprime la SCF de la manière suivante :

$$R(\mathbf{X}_i, \mathbf{X}_0) = \prod_{d=1}^{N} e^{-\theta_d |X_i^{(d)} - X_0^{(d)}|^{p_d}}, \tag{2.18}$$

où d représente la direction de l'espace, $\mathbf{X}_i$, $i = 1, ..., n$, est un point mesuré de l'espace, $\mathbf{X}_0$ est le point auquel on réalise la prédiction, θ_d est le rayon d'influence dans la direction d, p_d est le paramètre de lissage dans la direction d et $X_i^{(d)}$ est la d-ième coordonnée du point $\mathbf{X}_i$. Notons que cette méthode permet de sélectionner une SCF différente pour chaque direction d de l'espace. Il est possible notamment de choisir un paramètre θ_d et un paramètre p_d pour chaque direction $d = 1, ..., N$. Cependant, certains auteurs suggèrent que l'utilisation d'un seul couple de paramètre θ et p donne déjà de bons résultats.

Les SCF étant définies pour chaque direction, la prédiction en un point $\mathbf{X}_0$ est obtenu par la relation suivante :

$$\hat{z}(\mathbf{X}_0) = \mathbf{f}_{\mathbf{X}_0}^T \boldsymbol{\beta} + \mathbf{r}_{\mathbf{X}_0}^T \mathbf{R}^{-1} (\mathbf{z} - \mathbf{F}\boldsymbol{\beta}), \tag{2.19}$$

où $\boldsymbol{\beta}$ est donné par la relation :

$$\boldsymbol{\beta} = \left(\mathbf{F}^T \mathbf{R}^{-1} \mathbf{F}\right)^{-1} \mathbf{F}^T \mathbf{R}^{-1} \mathbf{z} \tag{2.20}$$

avec $\mathbf{f}_{\mathbf{X}_i}$ le vecteur $k \times 1$ des termes polynomiaux de la structure déterministe tel que :

$$\mathbf{f}_{\mathbf{X}_i} = \begin{pmatrix} f_1(\mathbf{X}_i) & \cdots & f_k(\mathbf{X}_i) \end{pmatrix}^T, \tag{2.21}$$

β_j les poids associés aux fonctions $f_j(\mathbf{X})$ regroupés dans le vecteur de dimension $k \times 1$:

$$\boldsymbol{\beta} = \begin{pmatrix} \beta_1 & \cdots & \beta_k \end{pmatrix}^T. \tag{2.22}$$

et $\mathbf{F}$ est la matrice $n \times k$ contenant les termes polynomiaux des n différents points mesurés :

$$\mathbf{F} = \begin{pmatrix} \mathbf{f}_{\mathbf{X}_1}^T \\ \vdots \\ \mathbf{f}_{\mathbf{X}_n}^T \end{pmatrix}. \tag{2.23}$$

Le vecteur $\mathbf{z}$ de dimension $n \times 1$ regroupe les différentes réalisations de $Z(\mathbf{X}_i)$, $i = 1, ..., n$,

$$\mathbf{z} = \begin{pmatrix} z(\mathbf{X}_1) & \cdots & z(\mathbf{X}_n) \end{pmatrix}^T, \tag{2.24}$$

tandis que le vecteur $\mathbf{r}_{\mathbf{X}_0}$ de dimension $n \times 1$ est le vecteur des corrélations spatiales entre le point $\mathbf{X}_0$ et les points mesurés $\mathbf{X}_i$, $i = 1, ..., n$,

$$\mathbf{r}_{\mathbf{X}_0} = \begin{pmatrix} R(\mathbf{X}_1, \mathbf{X}_0) & \cdots & R(\mathbf{X}_n, \mathbf{X}_0) \end{pmatrix}^T, \tag{2.25}$$

et la matrice carré $\mathbf{R}$ de dimension n est la matrice des corrélations spatiales dont la ligne i et la colonne j est définie par la SCF entre les points $\mathbf{X}_i$ et $\mathbf{X}_j$,

$$\mathbf{R} = \begin{pmatrix} R(\mathbf{X}_1, \mathbf{X}_1) & \cdots & R(\mathbf{X}_1, \mathbf{X}_n) \\ \vdots & \ddots & \vdots \\ R(\mathbf{X}_n, \mathbf{X}_1) & \cdots & R(\mathbf{X}_n, \mathbf{X}_n) \end{pmatrix}. \tag{2.26}$$

En pratique, plusieurs études, dont [9], ont suggéré que le krigeage ordinaire était suffisant dans un grand nombre de cas et simplifie grandement l'éq. (2.19) puisque $\mathbf{f}_{\mathbf{X}} = 1$ et $\mathbf{F} = \mathbf{1}$ où $\mathbf{1}$ est un vecteur de dimension $n \times 1$ ne contenant que des 1. Dans ce cas, l'éq. (2.19) devient :

$$\hat{z}(\mathbf{X}_0) = \beta + \mathbf{r}_{\mathbf{X}_0}^T \mathbf{R}^{-1} (\mathbf{z} - \mathbf{1}\beta), \tag{2.27}$$

avec

$$\beta = \left(\mathbf{1}^T \mathbf{R}^{-1} \mathbf{1}\right)^{-1} \mathbf{1}^T \mathbf{R}^{-1} \mathbf{z} \tag{2.28}$$

où β est maintenant un terme constant.

En pratique, le choix des paramètres θ_d et p_d peut se faire par deux méthodes : *maximum de vraisemblance* ou *validation croisée*. Cependant, trouver un ensemble de paramètres optimaux par l'une ou l'autre de ces méthodes est grandement limité par le coût de calcul de $\mathbf{R}$ qui peut devenir très important lorsque le nombre d'échantillons n augmente. De plus, $\mathbf{R}$ est sensible à la distribution des données et des points trop proches peuvent engendrer des singularités et la rendre non-inversible [9]. C'est pourquoi, en général, des hypothèses simplificatrices sont faites, comme notamment fixer $\forall d$, $p_d = 2$ et $\forall d$, $\theta_d = \theta$. De cette manière, seule la valeur du paramètre θ est à déterminer. Enfin, même s'il est possible d'extrapoler des données avec le krigeage, le consensus général est d'éviter de l'utiliser dans ce cas là, l'extrapolation étant très sensible à la forme de la tendance déterministe (krigeage simple, ordinaire, ou universel) , alors que l'effet est souvent négligeable en interpolation.

2.2.5 Méthodes stochastiques utilisant les fonctions à base radiale

Une fonction à base radiale, ou RBF pour *Radial Basis Function*, est une fonction ϕ à valeurs réelles qui ne dépend que de la distance $r_i = \|\mathbf{X}_0 - \mathbf{X}_i\|$ du point considéré $\mathbf{X}_0$ par rapport à un autre point $\mathbf{X}_i$, de telle sorte que $\phi(r_i) = \phi(\|\mathbf{X}_0 - \mathbf{X}_i\|)$. La norme employée est souvent la norme euclidienne mais d'autres mesures de distance sont envisageables. Les métamodèles utilisant de telles fonctions font partie des modélisations stochastiques et sont intimement liés au krigeage et aux modèles basés sur les splines. La référence [81] utilise d'ailleurs des métamodèles basés sur les RBF et sur le krigeage dans le cadre d'optimisation aérodynamique. Le principe est d'obtenir la valeur $\hat{z}(\mathbf{X}_0)$ à partir d'une combinaison linéaire, non pas des valeurs mesurées, mais des RBFs de ces points :

$$\hat{z}(\mathbf{X}_0) = \sum_{i=1}^{n} \omega_i \phi(r_i), \tag{2.29}$$

où ω_i est le poids associé à la RBF du i-ème point mesuré $\phi(r_i)$ avec $r_i = \|\mathbf{X}_i - \mathbf{X}_0\|$. Les RBFs peuvent avoir plusieurs formes dont les plus courantes sont répertoriées dans le tableau 2.1 [82–85]. Il est possible de remarquer, d'après celui-ci, qu'un paramètre c, relatif à un rayon d'action de la fonction, doit être fixé afin de définir complètement la RBF. La référence [86] propose une méthode de sélection de ce paramètre.

Name	$\phi(r)$
Gaussienne	$\phi(r) = e^{\frac{r^2}{c^2}},\ c > 0$
Multiquadratique	$\phi(r) = \sqrt{r^2 + c^2},\ c \geq 0$
Quadratique inverse	$\phi(r) = \frac{1}{r^2 + c^2},\ c > 0$
Multiquadratique inverse	$\phi(r) = \frac{1}{\sqrt{r^2 + c^2}},\ c > 0$
Spline polyharmonique (p impair)	$\phi(r) = r^p$
Spline polyharmonique (p pair)	$\phi(r) = r^p \ln(r)$
Spline de type plaque mince	$\phi(r) = r^2 \ln(r)$

Tableau 2.1 – Exemples classiques de RBF

La RBF étant sélectionnée, les coefficients ω_i sont calculés à partir des conditions d'interpolation,

$$\hat{z}(\mathbf{X}_i) = z(\mathbf{X}_i), \quad i = 1, ..., n, \tag{2.30}$$

qui peuvent être mises sous forme matricielle,

$$\mathbf{A}\boldsymbol{\omega} = \mathbf{z}, \tag{2.31}$$

avec $\boldsymbol{\omega} = (\omega_1 \ \cdots \ \omega_n)^T$ le vecteur contenant les coefficients ω_i, $i = 1, ..., n$, $\mathbf{z} = (z(\mathbf{X}_1) \ \cdots \ z(\mathbf{X}_n))^T$ le vecteur contenant les réalisations de la variable aléatoire $Z(\mathbf{X})$ et $\mathbf{A}$ la matrice symétrique dont la i-ème ligne et j-ème colonne contient la valeur de la RBF $\phi(\|\mathbf{X}_i - \mathbf{X}_j\|)$,

$$\mathbf{A} = \begin{pmatrix} \phi(\|\mathbf{X}_1 - \mathbf{X}_1\|) & \cdots & \phi(\|\mathbf{X}_1 - \mathbf{X}_n\|) \\ \vdots & \ddots & \vdots \\ \phi(\|\mathbf{X}_n - \mathbf{X}_1\|) & \cdots & \phi(\|\mathbf{X}_n - \mathbf{X}_n\|) \end{pmatrix} \tag{2.32}$$

La difficulté principale pour l'utilisateur de la méthode est qu'il est rarement possible de connaître, à l'avance, quelle RBF utiliser pour générer le métamodèle et, comme dans le cas du krigeage, un trop grand nombre de points cibles peut engendrer des difficultés dans l'inversion de la matrice $\mathbf{A}$. De plus, les RBFs ayant un comportement radial symétrique, le métamodèle qu'elle génère n'est généralement pas adapté pour représenter des données discontinues [1]. De plus, puisque tous les points mesurés impactent le modèle RBF, son comportement, comme pour le krigeage, n'est pas local et l'ajout de nouveaux points dans la base de données nécessite de recalculer tout le modèle. Même si, par définition, le métamodèle généré réalise une interpolation des données, il est possible de créer un métamodèle RBF d'approximation par l'addition d'un

paramètre dont le comportement est similaire à l'*effet pépite* pour le krigeage. L'éq. (2.29) devient alors :

$$\hat{z}(\mathbf{X}_0) = (1 - c_0) \sum_{i=1}^{n} \omega_i \phi \left(\|\mathbf{X}_i - \mathbf{X}_0\| \right), \tag{2.33}$$

où, pour des faibles valeurs du paramètre de pépite ($c_0 \leq 0.001$), on obtient un métamodèle RBF d'approximation.

Une récente extension des RBFs a été développée par Mullur [87,88] sous l'acronyme E-RBF pour *Extended Radial Basis Function*. L'idée est d'ajouter des contraintes au métamodèle afin d'assurer certaines propriétés comme, par exemple, la convexité dans des cas où l'on sait que ces propriétés existent dans l'ensemble des données. Ce comportement est obtenu en utilisant plusieurs fonctions de base, à comportement radial ou non, au lieu de n'en utiliser qu'une seule. Sur les cas étudiés, Mullur a conclu que les E-RBFs étaient au moins aussi performantes, voire meilleures que les simples RBFs. L'idée d'utiliser plusieurs fonctions pour construire le métamodèle des E-RBFs fait le lien avec les méthodes heuristiques basées sur les réseaux de neurones que nous allons présenter maintenant.

2.2.6 Réseaux de neurones

Les réseaux de neurones artificiels sont des métamodèles bio-inspirés du fonctionnement du cerveau. Ils ne peuvent cependant pas prétendre égaler les performances d'un réseau biologique, essentiellement parce que nous ne maîtrisons pas encore parfaitement la complexité des neurones humains et de leurs interactions. De plus, il est pour l'instant impossible de réaliser un modèle réduit dont les nombres de neurones et d'interconnexions seraient comparables à ceux d'un cerveau humain [11,33]. Un réseau de neurones est composé d'une couche d'entrée, formée par les valeurs des différentes coordonnées de $\mathbf{X}$, d'une couche finale qui compose les sorties du modèle et de n_l différentes couches intermédiaires ($n_l \geq 0$). Chacune des couches est à son tour composée d'un certain nombre de neurones. La première étant composée des entrées du métamodèle, elle compte nécessairement N neurones correspondant aux N coordonnées de $\mathbf{X}$ tandis que la dernière possède M neurones correspondant aux M coordonnées $z^{(j)}(\mathbf{X})$, $j = 1, ..., M$, de la fonction vectorielle $\mathbf{z}(\mathbf{X})$. La fig. 2.4 présente un exemple de réseau de neurones artificiel contenant deux couches intermédiaires de n_{c_1} et n_{c_2} neurones respectivement. En général, même s'il est possible d'obtenir plusieurs sorties avec un métamodèle par réseau de neurones, les résultats sont généralement moins bons qu'avec un réseau différent par sortie.

Les neurones des couches intermédiaires sont composés de fonctions fixées *a priori* par l'utilisateur et chaque connexion représente un poids à calculer comme le montre la fig. 2.5. Ils sont obtenus dans la phase d'*apprentissage*. Contrairement aux métamodèles utilisant une seule RBF, il est possible, grâce à un réseau, d'utiliser plusieurs fonctions différentes avec des pondérations indépendantes [89]. Il n'est cependant pas obligatoire de prendre seulement des fonctions ayant un comportement radial. La construction d'un réseau de neurones passe par trois étapes : la sélection des fonctions à inclure, la sélection de l'architecture et l'entrainement, ou l'apprentissage, du réseau afin d'établir les poids de chacune des interconnexions. Aucune de ces trois tâches n'est triviale. Par exemple, en augmentant le nombre de fonctions de base, les possibilités d'architecture pour disposer les neurones croient exponentiellement. Mais même avec un faible

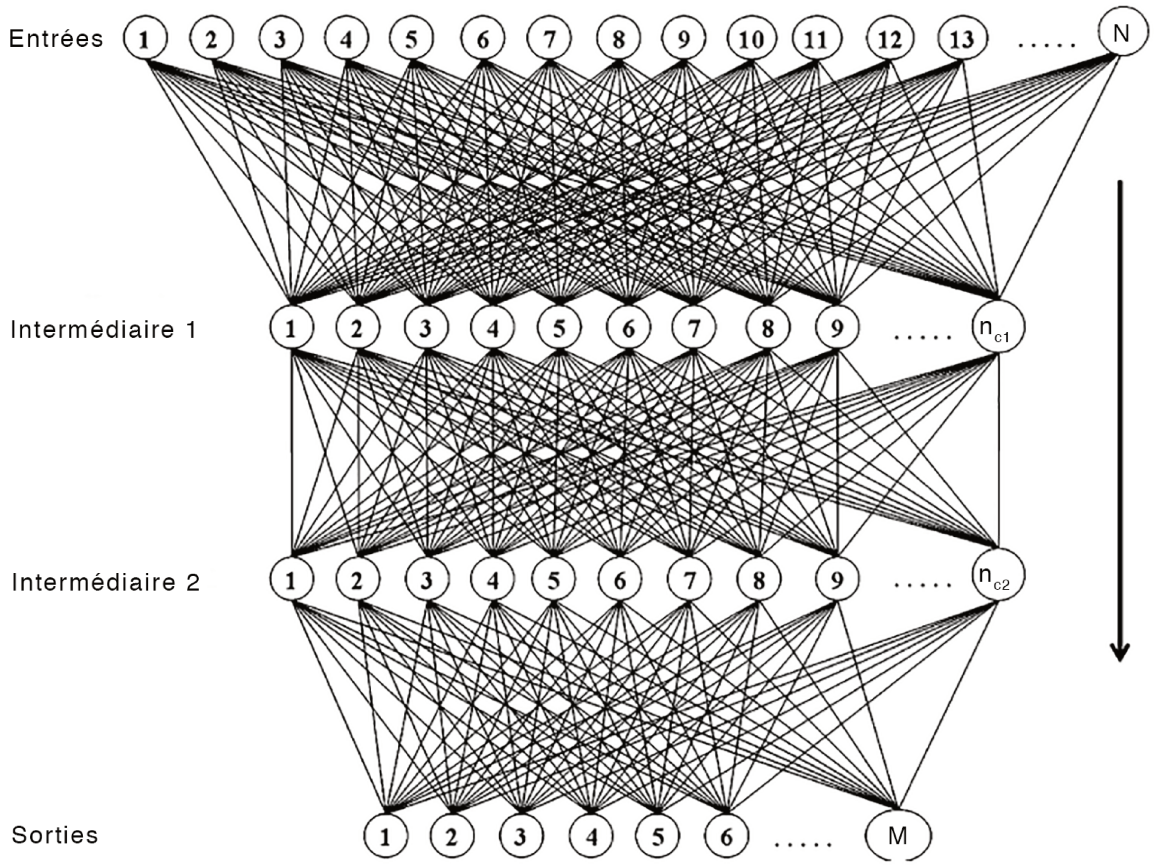


FIGURE 2.4 – Exemple d'architecture de réseau de neurones artificiel avec 2 couches intermédiaires

nombre de fonctions, l'architecture n'est pas nécessairement aussi simple que celle proposée par la fig. 2.5. Il est possible de les disposer en plusieurs couches et non pas en une seule (cf. fig. 2.4), et des informations peuvent être renvoyées d'une couche inférieure vers une ou des couches supérieures. Malgré l'existence de méthodes permettant d'adapter l'architecture automatiquement en fonction du problème [90], le choix des fonctions et de l'architecture passe souvent par une phase d'essai/erreur et l'utilisation des réseaux de neurones semble réservée aux ingénieurs spécialement entrainés pour cela [1]. C'est pour cela qu'en pratique, même s'il existe des exemples d'application de réseaux utilisés comme métamodèles [91–93] et qu'il existe des méthodes pour le choix de l'apprentissage [94], il est rare d'utiliser plus d'une couche intermédiaire et cette couche ne possède généralement pas un nombre élevé de neurones car la sélection de ces paramètres, pour des architectures complexes, requiert une bonne connaissance des réseaux de neurones et de leurs comportements.

La phase d'apprentissage peut également s'avérer problématique. En effet, il est par exemple

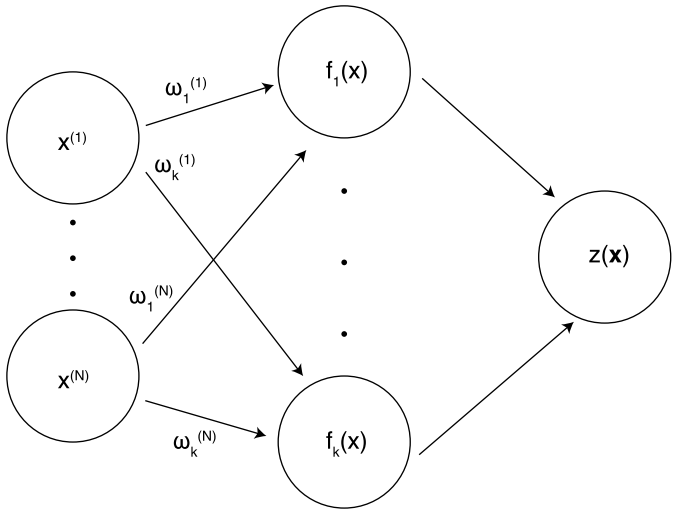


FIGURE 2.5 – Exemple de réseau de neurones contenant k fonctions et 1 couche intermédiaire

très compliqué, voire impossible, d'obtenir des métamodèles de solutions analytiques lorsque la complexité du réseau augmente, par manque de données pour l'apprentissage. Des méthodes d'optimisation non-linéaires sont souvent utilisées pour obtenir une solution mais celle-ci est généralement non-unique et adaptée aux données fournies qui peuvent ne pas être adéquate pour entrainer complètement le métamodèle [90]. Pour toutes ces raisons, définir les fonctions de base, optimiser l'architecture du réseau et l'entrainer peut prendre un temps non négligeable. De plus, il n'y a aucune garantie que les données d'apprentissage soient adéquates, que l'optimisation non-linéaire aboutisse à une solution optimale ou que la solution calculée sera une bonne prédiction [1]. S'ajoute à cela une tendance à obtenir des métamodèles souffrant du phénomène d'*over-fitting*. Ceci arrive lorsque un métamodèle représente précisément les données qui ont permis de l'obtenir mais donne une très mauvaise prédiction en dehors de celles-ci. Ce problème ne semble pas traité dans la littérature car les partisans des réseaux de neurones n'apprécient pas ce phénomène et par conséquent éludent le problème [95].

Malgré ces défauts, il serait mal venu de considérer que les réseaux de neurones n'ont pas leur place parmi les méthodes de réduction de modèle. En effet, elles sont, à l'heure actuelle, les seules méthodes à pouvoir traiter des problèmes ayant des milliers de variables d'entrées. Cependant, la nécessité de compétences spécifiques aux réseaux de neurones pour être implémentés correctement en fait une méthode peu versatile et difficilement utilisable par un utilisateur non-initié. Le lecteur souhaitant approfondir le sujet est renvoyé vers les références [11, 90, 96, 97].

2.2.7 Vector Support Machine

Les séparateurs à vaste marge, ou *Vector Support Machines*, sont des algorithmes d'apprentissage initialement développés pour la discrimination, i.e. la prévision d'une variable binaire, par Vapnik [98]. Ils ont ensuite été généralisés au cas de variables qualitatives et sont basés sur l'utilisation d'une fonction noyau permettant une transformation non-linéaire des données d'en-

trées vers un espace intermédiaire (*feature space*) de plus grande dimension. L'idée générale est de ramener le problème de discrimination à un problème linéaire de recherche d'un hyperplan de marge optimale. Cela se traduit par la recherche d'une fonction, dans l'espace intermédiaire, qui maximise l'écart de l'erreur entre cette fonction et les points cibles de la base de données [99]. La fonction, ainsi obtenue, est une fonction linéaire dans un espace non-linéaire, permettant de représenter des fonctions fortement non-linéaires par des expressions linéaires [100]. En fonction du choix de la fonction noyau, il est possible d'obtenir des métamodèles équivalents aux RBFs, aux surfaces de réponses ou à des réseaux de neurones à trois couches [100].

Les VSMs sont particulièrement efficaces pour les problèmes de grandes dimensions car une dimension N importantes n'implique pas que l'espace intermédiaire soit de grande dimension. En fait, l'algorithme est plus pénalisé par le nombre d'observations, i.e. le nombre de vecteurs supports potentiels, que par le nombre de variables d'entrées [101]. Malgré leur utilisation dans de nombreux domaines tels que les problèmes de classification [102–105] et de caractérisation de texte [106], la reconnaissance faciale [107], la reconnaissance de forme [108], l'analyse de composantes principales [100], les analyses de régression [99, 109] ou encore la modélisation de données aérodynamiques [110], les SVMs n'ont pas démontré de capacité à surpasser les autres techniques existantes ou à résoudre des problèmes insolvables auparavant [100].

Leur intérêt principal est, comme pour les réseaux de neurones, de pouvoir traiter des problèmes ayant un grand nombre de variables d'entrée. Cependant, les paramètres à fixer *a priori* par l'utilisateur requièrent une certaine connaissance de la méthode pour être implémentée correctement, même si le nombre de paramètres est bien moins important que pour les réseaux de neurones. De nombreuses références pertinentes existent et le lecteur intéressé par le domaine est renvoyé vers les références [111–114].

2.2.8 Méthodes déterministes utilisant les splines

Contrairement aux méthodes qui interpolent les valeurs *point à point*, l'idée générale derrière l'interpolation ou l'approximation par splines est d'ajuster une hypersurface passant par ou proche des points cibles $\mathbf{X}_i$. Cette surface a également comme propriété d'être la plus *lisse* possible et on les retrouve souvent dans la littérature sous le nom de *splines de lissage*. Ces deux conditions sont réunies dans un problème de minimisation d'une fonction $E(\hat{z})$:

$$E\left(\hat{z}\right) = \sum_{i=1}^{n} \left[z(\mathbf{X}_i) - \hat{z}(\mathbf{X}_i)\right]^2 + \rho I(\hat{z}), \tag{2.34}$$

avec ρ un paramètre de lissage fixé *a priori* et $I(\hat{z})$ la semi-norme de lissage. La solution de ce problème de minimisation peut être exprimée par la somme de deux composantes [115] :

$$\hat{z}(\mathbf{X}) = T(\mathbf{X}) + \sum_{i=1}^{n} \lambda_i R(\mathbf{X}, \mathbf{X}_i), \tag{2.35}$$

où $T(\mathbf{X})$ est une fonction polynomiale dite de *tendance* dont les coeffcients sont à définir, $R(\mathbf{X}, \mathbf{X}_i)$ est une fonction de base dont la forme dépend du choix de $I(\hat{z})$ et les coefficient λ_i sont à définir. L'une des formes les plus utilisées est celle des splines de type *plaque mince* [116]

plus connues sous le nom anglais *thin plate splines* [117]. Initialement développées en dimension $N = 2$, elles sont la généralisation des splines cubiques de lissage et donnent à la fonction de base $R(\mathbf{X}, \mathbf{X}_i)$ la forme de l'énergie de flexion d'une plaque mince :

$$R(\mathbf{X}, \mathbf{X}_i) = \int\int \left[\left(\frac{\partial^2 \hat{z}}{\partial x^2} \right)^2 + 2 \left(\frac{\partial^2 \hat{z}}{\partial x \partial y} \right)^2 + \left(\frac{\partial^2 \hat{z}}{\partial y^2} \right)^2 \right] dxdy. \tag{2.36}$$

Cette technique minimise la courbure de la surface et imite le comportement d'une tôle d'acier forcée de passer par les points cibles $\mathbf{X}_i$. Même si ce concept a été développé en dimension $N = 2$, il est tout à fait possible de le généraliser en dimension quelconque [118–120]. Il présente toutefois une limite importante. En effet, la rigidité de la plaque génère des dépassements dans les régions de l'espace présentant de forts gradients. Il est toutefois possible de résoudre ce problème en utilisant les splines de type plaque mince *avec tension* [121–123]. Cela a pour effet de modifier le comportement de la surface, passant d'une plaque rigide à une membrane élastique. De part son développement initial, il n'est pas étonnant de trouver de nombreux cas d'application dans le domaine de l'interpolation spatiale tels que le traitement de données météorologiques de précipitations [124–127] ou de données sismiques [128].

L'inconvénient de cette technique est qu'elle n'exploite pas le caractère local des splines même si des algorithmes ont été développés en se servant de cette propriété pour réduire la compléxité du calcul [129]. Il existe cependant des variantes utilisant les propriétés locales des splines dont la plus connue est celle développée par le physicien et statisticien Jerome Friedman [130, 131] et portant le nom MARS pour *Multivariate Adaptive Regression Splines*. Elle a par la suite été commercialisée par Salford Systems et a bénéficié d'une bonne popularité dans le domaine des statistiques [132–136] mais n'a pas encore trouvé sa place dans le domaine de l'ingénierie. Cette méthode est basée sur un schéma de découpage de l'espace développé par Friedman et crée ensuite un métamodèle local pour chaque région à partir de splines cubiques ou linéaires. La force de cette méthode n'est pas dans le choix des splines mais bel et bien dans l'algorithme de découpage de l'espace. Ce découpage peut s'avérer pertinent en permettant, par une multitude de modèles très simples, de modéliser des comportements fortement non-linéaires. Cependant, le choix des splines cubiques doit être associé à une paramétrisation adéquate pour éviter notamment les boucles, les plis ainsi que les discontinuités si des régions adjacentes ne partagent pas leurs coins [2]. Remplacer les splines cubiques par des linéaires évite ce problème mais le nombre de subdivisions nécessaires peut augmenter énormément.

2.2.9 Métamodèles basés sur les B-splines rationnelles non-uniformes

Les B-splines rationnelles non-uniformes, ou NURBS (*Non-Uniform Rational Basis Splines*), sont utilisées dans la plupart des logiciels de conception assistée par ordinateur (CAO) pour la représentation des courbes et surfaces. Elles sont notamment connues pour être capables de représenter des surfaces hautement non-convexes. Il existe un grand nombre d'exemples de leur utilisation en dimension $N \leq 2$ mais peu de développements ont été faits pour les généraliser au cas des hypersurfaces et, avant les travaux proposées par Turner [1], on ne trouve pas d'exemple significatif dans la littérature de leur utilisation pour la réduction de modèle.

Le cœur des travaux de ce manuscrit étant basé sur l'utilisation des NURBS, nous donnons ici une très brève description de leur formalisme, celui-ci étant détaillé au chapitre 3. Les NURBS

sont des entités géométriques paramétrées définies dans un espace de dimension N vers un espace de dimension M. Les cas d'utilisation les plus courants de la littérature sont répertoriés dans le tableau 2.2. La généralisation au cas de dimension N et M quelconques est développée au chapitre 3. Les NURBS sont la généralisation des courbes et surfaces B-Splines et, par extension, des courbes et surfaces de Bézier. Elles présentent l'intérêt de pouvoir décrire, de manière exacte et non approchée, des entités de formes polynomiales telles que les coniques, ce que les courbes et surfaces B-splines ne permettaient pas jusque là. Le point d'une hypersurface NURBS $\mathbf{H}(\mathbf{u}) : [a,b]^N \to \mathbb{R}^M$ est défini par l'équation suivante :

$$\begin{gathered}\mathbf{H}\left(\mathbf{u}\right) = \frac{\sum_{i_1=0}^{n_1} \cdots \sum_{i_N=0}^{n_N} N_{i_1,p_1}\left(u^{(1)}\right) \times ... \times N_{i_N,p_N}\left(u^{(N)}\right) \omega_{i_1,...,i_N} \mathbf{P}_{i_1,...,i_N}}{\sum_{j_1=0}^{n_1} \cdots \sum_{j_N=0}^{n_N} N_{j_1,p_1}\left(u^{(1)}\right) \times ... \times N_{j_N,p_N}\left(u^{(N)}\right) \omega_{j_1,...,j_N}}, \\ u^{(k)} \in \left[a_k, b_k\right], \quad k = 1, ..., N,\end{gathered} \tag{2.37}$$

où $\mathbf{u} = \left(u^{(1)} \cdots u^{(N)}\right)$ est le vecteur des paramètres $u^{(k)}$, $k = 1, ..., N$, a_k et b_k sont les bornes du paramètre $u^{(k)}$, $\omega_{i_1,...,i_N}$ est le poids affecté au point de contrôle $\mathbf{P}_{i_1,...,i_N} = \left\{P^{(1)}_{i_1,...,i_N}, ..., P^{(M)}_{i_1,...,i_N}\right\}$, $n_k + 1$ est le nombre de points de contrôle dans la direction k, $k = 1, ..., N$, $\mathbf{H}\left(\mathbf{u}\right) = \left\{H^{(1)}\left(\mathbf{u}\right), ..., H^{(M)}\left(\mathbf{u}\right)\right\}$ est le point de l'hypersurface et $N_{i_k,p_k}\left(u^{(k)}\right)$ est la fonction de base de degré p_k associée à la direction k, $k = 1, ..., N$, de l'espace et définie récursivement par

$$\begin{aligned} N_{i_k,0}\left(u^{(k)}\right) &= \begin{cases} 1, & \text{si} \quad U^{(k)}_{i_k} \leq u^{(k)} < U^{(k)}_{i_k+1}, \\ 0, & \text{sinon}, \end{cases} \\ N_{i_k,\tau}\left(u^{(k)}\right) &= \frac{u^{(k)} - U^{(k)}_{i_k}}{U^{(k)}_{i_k+\tau} - U^{(k)}_{i_k}} N_{i_k,\tau-1}\left(u^{(k)}\right) + \frac{U^{(k)}_{i_k+\tau+1} - u^{(k)}}{U^{(k)}_{i_k+\tau+1} - U^{(k)}_{i_k+1}} N_{i_k+1,\tau-1}\left(u^{(k)}\right), \\ i_k = 0, ..., n_k, \quad \tau = 1, ..., p_k, \quad k = 1, .., N, \end{aligned} \tag{2.38}$$

avec $\mathbf{U}^{(k)}$ le *knot vector* associé à la direction k de l'espace et formé de $m_k + 2$ valeurs $U^{(k)}_{l_k}$ avec $m_k = n_k + p_k + 1$ et se présente sous la forme,

$$\mathbf{U}^{(k)} = \{\underbrace{a_k, ..., a_k}_{p_k+1}, U^{(k)}_{p_k+1}, ..., U^{(k)}_{m_k-p_k+1}, \underbrace{b_k, ..., b_k}_{p_k+1}\}, \quad k = 1, ..., N. \tag{2.39}$$

Généralement, on fixe $a_k = 0$ et $b_k = 1$.

N	M	Type d'entité
1	2	Courbe plane
1	3	Courbe 3D
2	3	Surface 3D

Tableau 2.2 – Cas classiques d'utilisation des NURBS

Il existe un grand nombre de méthodes et d'algorithmes capables d'interpoler ou d'approximer un ensemble de points pour les cas des courbes et surfaces. Ils sont d'ailleurs largement utilisés en CAO car il est possible de contraindre la courbe, ou la surface, à passer par des points cibles mais il est également possible d'imposer des contraintes sur la tangence et la courbure locales [2]. La plupart des approches d'approximation peuvent être classées en deux catégories et sont basées sur un calcul itératif. L'inconvénient est qu'un certain nombre de paramètres doivent être fixés *a priori*, notamment les degrés p_k, $k = 1, ..., N$, et les *knot vectors* $U^{(k)}$, $k = 1, ..., N$. Suite à cela, seuls restent les nombres $n_k + 1$ de points de contrôle dans chaque direction k à déterminer, les coordonnées des points de contrôle étant obtenues par la méthode des moindres carrés et se traduisant par la résolution d'un système d'équations linéaires. La première approche vise à approximer la courbe ou la surface par le minimum de points nécessaires, en général les bornes du domaine, et on augmente progressivement le nombre de points de contrôle jusqu'à ce qu'un critère d'arrêt soit atteint, en général une erreur maximale d'approximation. La résolution du système d'équations linéaires permettant d'obtenir les coordonnées des points de contrôle est nécessaire à chaque itération. La seconde approche consiste à partir de l'interpolation des données, i.e. autant de points de contrôle que de points cibles, puis diminuer progressivement le nombre jusqu'à ce qu'un critère d'arrêt soit atteint, là encore souvent une erreur maximale d'approximation.

Ces méthodes sont, pour la plupart, généralisables aux cas de dimensions N et M quelconques mais jusqu'aux travaux de Turner [1], il ne semble pas y avoir eu de développement dans le cadre de la réduction de modèle. Il existe cependant des exemples où le cas d'hypersurface NURBS a été traité comme par exemple [137] où des hypersurfaces NURBS sont utilisées pour le calcul d'animation de scènes. La méthode de réduction de modèle développée par Turner est basée sur le principe itératif d'ajout de points de contrôle [138–140]. Elle se distingue des méthodes classiques par l'ajout de points de contrôle à l'endroit où l'erreur d'approximation est maximale, contrairement à la plupart des méthodes qui favorisent un maillage régulier des points de contrôle. De plus, la méthode n'implique pas à l'utilisateur de fixer des paramètres *a priori* et elle est associée à une méthode d'échantillonage pour l'obtention des points cibles [141]. Nous allons brièvement décrire son fonctionnement.

Dans un premier temps, les variables indépendantes $X^{(k)}$, $k = 1, ..., N$, sont normalisées de telle sorte qu'elles soient comprises entre 0 et 1. Elles correspondent alors aux coordonnées paramétriques de la NURBS et sont appelées coordonnées paramétrées,

$$u^{(k)} = \frac{X^{(k)} - X^{(k)}_{min}}{X^{(k)}_{max} - X^{(k)}_{min}}, \quad k = 1, ..., N. \tag{2.40}$$

Les sorties du métamodèle correspondant aux variables dépendantes n'ont pas besoin d'être normalisées, les hypersurfaces NURBS étant invariantes par les transformations géométriques (cf. le chapitre 3). Les degrés p_k, $k = 1, ..., N$, des fonctions de bases sont fixés à 1 dans la phase initiale car les différents n_k sont fixés à 1 également et ne permettent donc pas d'utiliser des fonctions de base de degré $p_k > 1$. A l'itération 2, les degrés p_k sont fixés à 2 et restent inchangés pour les itérations suivantes. Turner justifie ce choix par le fait qu'utiliser des fonctions de base de degrés plus élevés n'apporte pas de bénéfice notable [139]. Toutefois les degrés p_k des fonctions de bases sont liés à la dérivabilité de la NURBS dans la direction k au niveau des nœuds et, de ce fait, certaines applications peuvent nécessiter l'utilisation de degrés $p_k > 2$.

La fig. 2.6 présente le schéma itératif d'ajout des points de contrôle dans un cas ou $N = 2$. Il est bien sûr généralisable au cas de dimension N quelconque mais il est plus facile pour la compréhension et la visibilité de l'expliquer en dimension $N = 2$. Notons que la dimension M est, elle, quelconque. Dans un premier temps seuls les points de contrôle aux extrémités sont fixés, i.e. $u^{(k)} = \{0, 1\}$, $k = 1, ..., N$. A l'itération suivante un point de contrôle est ajouté à l'endroit où l'erreur RMS (*Root Mean Square*) est maximale et des points de contrôle sont ajoutés pour maintenir un hypercube de contrôle. Strictement parlant, seul un hyperrectangle est nécessaire mais la méthode est implémentée en utilisant des hypercubes [139].

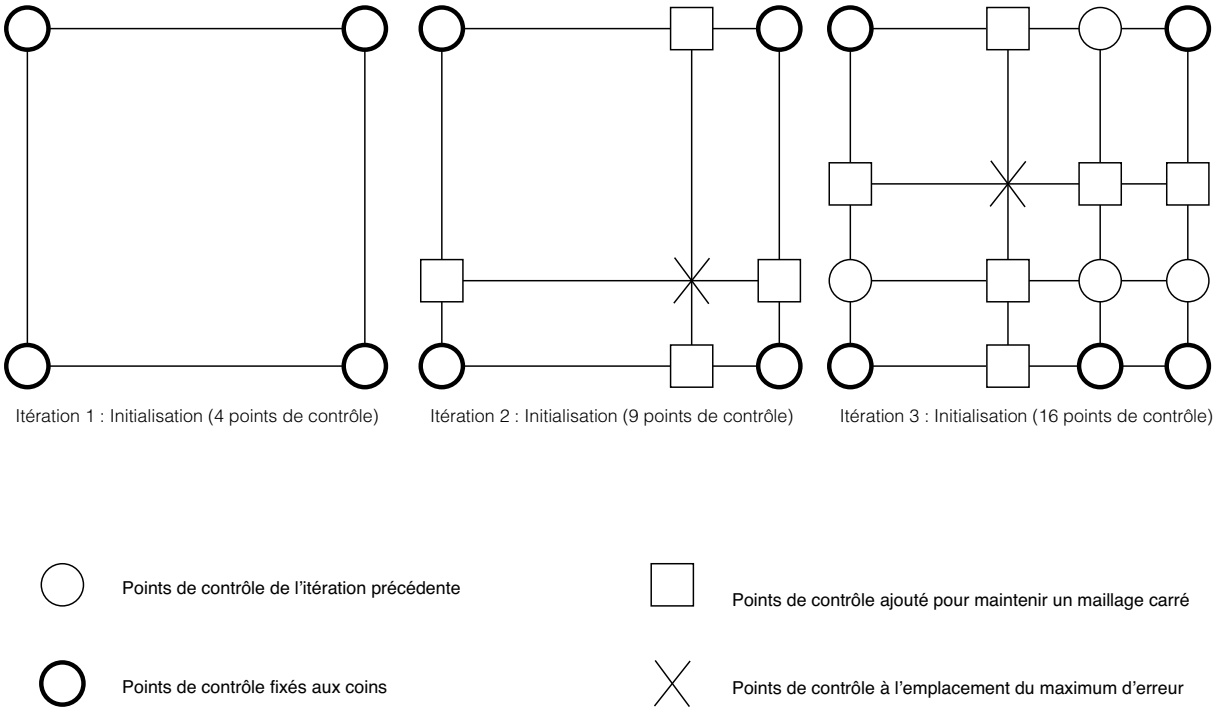


FIGURE 2.6 – Schéma itératif d'ajout de points de contrôle de la méthode HyPerModel [1] pour $N = 2$

Les différents *knot vectors* sont calculés à chaque itération de manière à assurer l'inversibilité du système d'équations permettant d'obtenir les coordonnées des points de contrôle. Enfin, selon le cas, il est possible d'affecter des poids non-unitaires et la méthode utilisée est celle du plus proche voisinage. Ces voisinages sont déterminés par un nombre n_v fixé *a priori* de points cibles utilisés dans le calcul des poids. La proximité d'un point cible avec un point de contrôle est déterminé seulement par les coordonnées paramétriques de ces points. Le poids affecté à un point de contrôle $\mathbf{P}_{i_1,...,i_N}$ est donné par une fonction de correlation spatiale [9] (cf. section 2.2.4) et, en dimension $N = 1$, le poids ω_i est lié au point de contrôle $\mathbf{P}_i$ par la relation suivante :

$$\omega_i = \omega_{min} + (\omega_{max} - \omega_{min}) \left(\mathbf{r}^T \mathbf{R}^{-1} \mathbf{r}\right), \tag{2.41}$$

avec ω_{min} et ω_{max} les poids minima et maxima respectivement, contraints par la relation $0 <$

$\omega_{min} \leq \omega \leq \omega_{max}$, $\mathbf{r}$ le vecteur des corrélations spatiales $r_s = R(u_i, u_s)$, $s = 1, ..., n_v$, entre le point de contrôle $\mathbf{P}_i$ et les n_v points cibles de son voisinage et $\mathbf{R}$ la matrice des corrélations spatiales $R_{s_1,s_2} = R(u_{s_1}, u_{s_2})$, $(s_1, s_2) \in \{1, ..., n_v\}$, entre les points du voisinage s_1 et s_2. La fonction de corrélation spatiale est donnée par la relation suivante :

$$R(u_{s_1}, u_{s_2}) = e^{-\theta |u_{s_1} - u_{s_2}|^p}, \tag{2.42}$$

où θ définit un rayon d'influence et p définit le taux de décroissance de cette influence. Turner [1] a estimé que les paramètres θ et p peuvent être fixés à partir de ω_{min} et du nombre de points de contrôle par les relations suivantes :

$$\theta = \ln(\omega_{min}), \tag{2.43}$$

$$p = \frac{\ln(\ln(C))}{\ln\left(\frac{1}{n_c}\right)} \tag{2.44}$$

où $C > 1.0$ est un coefficient définissant l'influence minimale du poids près des frontières du voisinage. Turner [139] a obtenu de bons résultats sur ses cas d'application avec $\omega_{min} = 0.1$, $\omega_{max} = 1$ et $C = 2$. Les coordonnées dépendantes des points de contrôle sont ensuite obtenues par l'inversion d'un système d'équations linéaires. En réalité la stratégie employée permet d'inverser plusieurs systèmes de dimensions bien inférieures mais l'algorithme employé étant similaire à celui que nous utilisons, la méthodologie sera détaillée dans le chapitre 4, où nous présentons également des stratégies alternatives.

Cette méthode à fait ses preuves sur un grand nombre de fonctions de référence de la littérature [1] mais a également été utilisée dans le cadre de l'optimisation [142, 143] et de la caractérisation inverse de matériaux composites [30]. Malgré le fait qu'il n'est pas nécessaire de fixer des paramètres *a priori* pour utiliser cette méthode, les choix des paramètres discrets de l'hypersurface NURBS qui ont été faits ne bénéficient d'aucune justification physique. En outre, cette approche n'est pas capable de fournir ces paramètres autrement que par des règles empiriques bien connues pour les courbes et surfaces NURBS [2]. Ces règles ne garantissent pourtant, ni un nombre minimal de points de contrôle, ni des valeurs optimales pour les *knot vectors*. Elles permettent simplement de faciliter le calcul numérique des coordonnées des points de contrôle. Dans le cadre des courbes et des surfaces, les temps de calcul et les dimensions des matrices à stocker ne représentent pas une limitation, ce qui justifie d'utiliser de telles règles. Dans le cadre de la réduction de modèle, ce n'est pas le cas, et c'est pourquoi nous proposons une approche novatrice dans le domaine des NURBS, capable de fournir les paramètres optimaux de l'hypersurface sans avoir à formuler d'hypothèses ou à utiliser des règles empiriques. En effet, plutôt que d'utiliser un schéma itératif, ne faisant varier que le nombre de points de contrôle, pour la phase de calibration du métamodèle, nous incluons tous les paramètres qui influent sur la forme de l'hypersurface NURBS. Le but étant d'obtenir un métamodèle optimal, garantissant le nombre le plus faible possible de données et minimisant le temps de calcul dans sa phase d'évaluation.

2.3 Méthodes de réduction d'ordre

Les méthodes de *réduction d'ordre* ont été développées comme une alternative possible aux méthodes de discrétisation existantes. Leur objectif est de réduire le coût de calcul d'une si-

mulation par rapport au modèle initial. Considérons par exemple un maillage discret de n_M nœuds. Dans le cas d'un problème transitoire ou non-linéaire, il est nécessaire de calculer les n_M valeurs pour chaque pas de temps, et dans le cas des éléments finis, cela correspond à inverser au moins une matrice de taille n_M par pas de temps, ce qui peut devenir relativement coûteux lorsque n_M augmente. L'idée derrière les modèles de réduction d'ordre est de trouver un sous-espace de dimension n_N plus petit que n_M afin de diminuer le coût de calcul. Elles sont toutefois transposables au cas de l'approximation [7].

2.3.1 Décomposition orthogonale aux valeurs propres (POD)

La décomposition orthogonale aux valeurs propres, ou POD (*Proper Orthogonal Decomposition*) aussi connue sous le nom de *Principal Component Analysis* (PCA) ou *Karhunen-Loève Decomposition* (KLD), fut initialement proposée indépendamment par plusieurs scientifiques [144–148] et introduite par Lumley [149] dans l'analyse d'écoulements turbulents. Elle est maintenant très utilisée dans l'analyse numérique de la mécanique des fluides [6, 150–157], ou *Computational Fluid Dynamic* en anglais, mais elle a également été adaptée à d'autres domaines comme la résolution de problèmes dynamiques non-linéaires [158], le traitement de signals et la reconnaisance de formes [159], la théorie du contrôle [160–164], la résolution de problèmes inverses [165, 166], la simulation de dynamique moléculaire [167] et l'approximation [7].

Cette méthode est dite *a posteriori* car, dans le cadre de la réduction d'ordre où elle a été initialement proposée, elle postule qu'une approximation d'un champs inconnu $z(\mathbf{X}, t)$ est connue en différents points $\mathbf{X}_i, i = 1, ..., n$, d'un maillage spatial pour des temps discrets $t_m, m = 1, ..., P$. Dans le cadre de l'approximation, $\mathbf{X}$ peut représenter des paramètres de conception et t des pas de calcul pour une analyse non-linéaire par exemple. Dans un premier temps, la fonction $z(\mathbf{X}, t)$ est décomposée en deux parties :

$$z(\mathbf{X}, t) = \langle z(\mathbf{X}) \rangle_t + u(\mathbf{X}, t), \tag{2.45}$$

où $\langle z(\mathbf{X}) \rangle_t$ est la moyenne temporelle de $z(\mathbf{X}, t)$ et $u(\mathbf{X}, t)$ la fonction de variation de $z(\mathbf{X}, t)$ autour de cette moyenne avec $\langle u(\mathbf{X}, t) \rangle = 0$. L'objectif principal de la POD est de trouver la structure la plus caractéristique $\phi(\mathbf{X})$ de l'ensemble des points connus du champs $z(\mathbf{X}, t)$, ce qui revient mathématiquement à trouver la fonction de base ϕ maximisant la moyenne de la projection de u sur cette fonction [35, 168], i.e.

$$\max_{\phi} \left(\frac{\langle (u|\phi)^2 \rangle}{(\phi|\phi)} \right), \tag{2.46}$$

où $\langle \cdot \rangle$ désigne l'opérateur moyenne et $(\cdot|\cdot)$ le produit scalaire. Il est possible d'ajouter une contrainte $\|\phi\|^2 = (\phi|\phi) = 1$ afin de rendre la solution unique [35]. Il a été montré que le problème peut alors être mis sous la forme d'un problème de calcul de valeurs propres [35, 169]. Les fonctions propres $\phi_k(\mathbf{X})$, associées aux valeurs propres λ_k, permettent d'écrire $u(\mathbf{X}, t)$ de la manière suivante :

$$u(\mathbf{X}, t) = \sum_{k=1}^{\infty} a_k(t)\phi_k(\mathbf{X}), \quad \lambda_k \geq \lambda_{k+1} \geq 0. \tag{2.47}$$

Les coefficients $a_k(t)$ sont les coefficients relatifs à la variation temporelle non-corrélés et sont déterminés par :

$$a_k(t) = (u(\mathbf{X}, t)|\phi_k(\mathbf{X})). \tag{2.48}$$

Le premier mode orthogonal ϕ_1, associé à la valeur propre la plus grande λ_1, est la fonction optimale caractérisant les échantillons (*snapshots*). Le second mode propre ϕ_2, associé à la valeur propre λ_2, est la fonction optimale caractérisant les *snapshots*, restreinte cependant à l'espace orthogonal au premier mode, et ainsi de suite. L'énergie E_{tot} contenue dans les données est ainsi définie comme la somme des valeurs propres, i.e. $E_{tot} = \sum_k \lambda_k$, et il est possible de définir un pourcentage d'énergie capturée par le mode k [35] :

$$E_k = \frac{\lambda_k}{\sum_k \lambda_k}. \tag{2.49}$$

En pratique il existe deux méthodes permettant d'obtenir les valeurs et vecteurs propres, la méthode classique utilisant une moyenne temporelle et la technique dite des échantillons (*snapshots*) introduite par Sirovich [170,171] utilisant une moyenne spatiale. Le lecteur intéréssé peut se référer à [35, 168, 172] pour en comprendre les différentes subtilités ainsi que le lien avec la décomposition en valeurs singulières, ou SVD (*Single Value Decomposition*). Nous dirons simplement que la technique des *snapshots* est adaptée au cas où le nombre d'échantillons n est élevé. Dans ce cas, le problème aux valeurs propres peut être mis sous la forme suivante :

$$\mathbf{C}\boldsymbol{\phi} = \lambda\boldsymbol{\phi}, \tag{2.50}$$

où $\mathbf{C}$ est la matrice de corrélation spatiale donnée par :

$$\mathbf{C} = \mathbf{Z}.\mathbf{Z}^T, \tag{2.51}$$

avec $\mathbf{Z}$ la matrice des points connus aussi appelés *matrice des snapshots*,

$$\begin{aligned} \mathbf{Z} &= \begin{pmatrix} u(\mathbf{X}_1, t_1) & \cdots & u(\mathbf{X}_1, t_P) \\ \vdots & \ddots & \vdots \\ u(\mathbf{X}_n, t_1) & \cdots & u(\mathbf{X}_n, t_P) \end{pmatrix} \\ &= \begin{pmatrix} z(\mathbf{X}_1, t_1) - \langle z(\mathbf{X}_1)\rangle & \cdots & z(\mathbf{X}_1, t_P) - \langle z(\mathbf{X}_1)\rangle \\ \vdots & \ddots & \vdots \\ z(\mathbf{X}_n, t_1) - \langle z(\mathbf{X}_n)\rangle & \cdots & z(\mathbf{X}_n, t_P) - \langle z(\mathbf{X}_n))\rangle \end{pmatrix}. \end{aligned} \tag{2.52}$$

La matrice $\mathbf{C}$ est hermitienne semi-définie positive, elle possède donc un ensemble complet de vecteurs propres orthogonaux correspondant aux valeurs propres réelles non-négatives de $\mathbf{C}$ [165]. Généralement, les valeurs propres sont rangées par ordre décroissant de telle sorte que :

$$\lambda_1 \geq ... \geq \lambda_n \geq 0. \tag{2.53}$$

Les n solutions λ_k de ce problème aux valeurs propres associées à leurs vecteurs propres ϕ_k forment alors une base de projection de $z(\mathbf{X}, t)$:

$$\hat{z}(\mathbf{X}, t) = \langle z(\mathbf{X})\rangle_t + \sum_{k=1}^{n} a_k(t)\phi_k(\mathbf{X}), \tag{2.54}$$

où $a_k(t)$ est le coefficient relatif à la variation temporelle de $z(\mathbf{X}, t)$ associé à la valeur propre λ_k et $\phi_k(\mathbf{X})$ est la fonction associée à cette même valeur propre. En général, après la résolution du problème aux valeurs propres, seules sont conservées les K premiers ϕ_k associées aux valeurs propres $\lambda_k \in [\lambda_1, \alpha\lambda_1]$, $0 < \alpha \leq 1$. Le paramètre α est choisi de manière à ne pas considérer les modes dont l'influence serait négligeable par rapport au premier. Une autre option consiste à fixer le pourcentage minimal d'énergie capturée $P_{E,min}$ par les K premiers modes, le pourcentage d'énergie capturée P_E étant défini de la manière suivante :

$$P_E = \frac{E_K}{E_n} = \frac{\sum_{k=1}^{K} \lambda_k}{\sum_{k=1}^{n} \lambda_k} \tag{2.55}$$

Normalement, K est beaucoup plus faible que n ce qui permet de diminuer grandement la complexité du calcul. En effet, les coefficients $a_k(t)$ sont obtenus par la résolution d'un système d'équations linéaires de taille $K \ll n$. Cette méthode est dite *a posteriori* car elle nécessite un ensemble d'échantillons déjà connus, ce qui est le cas des méthodes d'interpolation et d'approximation que nous avons décrites précédemment. Cependant, dans le cas de la réduction d'ordre où elle fut introduite initialement, il est possible de se questionner sur l'utilité de devoir calculer un modèle complet avant d'utiliser le modèle réduit, ce qui n'est pas le cas de la méthode de *décomposition aux valeurs propres généralisée* (PGD) dont nous parlerons dans le paragraphe suivant. Il y a en fait deux approches largement répandues et ayant montré leurs utilités. La première consiste à résoudre le modèle original sur un court intervalle de temps afin d'en extraire la structure caractéristique qui définit le modèle réduit. Le calcul sur un intervalle de temps plus long est ensuite effectué avec le modèle réduit et le gain de temps associé. L'autre approche consiste à résoudre le modèle original sur tout l'intervalle de temps pour ensuite utiliser le modèle réduit sur des problèmes similaires avec, par exemple, des variations dans les paramètres des matériaux ou dans les conditions limites.

Bien que développée dans le cadre de la réduction d'ordre, cette méthode est transposable au contexte de l'approximation [7]. Elle présente, dans ce cas, l'avantage de ne nécessiter qu'un seul paramètre à fixer *a priori* par l'utilisateur (α ou $P_{E,min}$) et s'adapte automatiquement aux données qu'elle traite. Elle en extrait en fait la structure caractéristique. Notons toutefois qu'un grand nombre de *snapshots* peut rendre le calcul des valeurs propres très coûteux et qu'il est quand même nécessaire, en phase d'utilisation du métamodèle, d'inverser un système de dimension K qui peut devenir important lorsque la dimension N des paramètres augmente.

2.3.2 Décomposition propre généralisée (PGD)

La *décomposition propre généralisée*, ou *Proper Generalized Decomposition* (PGD), fut initialement introduite par Ladevèze [173–179] sous le nom de *chargement radial* dans la méthode LATIN. Elle avait pour objectif de traiter les non-linéarités comme la plasticité dans des problèmes à dépendance temporelle [174, 175, 180–184]. Elle a ensuite été appliquée à d'autres domaines comme les grands déplacements [176, 185], le calcul d'homogénéisation [186], les problèmes de caractérisation inverse [187], la prédiction d'endommagement dans les matériaux composites [188–192], le calcul d'assemblage [193], la simulation dynamique de chocs [194–196], les études multiparamétriques [197–199] et l'optimisation [200, 201]. Étant la seule méthode dite *a*

priori (dans sa forme intrusive), car elle ne nécessite pas la résolution préalable du modèle complet, elle a également permis de résoudre des problèmes jusqu'alors impossibles à traiter à cause du *fléau de la dimension* (*curse of dimensionality*) dont notamment la résolution de problèmes de chimie quantique [5].

Cette méthode, contrairement à la POD, ne déduit pas une base de projection à partir de données connues mais calcule directement une base de manière itérative. Elle utilise une représentation séparée dont la forme générale est la suivante :

$$\hat{z}(X^{(1)}, ..., X^{(N)}) = \sum_{k=1}^{K} F_k^{(1)}(\mathbf{X}^{(1)}) \cdots F_k^{(D)}(\mathbf{X}^{(D)}), \tag{2.56}$$

où $F_k^{(i)}(\mathbf{X}^{(i)})$ est la fonction, inconnue *a priori*, liée au mode k, $k = 1, ..., K$, et à la variable $\mathbf{X}^{(i)}$ qui appartient à un sous-domaine $\Omega_i \subset R^{d_i}$, avec en général $d_i \leq 3$. Les différentes dimensions d_i des variables $\mathbf{X}^{(i)}$ respectent la propriété suivante :

$$\sum_{i=1}^{D} d_i = N. \tag{2.57}$$

On peut donc remarquer que, dans le cas où $d_i = 1$, $i = 1, ..., D$, alors D est égal à N.

L'approximation PGD forme ainsi une somme de K produits de D fonctions $F_k^{(i)}(\mathbf{X}^{(i)})$ qui sont déterminées par enrichissements successifs. A une étape n_s+1 d'enrichissement, les fonctions $F_k^{(i)}(\mathbf{X}^{(i)})$ sont toutes connues pour $k \leq n_s$, car elles ont été calculées aux étapes précédentes, et il faut calculer les D nouvelles fonctions $F_{n_s+1}^{(i)}(\mathbf{X}^{(i)})$. Elles sont obtenues par la *formulation faible* du problème considéré qui résulte en un problème non-linéaire nécessitant une résolution itérative pour chaque étape d'enrichissement. Ce problème non-linéaire nécessite la résolution de problèmes linéaires de dimension d_i et donc, en général, de faible dimension par rapport à N. Dans le cas où $d_i = 1$, ce qui est vrai dans beaucoup d'application, et en supposant que l'on prenne une discrétisation n pour chaque coordonnées $X^{(i)}$, le nombre total d'inconnues de la PGD est de $K \times n \times N$ au lieu de n^N degrés de liberté pour un calcul classique basé sur une discrétisation de l'espace telle que la *méthode des éléments finis*. En pratique, les exemples traités dans la littérature ont montré que le nombre total de modes K, permettant d'obtenir une solution précise, n'était pas fonction de la dimension N du problème traité mais plutôt des caractéristiques de séparabilité de la solution exacte. De ce fait, la complexité de calcul ne grandit généralement pas exponentiellement avec la dimension N, contrairement aux autres méthodes de discrétisation. Pour un aperçu plus complet de la PGD, et notamment des techniques de calcul des différents enrichissements, le lecteur intéressé par le sujet est renvoyé vers les références [4, 202–204].

De toute les méthodes présentées dans ce chapitre, elle est la seule capable de fournir un modèle réduit ne nécessitant pas de base de données au préalable. Il est toutefois nécessaire, dans ce cas, de déterminer la formulation faible du phénomène physique à l'étude. Elle présente donc un avantage certain sur les autres techniques et c'est ce qui explique son développement. Il existe également une forme non-intrusive de la PGD qui s'utilise dans le cas de l'approximation. Dans ce cas, la formulation de la fonction $\hat{z}(\mathbf{X})$ reste celle de l'éq. (2.56) et le nombre de modes est déterminé par un seuil fixé sur la précision du métamodèle. Le calcul de chaque mode est, en

général, effectué en minimisant l'erreur des moindres carrés. Il est également possible d'utiliser, comme critère de convergence, le gain obtenu par l'ajout d'un mode. Comme pour la POD, lorsque le gain d'un mode devient trop faible par rapport au premier, il n'est plus utile d'en ajouter de nouveaux. Il est également possible d'utiliser ces deux critères d'arrêt simultanément.

Il faut toutefois noter quelques limites et, notamment, une méthode d'interpolation doit être choisie en phase d'utilisation du métamodèle pour les valeurs intermédiaires non comprises dans le pas de discrétisation. De ce fait, l'erreur d'approximation en ces points intermédiaires est souvent moins bonne que pour les points ayant permis de paramétrer le métamodèle.

2.4 Conclusions

Beaucoup de méthodes ont été présentées dans ce chapitre, toutes présentant un certain nombre d'avantages et d'inconvénients. Un nombre important d'entre elles nécessitent de fixer un certain nombre de paramètres *a priori*. Par exemple, même si les réseaux de neurones et les machines à vecteur support sont les seules méthodes capables de traiter des problèmes de très grande dimension, leur implémentation n'est pas à la portée d'un novice et nécessite une expertise de la méthode. Les méthodes issues de l'interpolation spatiale, telles que le *krigeage* ou les *méthodes barycentriques*, sont quant à elles généralement moins efficaces lorsque la dimension du problème augmente. De plus, certaines méthodes, comme la *régression*, nécessitent de calculer les coefficients pour chaque sortie du métamodèle. Dans le cadre des NURBS, ajouter une sortie se traduit simplement par l'ajout d'une coordonnée aux points de contrôle, et ne nécessite pas de recalculer les différents poids ou fonctions de base pour chaque sortie.

Très peu d'auteurs ont comparé plus de trois méthodes de réduction en même temps et le travail de Turner [1] en ce sens a démontré l'utilité et l'efficacité d'une méthode basée sur les NURBS. En particulier, il a montré que la méthode n'est pas la meilleure dans tous les domaines mais présente de bons résultats partout. L'un des points clés cependant d'une bonne méthode de réduction de modèle est de ne pas nécessiter de paramètres à fixer *a priori* par l'utilisateur. Or, ce qui fait la grande capacité des NURBS à s'adapter à des topologies hautement non-convexes est le grand nombre de paramètres permettant de modifier leurs formes. Turner [1] a réglé ce problème par l'utilisation d'un grand nombre de règles empiriques, notamment sur le choix des degrés des fonctions de forme et des valeurs des *knot vectors*. Ainsi, l'utilisateur n'a pas de paramètres à fixer pour utiliser la méthode mais les valeurs des paramètres qui ont été choisies ne sont pas optimales. De ce postulat, la méthode que nous proposons vise à fournir les valeurs optimales de ces paramètres plutôt que de les fixer à l'avance. Le problème étant non-convexe, l'utilisation d'un algorithme capable de traiter des problèmes de dimension variable est requise et fera l'objet de la seconde partie du chapitre 3.

Chapitre 3

Outils et méthodes

3.1 Introduction

La volonté de proposer une méthode de réduction de modèle capable de s'adapter à un grand nombre de problèmes et ne nécessitant pas d'expertise de la part de l'utilisateur, ni de connaissance préalable sur la nature des variables à interpréter, nous a très vite conduit vers le formalisme des NURBS. À notre connaissance, ce formalisme n'avait été utilisé que par Turner [1,139], et son équipe de recherche, dans le cadre de la réduction de modèle en dimension $N > 2$. Cependant, ce qui fait la versatilité des NURBS, à savoir, le nombre de paramètres permettant de définir leurs formes est aussi un inconvénient pour un utilisateur non familier de ces entités. Le problème a été éludé par Turner [1] en proposant des valeurs prédéfinies pour tous les paramètres libres de l'hypersurface NURBS, choisies à partir de règles empiriques bien connues pour les courbes et surfaces [2]. Ces règles, bien que liées à la robustesse de calcul la plupart du temps, ne permettent pas à la méthode de Turner d'exploiter tous les paramètres régissant l'hypersurface NURBS. Afin de familiariser le lecteur avec les entités NURBS, et l'ensemble de leurs paramètres, la première partie de ce chapitre développe le formalisme mathématique des courbes et surfaces NURBS avant de généraliser le concept au cas des hypersurfaces.

La plupart des paramètres des NURBS sont en général fixés par des règles métiers car la recherche des paramètres optimaux d'une entité NURBS est un problème non-convexe de dimension variable. Pour le résoudre, notre méthode s'appuie sur l'utilisation d'une métaheuristique, notamment sur un algorithme génétique développé précisément pour traiter des problèmes dont la dimension de l'espace des paramètres varie, et d'un algorithme déterministe, i.e. basé sur le gradient. La seconde partie de ce chapitre expose donc les principes généraux de l'optimisation. Une présentation succincte est faite des différents types d'algorithmes existant puis nous présentons en détail l'algorithme génétique utilisé dans la méthode que nous avons développée ainsi que les modifications que nous y avons apportées. Notre méthode s'appuie également sur l'utilisation d'un algorithme déterministe que nous présentons après avoir introduit les principes généraux de ces méthodes.

3.2 Généralisation des entités géométriques NURBS classiques

Avant d'introduire le formalisme mathématique des courbes et surfaces NURBS, acronyme pour *Non-Uniform Rational B-Splines*, ainsi que la généralisation au cas des hypersurfaces, nous souhaitons préciser que les notations utilisées sont celles du livre « The NURBS Book » [2]. De plus, cette section a une vocation didactique afin d'aider le lecteur dans la compréhension des effets des différents paramètres régissant les entités NURBS. Pour un aperçu plus approfondi du domaine, le lecteur est renvoyé vers les références [2, 205].

Les courbes et surfaces NURBS sont, de nos jours, massivement utilisées dans les applications CAO (*Conception Assisté par Ordinateur*) où elles sont devenues la norme car capables de représenter à la fois des formes géométriques analytiques, telles que les sections coniques ou les surfaces quadriques, et des formes libres comme des carrosseries de voiture.

3.2.1 Courbes

Une courbe NURBS est une entité géométrique paramétrique (i.e. exprimée en forme explicite) dont la forme mathématique, pour une courbe de degré p, est la suivante [2] :

$$\mathbf{C}(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u)\,\omega_i \mathbf{P}_i}{\sum_{j=0}^{n} N_{j,p}(u)\,\omega_j}, \quad a \leq u \leq b, \tag{3.1}$$

que l'on retrouve également sous la forme [2] :

$$\mathbf{C}(u) = \sum_{i=0}^{n} R_{i,p}(u)\,\mathbf{P}_i, \quad a \leq u \leq b, \tag{3.2}$$

avec $R_{i,p}(u)$ la fonction de base rationnelle par morceaux de degré p calculée au paramètre u et liée au point de contrôle $\mathbf{P}_i$:

$$R_{i,p}(u) = \frac{N_{i,p}(u)\,\omega_i}{\sum_{j=0}^{n} N_{j,p}(u)\,\omega_j}, \tag{3.3}$$

avec $\mathbf{C}(u) = \{C^{(1)}(u), C^{(2)}(u), C^{(3)}(u)\}$ les coordonnées cartésiennes du point de la courbe NURBS au paramètre u, ω_i le poids affecté au point de contrôle $\mathbf{P}_i = \{P_i^{(1)}, P_i^{(2)}, P_i^{(3)}\}$. L'ensemble des $n+1$ points représente le *polygone de contrôle* et les $N_{i,p}(u)$ sont les fonctions de base, définies récursivement de la manière suivante :

$$\begin{aligned} N_{i,0}(u) &= \begin{cases} 1, & \text{si} \quad U_i \leq u < U_{i+1}, \\ 0, & \text{sinon}, \end{cases} \\ N_{i,\tau}(u) &= \frac{u - U_i}{U_{i+\tau} - U_i} N_{i,\tau-1}(u) + \frac{U_{i+\tau+1} - u}{U_{i+\tau+1} - U_{i+1}} N_{i+1,\tau-1}(u), \\ & i = 0, ..., n, \quad \tau = 1, ..., p. \end{aligned} \tag{3.4}$$

où U_i est la i^{e} composante du *knot-vector* non-périodique et non-uniforme suivant :

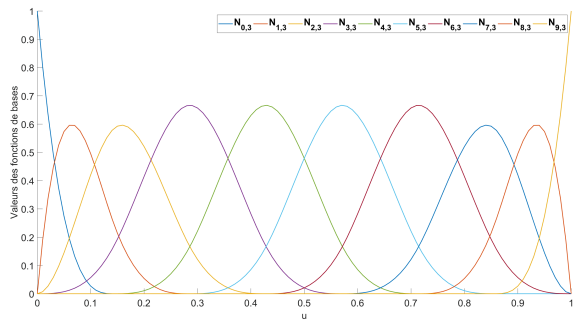
$$\mathbf{U} = \{\underbrace{a, ..., a}_{p+1}, U_{p+1}, ..., U_{m-p+1}, \underbrace{b, ..., b}_{p+1}\}. \tag{3.5}$$

Nous pouvons noter que sa taille est $m+1$, avec :

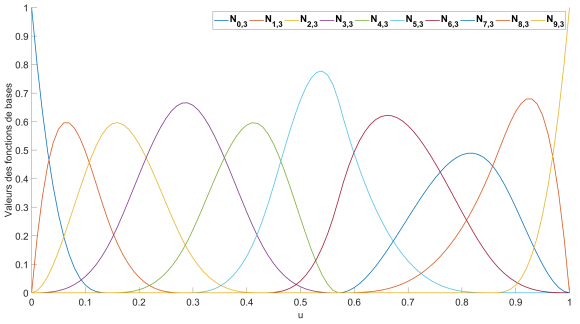
$$m = n + p + 1. \tag{3.6}$$

Il est composé d'une séquence non décroissante de nombres réels, appelés "noeuds", ou *knots* en anglais, qui peut être interprétée comme étant un ensemble discret de valeurs du paramètre u et divisant la courbe en arcs. Chacun de ces *knots* peut avoir une multiplicité λ. Pour la suite du document nous adopterons une convention classique fixant $a = 0$ et $b = 1$, le *knot-vector* ayant donc la forme suivante :

$$\mathbf{U} = \{\underbrace{0, ..., 0}_{p+1}, U_{p+1}, ..., U_{m-p+1}, \underbrace{1, ..., 1}_{p+1}\}. \tag{3.7}$$



(a) Fonctions de base associées au *Knot-vector* uniforme $\mathbf{U_A}$



(b) Fonctions de base associées au *Knot-vector* avec une multiplicité $\lambda = 2$ $\mathbf{U_B}$

FIGURE 3.1 – Fonctions de forme pour des courbes B-Splines/NURBS de degré $p = 3$ et d'indice maximal de point de contrôle $n = 9$

Parmi les propriétés qui régissent les fonctions de forme, il est important de noter *la propriété de partition de l'unité* :

$$\sum_{i=0}^{n} N_{i,p}(u) = 1, \quad \forall u \in [0, 1]. \tag{3.8}$$

Cette propriété permet d'établir deux cas particuliers intéressants. En effet, en imposant que la valeur de l'ensemble des poids $\{\omega_i\}$ soit la même, l'éq. (3.1) se simplifie et nous permet de retrouver l'équation régissant une courbe B-Spline de degré p :

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u) \mathbf{P}_i, \quad 0 \leq u \leq 1 \tag{3.9}$$

De plus, associée à un *knot-vector* de la forme $\mathbf{U} = \{\underbrace{0, ..., 0}_{p+1}, \underbrace{1, ..., 1}_{p+1}\}$, on obtient la formulation d'une courbe de Béziers. Les NURBS sont donc une généralisation des courbes de Béziers et des

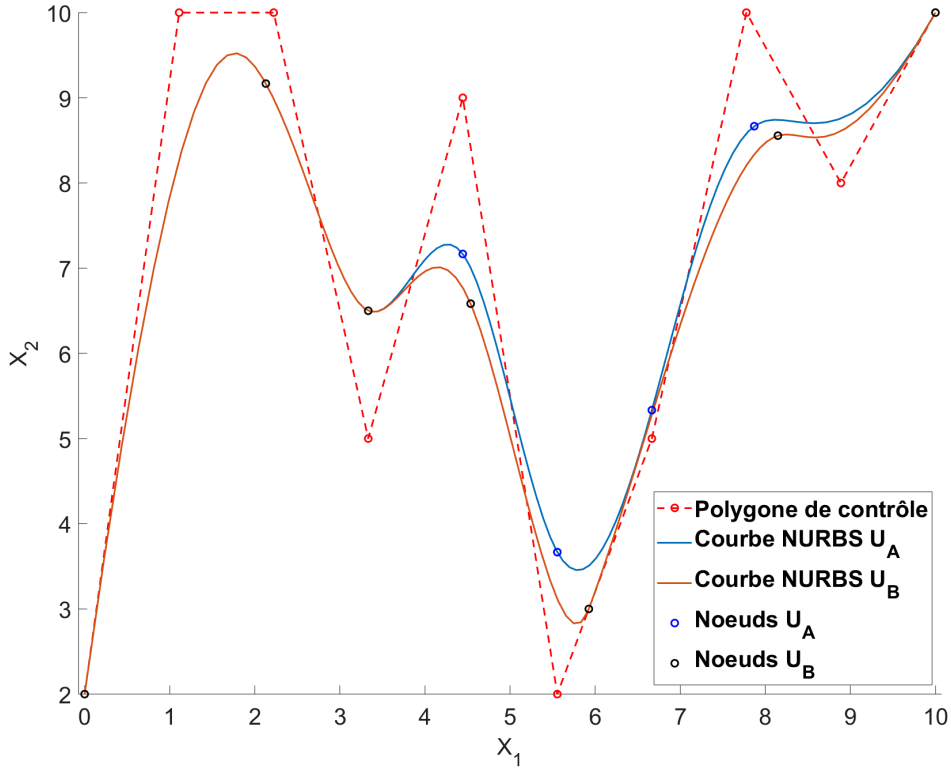


FIGURE 3.2 – Courbes B-Spline de degré $p = 3$ et d'indice maximal de point de contrôle $n = 9$ pour les deux *knot-vectors* $\mathbf{U_A}$ et $\mathbf{U_B}$

B-Splines, dont l'attrait majeur est la capacité à représenter de façon exacte les sections coniques (paraboles, hyperboles, ellipses) contrairement aux deux formulations précédentes qui ne peuvent que les approximer.

Afin de clarifier l'effet des différents paramètres, nous allons présenter une série d'exemples significatifs. Le premier exemple présente une B-Spline ($\{\omega_i\} = 1$) caractérisée par des fonctions de base de degré $p = 3$ et 10 points de contrôle ($n = 9$). Chaque courbe est définie par un *knot-vector* différent, le premier étant uniforme :

$$\mathbf{U_A} = \{0, 0, 0, 0, 0.14, 0.29, 0.43, 0.57, 0.71, 0.86, 1, 1, 1, 1\}, \tag{3.10}$$

et le second possédant un nœud avec une multiplicité $\lambda = 2$ ($U_7 = U_8$),

$$\mathbf{U_B} = \{0, 0, 0, 0, 0.14, 0.29, 0.43, 0.57, 0.57, 0.86, 1, 1, 1, 1\}. \tag{3.11}$$

Les fonctions de forme correspondantes sont présentées à la fig. 3.1. Les différentes courbes obtenues sont présentées à la fig. 3.2. Ces deux figures permettent d'illustrer quelques unes des plus importantes caractéristiques des courbes B-Splines et NURBS :

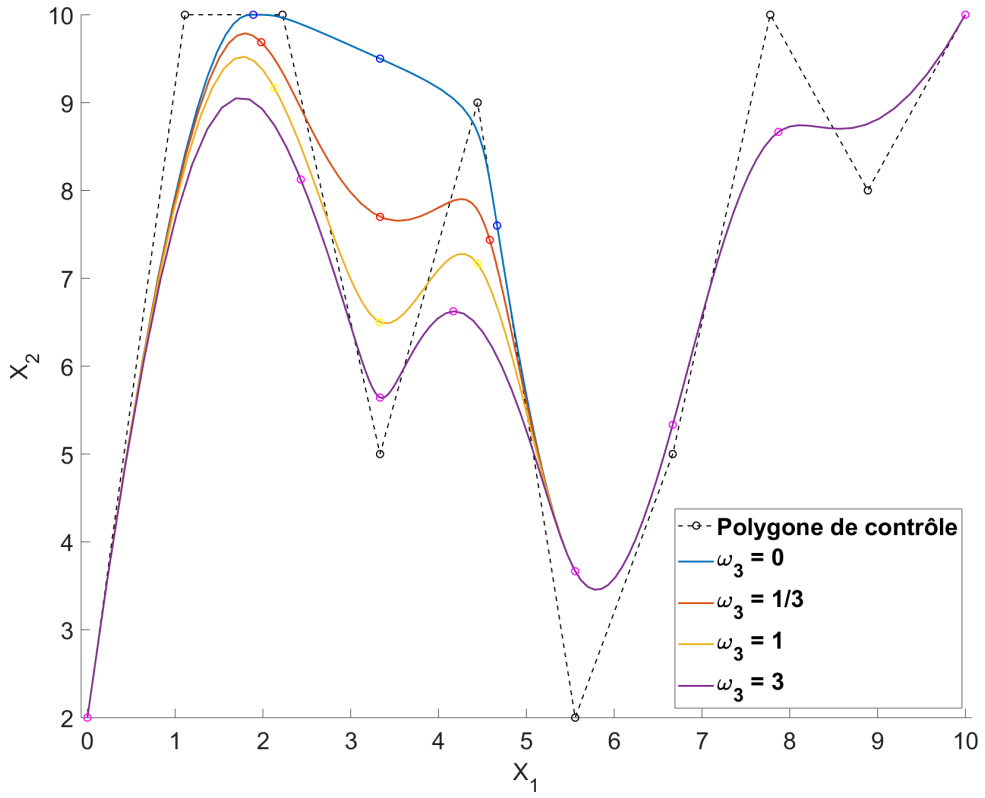


FIGURE 3.3 – Effet du poids sur une courbe NURBS à degré, *knot-vector* et points de contrôle fixés

- *Propriété de support local.* La fig. 3.1 permet d'observer que toutes les fonctions de base sont définies sur un support local (i.e. les fonctions de base sont nulles en dehors d'un certain intervalle de valeurs du paramètre u). La dimension de ce support dépend des valeurs du *knot-vector* et est donnée par la relation suivante [2] :

$$N_{i,p}(u) \neq 0 \quad \texttt{si} \quad u \in [U_i, U_{i+p+1}[\,. \tag{3.12}$$

  Pour cette raison, déplacer un point de contrôle, ou modifier le poids qui lui est associé, n'a qu'un *impact local* sur la forme de la courbe, contrairement à une courbe de Béziers où la modification d'un point de contrôle entraine la modification de l'ensemble de la courbe.
- *Propriété d'enveloppe convexe.* Les fonctions de base étant, par définition, positives, et en utilisant la propriété de *partition de l'unité*, il est possible de démontrer qu'une courbe NURBS (ou B-Spline) est toujours contenue dans l'enveloppe convexe du polygone de contrôle [2].
- *Propriété de continuité et de dérivabilité.* La fonction de base $N_{i,p}(u)$ est $p-\lambda$ fois dérivable à un nœud donné [2], λ étant la multiplicité du nœud. On peut donc remarquer qu'augmenter le degré augmente la continuité alors que la multiplicité d'un *knot* la diminue. Il est

donc évident que le *knot-vector* a un impact important sur les fonctions de base, et donc sur la forme finale de la courbe NURBS obtenue. De plus, il est facilement démontrable (par récurrence) [2] que la dérivée d'une fonction de base est de la forme :

$$N'_{i,p} = \frac{p}{U_{i+p} - U_i} N_{i,p-1}(u) - \frac{p}{U_{i+p+1} - U_{i+1}} N_{i+1,p-1}(u) \tag{3.13}$$

- *Propriété d'invariance par transformation affine et projection.* Les transformations affines, telles que la translation, la rotation ou encore l'homothétie, s'appliquent directement aux points de contrôle. Les courbes NURBS sont également invariantes par projection et notamment par projection perspective.

Le second exemple présente, sur la fig. 3.3, l'effet de la variation du poids sur une courbe NURBS. De part la *propriété de support local*, il est possible de remarquer que la variation du poids ω_3 affecté au point de contrôle $\mathbf{P_3}$ à un impact très localisé. En effet, seul les points de la courbe dont le paramètre u appartient au *support local* des fonctions de base liées à ce point de contrôle sont modifiés. On remarquera notamment que les courbes sont identiques à partir du 5^{e} nœud. Il est également important de noter qu'augmenter le poids affecté à un point de contrôle a pour effet "d'attirer" la courbe vers ce point, tandis que diminuer ce poids a tendance à "l'éloigner".

Nous insistons sur le fait que les propriétés des courbes NURBS ne se limitent pas à celles listées ci-dessus mais elles représentent les plus importantes et pertinentes pour les travaux de cette thèse. Pour un approfondissement du sujet, le lecteur est renvoyé vers les références [2,205].

3.2.2 Surfaces

La formulation mathématique des surfaces NURBS peut se déduire de celles des courbes. Ainsi, les équations précédemment établies peuvent être transposées au cas des surfaces et la forme mathématique est la suivante :

$$\mathbf{S}\left(u^{(1)}, u^{(2)}\right) = \frac{\sum_{i_1=0}^{n_1} \sum_{i_2=0}^{n_2} N_{i_1,p_1}\left(u^{(1)}\right) N_{i_2,p_2}\left(u^{(2)}\right) \omega_{i_1,i_2} \mathbf{P}_{i_1,i_2}}{\sum_{j_1=0}^{n_1} \sum_{j_2=0}^{n_2} N_{j_1,p_1}\left(u^{(1)}\right) N_{j_2,p_2}\left(u^{(2)}\right) \omega_{j_1,j_2}}, \quad \left(u^{(1)}, u^{(2)}\right) \in [0,1]\,, \tag{3.14}$$

qui peut également être mis sous la forme condensée :

$$\mathbf{S}\left(u^{(1)}, u^{(2)}\right) = \sum_{i_1=0}^{n_1} \sum_{i_2=0}^{n_2} R_{i_1,i_2}\left(u^{(1)}, u^{(2)}\right) \mathbf{P}_{i_1,i_2}, \quad \left(u^{(1)}, u^{(2)}\right) \in [0,1]\,, \tag{3.15}$$

avec $R_{i_1,i_2}\left(u^{(1)}, u^{(2)}\right)$ la fonction rationnelle par morceaux liée aux fonctions de base $N_{i_1,p_1}\left(u^{(1)}\right)$ et $N_{i_2,p_2}\left(u^{(2)}\right)$ par la relation suivante :

$$R_{i_1,i_2}\left(u^{(1)}, u^{(2)}\right) = \frac{N_{i_1,p_1}\left(u^{(1)}\right) N_{i_2,p_2}\left(u^{(2)}\right) \omega_{i_1,i_2}}{\sum_{j_1=0}^{n_1} \sum_{j_2=0}^{n_2} N_{j_1,p_1}\left(u^{(1)}\right) N_{j_2,p_2}\left(u^{(2)}\right) \omega_{j_1,j_2}}. \tag{3.16}$$

Dans l'éq. (3.14), $\mathbf{S}\left(u^{(1)}, u^{(2)}\right) \in \mathbb{R}^3$ représente les coordonnées cartésiennes du point de la surface NURBS aux paramètres $u^{(1)}$ et $u^{(2)}$, tandis que p_1 et p_2 représentent les degrés de cette

surface dans les directions $u^{(1)}$ et $u^{(2)}$ respectivement, ω_{i_1,i_2} est le poids relié au point de contrôle $\mathbf{P}_{i_1,i_2} = \left\{P^{(1)}_{i_1,i_2}, P^{(2)}_{i_1,i_2}, P^{(3)}_{i_1,i_2}\right\}$ où $P^{(j)}_{i_1,i_2}$, $j = 1,2,3$, représente les coordonnées cartésiennes du point de contrôle. De la même façon que les points de contrôle forment un *polygone de contrôle* pour les courbes, le réseau des $(n_1+1) \times (n_2+1)$ points forme ce qu'on appelle un *réseau de contrôle*. Chacune des directions, associées aux paramètres $u^{(1)}$ et $u^{(2)}$, nécessite sa propre fonction de base, i.e. $N_{i_1,p_1}\left(u^{(1)}\right)$ et $N_{i_2,p_2}\left(u^{(2)}\right)$ définies par l'éq. (3.4). Par conséquent, une surface NURBS, ou B-Spline, doit avoir deux *knot-vectors* distincts pour chacune des directions paramétriques, i.e.

$$\begin{cases} \mathbf{U}^{(1)} = \{\underbrace{0,...,0}_{p_1+1}, U^{(1)}_{p_1+1},...,U^{(1)}_{m_1-p_1+1}, \underbrace{1,...,1}_{p_1+1}\}, \\ \mathbf{U}^{(2)} = \{\underbrace{0,...,0}_{p_2+1}, U^{(2)}_{p_2+1},...,U^{(2)}_{m_2-p_2+1}, \underbrace{1,...,1}_{p_2+1}\}, \end{cases} \tag{3.17}$$

de taille m_1+1 et m_2+1 respectivement. Ces deux entiers, m_1 et m_2, sont liés aux degrés ainsi qu'aux nombres de points de contrôle de chaque direction par l'éq. (3.6). Les degrés et nombres de points de contrôle pouvant être différents pour chaque direction, les deux *knot-vectors* sont complètement *indépendants* l'un de l'autre.

Les propriétés des courbes NURBS s'appliquent également aux surfaces NURBS, ainsi la *propriété de partition de l'unité* devient,

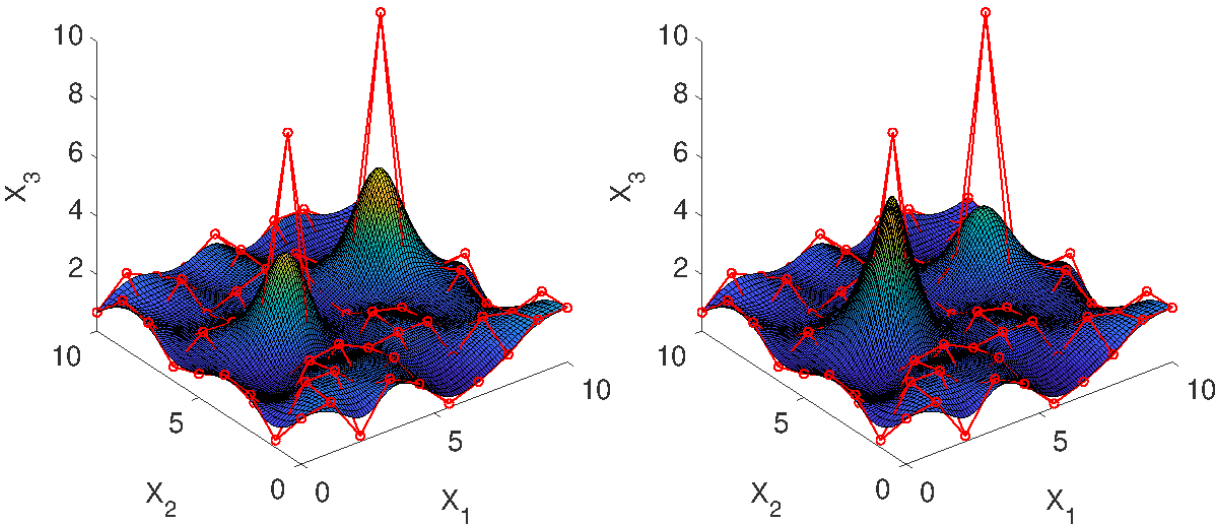
$$\sum_{i_1=0}^{n_1}\sum_{i_2=0}^{n_2} N_{i_1,p_1}\left(u^{(1)}\right) N_{i_2,p_2}\left(u^{(2)}\right) = 1, \quad \forall \left(u^{(1)}, u^{(2)}\right) \in [0,1]^2 . \tag{3.18}$$

On peut donc remarquer que la forme générale des surfaces NURBS inclut la formulation des surfaces B-Splines, de la même manière que pour les courbes. En effet, lorsque tous les poids ω_{i_1,i_2} sont fixés à la même valeur et en introduisant l'éq. (3.18) dans l'éq. (3.14) on obtient :

$$\mathbf{S}\left(u^{(1)}, u^{(2)}\right) = \sum_{i_1=0}^{n_1}\sum_{i_2=0}^{n_2} N_{i_1,p_1}\left(u^{(1)}\right) N_{i_2,p_2}\left(u^{(2)}\right) \mathbf{P}_{i_1,i_2}, \quad \left(u^{(1)}, u^{(2)}\right) \in [0,1]^2 . \tag{3.19}$$

La fig. 3.4 présente un exemple très simple de surfaces NURBS et B-Spline (le lecteur intéressé peut se référer au livre [2] pour approfondir le sujet sur les différences entre B-Spline et NURBS). Les deux surfaces ont été obtenues avec les mêmes degrés, les mêmes *knot-vectors* et le même réseau de contrôle (i.e. les mêmes points de contrôle). Toutefois, les poids affectés aux deux différents "pics" sont différents dans le cas de la NURBS de la fig. 3.4b. Il est possible de remarquer visuellement que le poids associé au plus grand pic a été diminué (inférieur à 1) alors que celui du plus petit pic a été augmenté (supérieur à 1). Le comportement d'une surface NURBS vis à vis des poids est donc le même que pour les courbes NURBS : plus le poids ω_{i_1,i_2} est élevé, plus la surface sera "attirée" par le point de contrôle $\mathbf{P}_{i_1,i_2}$ et, à l'inverse, plus il sera bas, plus la surface va s'éloigner du point de contrôle.

Les propriétés énoncées précédemment pour les courbes NURBS restent valables pour les surfaces :



(a) Surface B-Spline $\{\omega_{i_1,i_2} = 1\}$ (b) Surface NURBS $\{\omega_{7,5} = \frac{1}{2}, \omega_{3,4} = 3\}$

FIGURE 3.4 – Exemple de surfaces NURBS et B-Spline avec les mêmes degrés, points de contrôle et *knot-vector*

- *Propriété de support local.* Comme $N_{i_k,p_k}\left(u^{(k)}\right) = 0$ si $u^{(k)}$ est en dehors de l'intervalle $\left[U^{(k)}_{i_k}, U^{(k)}_{i_k+p_k+1}\right[$, $k = 1, 2$, il est évident que :

$$R_{i_1,i_2}\left(u^{(1)}, u^{(2)}\right) \neq 0 \quad \Leftrightarrow \left(u^{(1)}, u^{(2)}\right) \in \left[U^{(1)}_{i_1}, U^{(1)}_{i_1+p_1+1}\right[\times \left[U^{(2)}_{i_2}, U^{(2)}_{i_2+p_2+1}\right[. \tag{3.20}$$

  Le rectangle ouvert $\left[U^{(1)}_{i_1}, U^{(1)}_{i_1+p_1+1}\right[\times \left[U^{(2)}_{i_2}, U^{(2)}_{i_2+p_2+1}\right[$ est le support local associé au point de contrôle $\mathbf{P}_{i_1,i_2}$. Ceci peut également être observé sur la fig. 3.4 où l'impact des poids est localisé dans une certaine zone autour des points de contrôle auxquels ils sont reliés alors que le reste de la surface est identique pour la B-Spline et pour la NURBS.

- *Propriété d'enveloppe convexe.* Les fonctions de base étant positives par définition, et la propriété de *partition de l'unité* restant valable pour les surfaces, il est également possible de démontrer que les surfaces NURBS (ou B-Spline) sont toujours contenues dans l'enveloppe convexe du réseau de contrôle [2].

- *Propriété de continuité et de dérivabilité.* Les propriétés de continuité et dérivabilité vues pour les courbes s'appliquent à chaque fonction de base $N_{i_k,p_k}\left(u^{(k)}\right)$, $k = 1, 2$. En particulier, l'éq. (3.13) reste valable pour chaque direction. De ce fait, on peut montrer qu'augmenter la multiplicité λ_k dans la direction k diminue la dérivabilité de la surface dans cette direction.

- *Propriété d'invariance par transformation affine et projection.* Les propriétés vues pour les courbes sont tout à fait transposables aux surfaces.

3.2.3 Hypersurfaces

Nous allons présenter ici le concept des hypersurfaces NURBS. Il s'agit d'une application $\mathbf{H} : \mathbb{R}^N \rightarrow \mathbb{R}^M$, où N est la dimension de l'espace des paramètres et M la dimension de l'hypersurface. Il est possible d'observer que le cas $N = 1$ et $M = 2$ est le cas d'une courbe plane tandis qu'une courbe 3D est caractérisée par $N = 1$ et $M = 3$. Bien évidement, le cas des surfaces est également couvert par cette formulation avec $N = 2$ et $M = 3$. En général, on parlera d'hypersurface pour le cas $N > 3$ et/ou $M > 3$. Dans la suite de ce manuscrit, N représentera la dimension de l'espace des paramètres d'entrée du modèle réduit alors que M représentera le nombre de coordonnées de l'hypersurface, i.e. le nombre de sorties du métamodèle.

La théorie des hypersurfaces NURBS découle directement de celle des surfaces à la différence qu'il n'est pas possible de les représenter visuellement. La formulation générale d'une hypersurface NURBS $\mathbf{H} : \mathbb{R}^N \rightarrow \mathbb{R}^M$ est la suivante :

$$\begin{aligned} &\mathbf{H}\left(u^{(1)}, ..., u^{(N)}\right) = \frac{\sum_{i_1=0}^{n_1} \cdots \sum_{i_N=0}^{n_N} N_{i_1,p_1}\left(u^{(1)}\right) \times ... \times N_{i_N,p_N}\left(u^{(N)}\right) \omega_{i_1,...,i_N} \mathbf{P}_{i_1,...,i_N}}{\sum_{j_1=0}^{n_1} \cdots \sum_{j_N=0}^{n_N} N_{j_1,p_1}\left(u^{(1)}\right) \times ... \times N_{j_N,p_N}\left(u^{(N)}\right) \omega_{j_1,...,j_N}}, \\ &\left(u^{(1)}, ..., u^{(N)}\right) \in [0,1], \end{aligned} \tag{3.21}$$

que nous pouvons également présenter sous la forme réduite suivante :

$$\mathbf{H}\left(u^{(1)}, ..., u^{(N)}\right) = \sum_{i_1=0}^{n_1} ... \sum_{i_N=0}^{n_N} R_{i_1,...,i_N}\left(u^{(1)}, ..., u^{(N)}\right) \mathbf{P}_{i_1,...,i_N}, \quad \left(u^{(1)}, ..., u^{(N)}\right) \in [0,1], \tag{3.22}$$

où la fonction rationnelle par morceaux $R_{i_1,...,i_N}\left(u^{(1)}, ..., u^{(N)}\right)$ est donnée par :

$$\begin{aligned} &R_{i_1,...,i_N}\left(u^{(1)}, ..., u^{(N)}\right) = \frac{N_{i_1,p_1}\left(u^{(1)}\right) \times ... \times N_{i_N,p_N}\left(u^{(N)}\right) \omega_{i_1,...,i_N}}{\sum_{j_1=0}^{n_1} \cdots \sum_{j_N=0}^{n_N} N_{j_1,p_1}\left(u^{(1)}\right) \times ... \times N_{j_N,p_N}\left(u^{(N)}\right) \omega_{j_1,...,j_N}}, \\ &\left(u^{(1)}, ..., u^{(N)}\right) \in [0,1]. \end{aligned} \tag{3.23}$$

Dans l'éq. (3.21), $\mathbf{H}\left(u^{(1)}, ..., u^{(N)}\right) = \left\{H^{(1)}\left(u^{(1)}, ..., u^{(N)}\right), ..., H^{(M)}\left(u^{(1)}, ..., u^{(N)}\right)\right\}$ est l'ensemble des coordonnées du point de l'hypersurface de dimension M aux valeurs de paramètre $\left(u^{(1)}, ..., u^{(N)}\right)$, $N_{i_k,p_k}\left(u^{(k)}\right)$ sont les fonctions de base associées à la direction $k = 1, ..., N$, $\omega_{i_1,...,i_N}$ est le poids associé au point de contrôle $\mathbf{P}_{i_1,...,i_N} = \left\{P^{(1)}_{i_1,...,i_N}, ..., P^{(M)}_{i_1,...,i_N}\right\}$. L'ensemble des $(n_1 + 1) \times ... \times (n_N + 1)$ points forme *l'hyper-réseau de contrôle*. Chacune des directions paramétriques, représentée par le paramètre $u^{(k)}$, nécessite sa propre fonction de base caractérisée par un degré p_k et un nombre $(n_k + 1)$ de points de contrôle, en accord avec l'éq. (3.4). De même que pour les surfaces, chacun des *knot-vectors* est indépendant des autres, les degrés et nombres de points de contrôle l'étant aussi. Le concept des hypersurfaces B-Splines découle naturellement de la formulation NURBS en utilisant la *propriété de partition de l'unité*, i.e.,

$$\sum_{i_1=0}^{n_1} \cdots \sum_{i_N=0}^{n_N} \left(\prod_{k=1}^{N} N_{i_k,p_k}(u^{(k)}) \right) = 1. \tag{3.24}$$

De même que pour les courbes et les surfaces, en introduisant l'éq. (3.24) dans l'éq. (3.21) et en fixant l'ensemble des poids $\{\omega_{i_1,...,i_N}\}$ à la même valeur, on obtient la formulation générale d'une hypersurface B-Spline :

$$\begin{aligned}&\mathbf{H}\left(u^{(1)},...,u^{(N)}\right)=\sum_{i_1=0}^{n_1}...\sum_{i_N=0}^{n_N}N_{i_1,p_1}\left(u^{(1)}\right)\times...\times N_{i_N,p_N}\left(u^{(N)}\right)\mathbf{P}_{i_1,...,i_N},\\&\left(u^{(1)},...,u^{(N)}\right)\in[0,1]\,.\end{aligned}\tag{3.25}$$

Les propriétés précédemment énoncées pour le cas des courbes et surfaces restent valables pour le cas général des hypersurfaces NURBS, notamment :

- *Propriété de support local.* Dans le cas des hypersurfaces le support de chacune des fonctions de base est donné par l'intervalle $\left[U_{i_k}^{(k)},\,U_{i_k+p_k+1}^{(k)}\right[$, ce qui conduit à la propriété suivante :

$$R_{i_1,...,i_N}\left(u^{(1)},...,u^{(N)}\right)\neq 0\Leftrightarrow\left(u^{(1)},...,u^{(N)}\right)\in\left[U_{i_1}^{(1)},U_{i_1+p_1+1}^{(1)}\right[\times...\times\left[U_{i_N}^{(N)},U_{i_N+p_N+1}^{(N)}\right[.\tag{3.26}$$

  Où l'hyper-rectangle ouvert $\left[U_{i_1}^{(1)},U_{i_1+p_1+1}^{(1)}\right[\times...\times\left[U_{i_N}^{(N)},U_{i_N+p_N+1}^{(N)}\right[$ est le support local associé au point de contrôle $\mathbf{P}_{i_1,...,i_N}$.
- *Propriété d'enveloppe convexe.* Les fonctions de base étant positives par définition pour chaque direction de l'espace des paramètres, et la propriété de *partition de l'unité* restant valable pour les hypersurfaces NURBS, il est également possible de démontrer qu'elles sont toujours contenues dans l'enveloppe convexe de l'hyper-réseau de contrôle.
- *Propriété de continuité et de dérivabilité.* Les propriétés de continuité et dérivabilité vues pour les courbes et surfaces s'appliquent également au cas des hypersurfaces, chaque fonction de base $N_{i_k,p_k}\left(u^{(k)}\right)$, $k=1,...,N$, ayant les mêmes propriétés. Ainsi on notera que l'hypersurface NURBS est $p_k-\lambda_{l_k}$ fois dérivable au nœud $U_{l_k}^{(k)}$ dans la direction k, λ_{l_k} étant sa multiplicité. Notons également que l'hypersurface est infiniment dérivable entre chaque *knot* et que l'éq. (3.13) s'applique également à chacune des fonctions de base.
- *Propriété d'invariance par transformation affine et projection.* Les propriétés vues pour les courbes et les surfaces sont tout à fait transposables aux hypersurfaces NURBS.

S'il existe un grand nombre d'outils permettant de traiter les courbes et surfaces NURBS, de part leur implémentation dans la plupart des logiciels de CAO, ce n'est pas le cas des hypersurfaces. Nous avons donc développé, dans le cadre de cette thèse, un grand nombre de routines permettant de les traiter. Les premiers essais, effectués dans le langage MATLAB, nous ont très vite conduit à la conclusion que ce langage n'était pas adapté au traitement des hypersurfaces (notamment quand la dimension de l'espace paramétrique N augmente), principalement parce que les fonctions de base sont définies récursivement et donc difficilement vectorisables. Les routines développées étant la généralisation des algorithmes proposés par [2], présentés en langage C, nous avons choisi de les coder dans le langage C++ (afin de pouvoir utiliser, à termes, la programmation orientée objet). Il est, de plus, possible d'utiliser des fonctions codées en langage C, C++ ou encore FORTRAN dans MATLAB, grâce à l'utilisation de fichier "mex" [206], comme si elles étaient des fonctions classiques de MATLAB. Le choix a donc été fait de développer ces routines dans le format "mex" adapté au C++, afin de conserver la versatilité de MATLAB

d'une part, et d'être facilement interfacé avec les algorithmes d'optimisation développés à l'I2M d'autre part. Ci-après, les principales routines, utilisées notamment pour le calcul d'un point de l'hypersurface NURBS, sont fournies sous formes de pseudo-code (le code en format "mex" est fourni en annexe A :

- Calcul de l'indice de portée (*span index*). Cet indice permet de déterminer l'intervalle $\left[U_{i_k}^{(k)}, U_{i_k+1}^{(k)}\right]$, du *knot vector* $\mathbf{U}^{(k)}$, dans lequel se trouve la coordonnée paramétrique $u^{(k)}$ (i.e. on cherche i_k tel que $u^{(k)} \in \left[U_{i_k}^{(k)}, U_{i_k+1}^{(k)}\right]$). L'algorithme est identique à celui proposé par [2] (algorithme A2.1 p.68) et peut être utilisé pour n'importe quelle direction k de l'espace paramétrique :

Algorithm 1 Calculate $i_k = \text{findSpan}(n_k, p_k, u^{(k)}, \mathbf{U}^{(k)})$

Input : $n_k + 1$ number of control points along direction k, p_k degree of the basis function along direction k, $u^{(k)}$ parametric coordinate along direction k and $\mathbf{U}^{(k)}$ knot vector of $n_k + p_k + 2$ values along direction k.

Output : i_k span index of the parametric coordinate $u^{(k)}$ along direction k.

if $u^{(k)} = U^{(k)}[n_k+1]$ **then** // Note that indexes of vector $\mathbf{U}^{(k)}$ are in $\{0, ..., n_k + p_k + 1\}$
  return n_k
else
  $low = p_k$
  $high = n_k + 1$
  $mid = \dfrac{low + high}{2}$
  while $u^{(k)} < U^{(k)}[mid]$ or $u^{(k)} \geq U^{(k)}[mid + 1]$ **do**
    if $u^{(k)} < U^{(k)}[mid]$ **then**
      $high = mid$
    else
      $low = mid$
    end if
    $mid = \dfrac{low + high}{2}$
  end while
  return mid
end if

- Calcul des fonctions de base. L'indice de portée ayant été trouvée, il est possible de calculer les fonctions de base associées à la coordonnée paramétrique $u^{(k)}$. Un grand nombre d'entre-elles peuvent cependant être nulles (cf. *propriété de support local*) et la routine permet de calculer seulement les $p_k + 1$ valeurs non nulles. De même que précédemment, l'algorithme est celui proposé par [2] (algorithme A2.2 p.70) et peut être utilisé pour n'importe quelle direction k de l'espace paramétrique :

Algorithm 2 Calculate $\mathbf{N}_{u^{(k)}}^{(k)} = \text{basisFun}(i_k, u^{(k)}, p_k, \mathbf{U}^{(k)})$

Input : i_k span index of the parametric coordiante $u^{(k)}$ along direction k, $u^{(k)}$ parametric coordinate, p_k degree of the basis function along direction k and $\mathbf{U}^{(k)}$ knot vector of $n_k + p_k + 2$ values along direction k.

Output : $\mathbf{N}^{(k)}_{u^{(k)}}$ vector containing the $p_k + 1$ non-zero values of the basis functions related to the parametric coordinate $u^{(k)}$ and the knot vector $\mathbf{U}^{(k)}$.

$N^{(k)}_{u^{(k)}}[0] = 1$
for $s = 1 : p_k$ **do**
  $left[s] = u^{(k)} - U^{(k)}[i_k + 1 - s]$
  $right[s] = U^{(k)}[i_k + s] - u^{(k)}$
  $saved = 0$
  for $r = 0 : s$ **do**
    $$temp = \frac{N^{(k)}_{u^{(k)}}[r]}{right[r+1] + left[s-r]}$$
    $N^{(k)}_{u^{(k)}}[r] = saved + right[r+1] \times temp$
    $saved = left[s-r] \times temp$
  end for
  $N^{(k)}_{u^{(k)}}[s] = saved$
end for
return $\mathbf{N}^{(k)}_{u^{(k)}}$

- Calcul d'un point de l'hypersurface NURBS. Le calcul d'un point de l'hypersurface utilise N fois les deux routines précédentes, N étant la dimension de l'espace paramétrique. L'algorithme présenté ici est la version que nous avons généralisée de l'algorithme A3.5 (p.103) de [2] et permet de calculer les M coordonnées de l'hypersurface NURBS :

Algorithm 3 Calculate $\mathbf{H}$ = hyperSurfacePoint(n_1, p_1, u_1, $\mathbf{U}^{(1)}$, ..., n_N, p_N, u_N, $\mathbf{U}^{(N)}$, ω, P_1, ..., P_M)

Input : n_k+1 the number of control points, p_k degree of the basis function, $u^{(k)}$ parametric coordinate, $\mathbf{U}^{(k)}$ knot vector of n_k+p_k+2 values along direction k, with $k = 1, ..., N$, ω N-D array of weights and $P_1, \ldots, P_M$ arrays of control points coordinates along hypersurface axis $1, \ldots, M$.

Output : $\mathbf{H}$ hypersurface point coordinates at parametric coordinates $\mathbf{u} = (u^{(1)}, \ldots, u^{(N)})$.

for $k = 1 : N$ **do**
  $span_k = \text{findSpan}(n_k, p_k, u^{(k)}, \mathbf{U}^{(k)})$
  $\mathbf{N}^{(k)}_{u^{(k)}} = \text{basisFun}(span_k, u^{(k)}, p_k, \mathbf{U}^{(k)})$
  $ind_k = span_k - p_k$
end for
$temp_1[N-1] = 0$
$\vdots$
$temp_M[N-1] = 0$
$W[N-1] = 0$

for $i_N = 0 : p_N$ **do**
$temp_1[N-2] = 0$
$\vdots$
$temp_M[N-2] = 0$
$W[N-2] = 0$
for $i_{N-1} = 0 : p_{N-1}$ **do**
$\vdots$
for $i_2 = 0 : p_2$ **do**
$temp_1[0] = 0$
$\vdots$
$temp_M[0] = 0$
$W[0] = 0$
for $i_1 = 0 : p_1$ **do**
$temp_1[0] = temp_1[0] + N^{(1)}_{u^{(1)}}[i_1] \times \omega[ind_1 + i_1, \ldots, ind_N + i_N] \times P_1[ind_1 + i_1, \ldots, ind_N + i_N]$
$\vdots$
$temp_M[0] = temp_M[0] + N^{(1)}_{u^{(1)}}[i_1] \times \omega[ind_1 + i_1, \ldots, ind_N + i_N] \times P_M[ind_1 + i_1, \ldots, ind_N + i_N]$
$W[0] = W[0] + N^{(1)}_{u^{(1)}}[i_1] \times \omega[ind_1 + i_1, \ldots, ind_N + i_N]$
end for
$temp_1[1] = temp_1[1] + N^{(2)}_{u^{(2)}}[i_2] \times temp_1[0]$
$\vdots$
$temp_M[1] = temp_M[1] + N^{(2)}_{u^{(2)}}[i_2] \times temp_M[0]$
$W[1] = W[1] + N^{(2)}_{u^{(2)}}[i_2] \times W[0]$
end for
$\vdots$
end for
$temp_1[N-1] = temp_1[N-1] + N^{(N)}_{u^{(N)}}[i_N] \times temp_1[N-2]$
$\vdots$
$temp_M[N-1] = temp_M[N-1] + N^{(N)}_{u^{(N)}}[i_N] \times temp_M[N-2]$
$W[N-1] = W[N-1] + N^{(N)}_{u^{(N)}}[i_N] \times W[N-2]$
end for
$\mathbf{H} = \left\{ \frac{temp_1[N-1]}{W[N-1]}, \ldots, \frac{temp_M[N-1]}{W[N-1]} \right\}$
return H

Ces différents algorithmes permettent de remarquer que le calcul d'un point de l'hypersurface (i.e. d'une réponse du métamodèle dans notre cas) nécessite la définition d'un certain nombre de paramètres :

- Le nombre de points de contrôle $n_k + 1$ dans chaque direction $k = 1, ..., N$, de l'espace paramétrique,

- Le degré des fonctions de base p_k dans chaque direction $k = 1, ..., N$, de l'espace paramétrique,
- Les *knot vectors* $\mathbf{U}^{(k)}$ dans chaque direction $k = 1, ..., N$, de l'espace paramétrique,
- Les coordonnées des points de contrôle $\mathbf{P}^{(j)}_{i_1,...,i_N}$, $i_k = 0, \ldots, n_k$, dans chaque dimension $j = 1, \ldots, M$, de l'hypersurface
- Les poids $\omega_{i_1,...,i_N}$, $i_k = 0, \ldots, n_k$, qui leurs sont associés.

L'inconvénient majeur est que ces paramètres ne sont pas tout à fait indépendants et, en particulier, la taille du *knot vector* $\mathbf{U}^{(k)}$ est directement liée au nombre de points de contrôle $n_k + 1$ et au degré p_k par la relation (3.6), alors que celles des matrices multidimensionnelles collectant les coordonnées des points de contrôle et les poids ne dépend que des n_k. C'est pourquoi, dans la plupart des méthodes actuelles, des hypothèses sont faites sur certains des paramètres afin que les seuls paramètres à identifier soient les nombres de points de contrôle $n_k + 1$ dans chaque direction de l'espace paramétrique ainsi que les valeurs de leurs coordonnées. En général, les degrés sont fixés et les valeurs des différents *knot vectors*, ainsi que celles des poids, sont déterminées par des règles empiriques [1,2]. Dans la suite de ce chapitre, nous présenterons la notion de système modulaire et les problèmes d'optimisation associés. Nous verrons dans le chapitre 4 que le problème d'optimisation des paramètres d'une hypersurface NURBS peut être vu comme un problème d'optimisation de système modulaire, ayant la particularité de posséder plusieurs types de modules.

3.2.4 Conclusions sur les entités NURBS

Les courbes et les surfaces NURBS sont vues, par certains, comme étant les "enfants" des entités B-Splines, elles-mêmes pouvant être considérées comme les "enfants" des courbes et surfaces de Béziers. Nous l'avons en effet vu dans ce chapitre, la formulation NURBS inclut dans son formalisme les deux cas précédents. Les entités NURBS représentent, cependant, une évolution importante car elles sont capables de représenter de manière exacte des formes géométriques que les B-Splines sont seulement capables d'approximer. Ceci, associé aux *propriétés d'invariance par transformation affine* et à la stabilité des algorithmes de calcul, explique leur développement massif dans les outils CAO où elles sont devenues la représentation standard des courbes et surfaces. Le cas généralisé des hypersurfaces que nous avons introduit précédemment est, quant à lui, bien moins standardisé. Il existe quelques exemples d'utilisation d'hypersurfaces NURBS dont [137] pour la modélisation d'animation ou encore dans le cadre de l'optimisation topologique [207–210]. Elles ont également été récemment utilisées pour la modélisation et l'optimisation multi-échelle des composites à rigidité variable obtenus à l'aide de la technologie AFP (*automated fibre placement*) [17,18,211,212]. Dans le cadre de cette thèse, ce formalisme est utilisé pour décrire de façon très générale, et sans introduire aucune hypothèse simplificatrice, le comportement physique de divers systèmes dans une logique de réduction de modèle. A notre connaissance, même s'il est possible de trouver beaucoup d'exemples d'algortihmes pour le cas des courbes et des surfaces [2,207] , il n'existe pas, hormis les travaux de Turner [1], des méthodes de réduction de modèle basées sur les hypersurfaces NURBS.

Même si l'utilisation des hypersurfaces NURBS est relativement nouvelle, l'algorithme itératif utilisé par Turner est basé sur la généralisation d'algorithmes utilisés pour les courbes et surfaces.

Notamment, beaucoup de paramètres, tels que les degrés p_k ou les *knot vectors*, sont fixés par des règles métiers sans apporter aucune justification de nature physique/géométrique pertinente. De plus, la détermination de la valeur optimale de ces paramètres est un problème non-convexe de dimension variable. D'autre part, dans le domaine de la CAO, les entités NURBS sont essentiellement utilisées sous forme de courbes et surfaces et dans ce cas, même si l'optimisation des paramètres permet de réduire le nombre de points de contrôle nécessaire pour la représentation de celles-ci, un nombre un peu plus élevé que nécessaire ne génère pas de limitations pour les capacités actuelles des ordinateurs. De ce fait, très peu d'études ont été menées pour éviter l'utilisation de règles empiriques dans la sélection des paramètres des NURBS. L'étude des hypersurfaces NURBS, dans le cadre de la réduction de modèle, remet en cause cette tendance et c'est pourquoi la méthode que nous proposons vise à fournir à l'utilisateur les paramètres optimaux définissant le métamodèle sans aucune hypothèse simplificatrice. Pour cela, la recherche des paramètres de l'hypersurface NURBS passe par la résolution d'un problème d'optimisation dont nous allons voir la formulation au chapitre 4. La suite de ce chapitre présente les généralités de l'optimisation ainsi que les deux méthodes employées dans notre étude.

3.3 Méthodes numériques d'optimisation

3.3.1 Introduction sur l'optimisation

Dans un cadre général, l'*optimisation* peut être définie par la sélection du meilleur élément parmi un ensemble d'alternatives possibles au regard de certains critères. Du point de vue mathématique, un problème d'optimisation a pour but de *minimiser* ou *maximiser* une *fonction objectif* f qui dépend de plusieurs paramètres. Ces paramètres sont appelés *variables d'optimisation* ou *variables de conception* selon le contexte et sont représentés par un vecteur $\mathbf{x} \in \mathbb{R}^n$. Les variables d'optimisation représentent les inconnues du problème et la dépendance de la fonction objectif f par rapport à ces variables est notée $f(\mathbf{x})$. En pratique, il est très rare d'avoir un problème d'ingénierie caractérisé seulement par la fonction objectif et ses variables d'optimisation associées. La formulation du problème s'accompagne généralement de m_e *contraintes d'égalité* de la forme $h_i(\mathbf{x}) = 0$, $i = 1, ..., m_e$, et/ou de m_i *contraintes d'inégalité* de la forme $g_j(\mathbf{x}) \leq 0$, $j = 1, ..., m_i$. Ces contraintes permettent de formaliser des exigences physiques ou techniques qui dépendent du problème considéré. Un point $\mathbf{x}_f$ respectant toutes les contraintes est dit *faisable* tandis qu'un point $\mathbf{x}_u$ ne respectant pas une contrainte, ou plus, est dit *infaisable*. Un problème d'optimisation est généralement posé comme un problème de minimisation sous contraintes, (la maximisation pouvant être ramenée très simplement à un problème de minimisation) de la manière suivante :

$$\begin{gathered} \min_{\mathbf{x}} f(\mathbf{x}), \\ \text{sujet à :} \\ \begin{cases} g_i(\mathbf{x}) \leq 0, \ i = 1, ..., m_i, \\ h_j(\mathbf{x}) = 0, \ j = 1, ..., m_e, \\ \mathbf{x}_{LB} \leq \mathbf{x} \leq \mathbf{x}_{UB}. \end{cases} \end{gathered} \tag{3.27}$$

Dans l'éq. (3.27), les vecteurs $\mathbf{x}_{LB}$ et $\mathbf{x}_{UB}$ représentent respectivement les bornes inférieures et supérieures de la variable $\mathbf{x}$. Ce problème est conventionnellement appelé *problème de programmation non-linéaire sous contraintes* ou CNLPP pour *Constrained Non-Linear Programming*

Problem. Sauf cas particulier très rare, le problème (3.27) ne possède pas une solution analytique et il faut utiliser un algorithme adapté pour chercher la solution. Le choix de l'algorithme dépend de plusieurs facteurs dont la nature des variables d'optimisation, la présence ou non de contrainte et la nature à la fois de la fonction objectif et des fonctions contraintes (continuité, convexité, linéarité, ...). Les principaux critères sont listés ci-dessous :

- *Présence de contrainte(s).* Même si généralement les problèmes d'ingénierie impliquent un grand nombre de contraintes, il existe une variété de problèmes sans contraintes comme par exemple les problèmes de minimisation des moindres carrés dans le contexte de l'ajustement de courbe. Ces problèmes présentent un réel intérêt car les problèmes d'optimisation sous contraintes sont souvent résolus par la formulation d'un problème sans contraintes équivalent.

- *Linéarité.* Si les fonctions objectif et contraintes sont des fonctions linéaires des variables d'optimisation $\mathbf{x}$, alors le problème est linéaire. La particularité des méthodes de résolution de cette classe de problèmes est qu'elles n'utilisent, au plus, que le gradient des fonctions objectifs et contraintes (en effet, la matrice hessienne n'est constitué que de valeurs nulles dans ce cas).

- *Convexité.* La convexité du problème est une notion importante car elle détermine en grande partie la stratégie à employer pour la recherche de la solution optimale. La notion de convexité est définie dans le cas des fonctions et est étendue aux problèmes d'optimisation si les conditions suivantes sont respectées : a. La fonction f est convexe ; b. Les fonctions contraintes d'égalité h_j, $j = 1, ..., m_e$ sont convexes ; c. Les fonctions contraintes d'inégalité g_i, $i = 1, ..., m_i$ sont convexes.

- *Continuité.* Un problème de minimisation est considéré continu si les fonctions objectif et contraintes sont continues et que toutes les variables d'optimisation sont continues dans $\mathbb{R}^n$ ou dans un sous-espace $\Omega \subset \mathbb{R}^n$. La discontinuité du problème peut donc venir des fonctions et/ou des variables d'optimisation. Si les variables sont discrètes, reparties régulièrement ou éparpillées (i.e. avec un pas de discrétisation variable), il est nécessaire d'utiliser une méthode adaptée à la résolution de ce type de problème.

- *Nature du modèle.* La nature des variables d'optimisation ainsi que des fonctions objectif et contraintes joue un rôle prépondérant dans le choix de l'algorithme de recherche de solution qui sera utilisé. En effet, lorsqu'un problème est convexe et qu'il est continu, alors les algorithmes déterministes sont les plus adaptés à la recherche de solution. Si cependant certaines variables sont caractérisées par des incertitudes, les algorithmes stochastiques sont les mieux adaptés. Enfin, dans le cas de problèmes non-convexes, à variables ou fonctions non continues, la classe des métaheuristiques est parfois la seule capable d'effectuer la recherche de la solution.

On définit un minimiseur global $\mathbf{x}_g^*$ de f comme étant un point pour lequel la fonction f est miniminale sur le domaine faisable, i.e.

$$f(\mathbf{x}_g^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega_f, \tag{3.28}$$

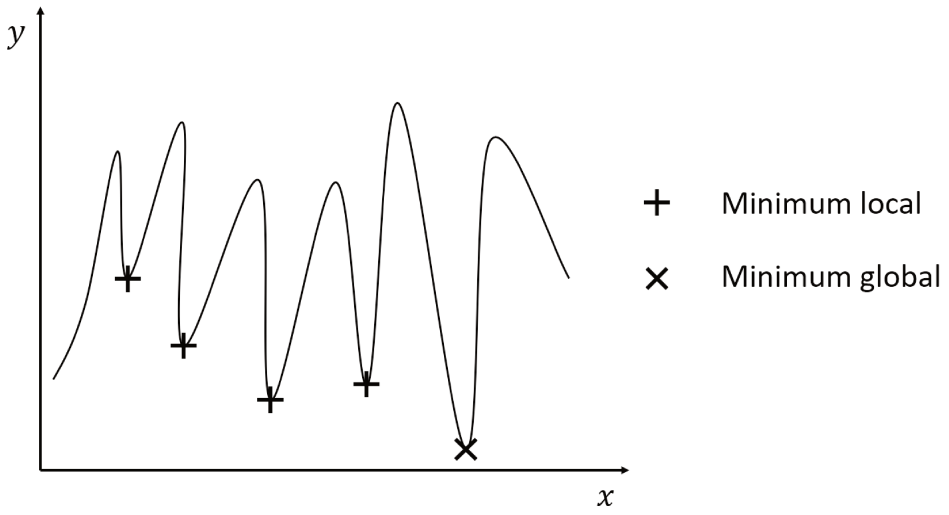


FIGURE 3.5 – Exemple de fonction non-convexe

où Ω_f représente le domaine faisable :

$$\forall \mathbf{x} \in \Omega_f, \begin{cases} h_j(\mathbf{x}) = 0, & j = 1, ..., m_e, \\ g_i(\mathbf{x}) \leq 0, & i = 1, ..., m_i. \end{cases} \tag{3.29}$$

La valeur de la fonction en ce point $f(\mathbf{x}_g^*)$ est appelée *minimum global*, *optimum global* ou encore *solution globale*. Par opposition, un minimiseur local $\mathbf{x}_l^*$ de f est un point pour lequel la fonction f n'admet pas de valeur inférieure à $f(\mathbf{x}_l^*)$ dans un voisinage proche de $\mathbf{x}_l^*$, i.e.

$$f(\mathbf{x}_l^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in V(\mathbf{x}_l^*), \tag{3.30}$$

où $V(\mathbf{x}_l^*)$ est un voisinage de $\mathbf{x}_l^*$. La valeur $f(\mathbf{x}_l^*)$ est alors appelée *minimum local*, *optimum local* ou encore *solution locale*. On peut montrer facilement que si f est convexe, alors n'importe quel minimum local est un minimum global. De plus, si f est dérivable, alors n'importe quel point stationnaire $\mathbf{x}^*$ est un minimiseur global [213]. Un exemple simple de fonction non-convexe est fourni à la fig. 3.5. Il existe principalement trois grandes classes d'algorithmes de recherche de solution :

- Les *algorithmes déterministes* basés sur l'utilisation des dérivées premières et secondes des fonctions objectif et contraintes. Ce sont des algorithmes rapides mais ils nécessitent que le problème soit continu. De plus, dans le cas d'un problème non-convexe, la solution trouvée dépend fortement du point de départ de l'algorithme et renvoie souvent un minimum local de la fonction f.
- Les *algorithmes stochastiques* qui utilisent les mêmes fondements que les algorithmes déterministes en ce qui concerne l'utilisation des dérivées. Ils permettent, cependant, d'inclure des caractéristiques d'incertitudes sur les variables en prenant en compte une distribution de probabilité sur les variables concernées.

- Les *algorithmes métaheuristiques* basés sur l'insertion d'une composante aléatoire dans la recherche de la solution, permettant ainsi une exploration efficace du domaine. Ils sont particulièrement adaptés à la recherche de solution de problèmes non-convexes. De plus, ils ne sont impactés ni par les discontinuités des variables d'optimisation ni par celles des fonctions objectif et contraintes puisqu'ils n'utilisent pas les dérivées de celles-ci.

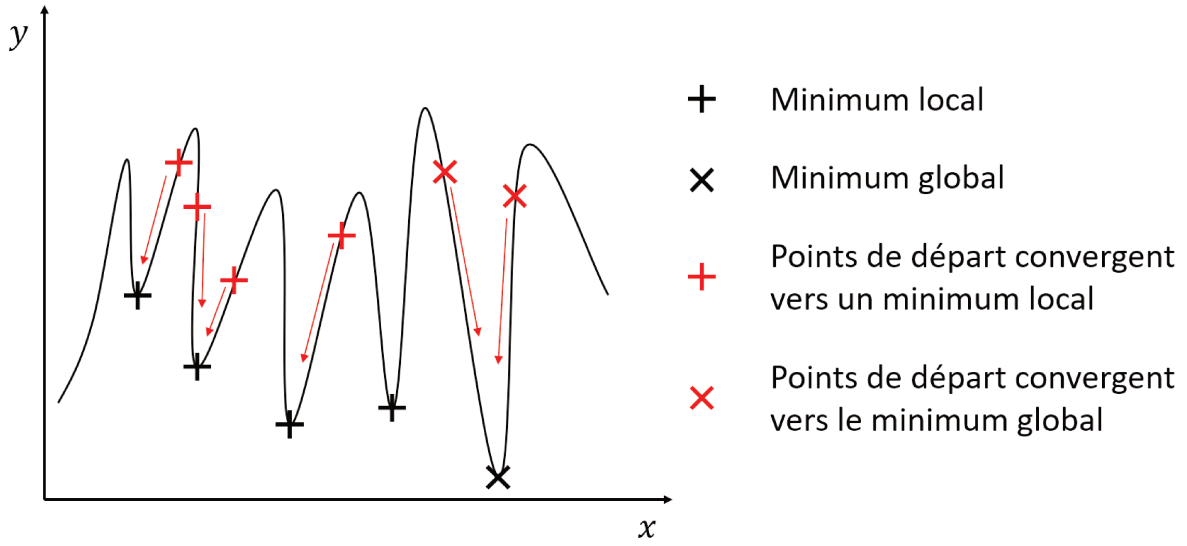


FIGURE 3.6 – Influence du point de départ pour la résolution de problème d'optimisation par des méthodes déterministes

Les algorithmes déterministes sont généralement beaucoup plus rapides que les métaheuristiques. L'inconvénient est que, dans le cas d'un problème non-convexe, la solution renvoyée dépend en grande partie du point de départ choisi. La fig. 3.6 montre l'influence du point de départ sur la solution renvoyée par l'algorithme pour la fonction de la fig. 3.5. Comme on peut le constater, un grand nombre de points de départ vont conduire à un minimum local. Dans ce cas $1D$, le point de départ doit se trouver dans le "creux" contenant le minimum global. Tout autre point de départ conduira l'algorithme à trouver un minimum local. Malheureusement, la plupart des problèmes industriels sont intrinsèquement non-convexes, ce qui rend la recherche des solutions globales difficile pour un grand nombre d'entres eux. Si le problème est continu (variables et fonctions), il est possible d'utiliser un algorithme déterministe en utilisant plusieurs points de départ afin d'explorer le domaine. Cependant, il n'existe pas de méthode efficace pour choisir ces points de départ et l'exploration n'est pas aussi efficace qu'avec une métaheuristique. Dans notre étude, la recherche des paramètres optimaux caractérisant l'hypersurface NURBS est un problème non-convexe ayant conduit à l'élaboration de nombreuses règles empiriques [2] afin d'éluder la recherche de ces paramètres. De plus, la dimension de l'espace de recherche est variable (cet aspect est développé au chapitre 4) et très peu de méthodes sont capables de gérer cette spécificité. Dans la suite de ce chapitre, les deux algorithmes d'optimisation utilisés par la méthode hybride que nous avons développée seront présentés. Nous nous appuyons sur deux

types d'algorithmes dont l'un est une métaheuristique de type *algorithme génétique* permettant d'exploiter efficacement le domaine des variables d'optimisation et le second est un algorithme déterministe. L'algorithme génétique employé a spécialement été conçu pour traiter les problèmes d'optimisation de systèmes modulaires dont la notion sera introduite dans ce chapitre (problèmes dont la dimension de l'espace d'optimisation est variable). Il a été proposé par Montemurro [14] et généralisé dans le cadre de cette thèse afin, notamment, de s'adapter à des problèmes d'optimisation de systèmes modulaires possédant différents types de modules. Initialement conçu en langage *FORTRAN*, le code a été réécrit en langage *MATLAB*, ce qui nous a conduit à utiliser les algorithmes déterministes proposés par l'*Optimization Toolbox* [214] de cet outil. Ces deux méthodes sont détaillées dans le reste du chapitre. Nous verrons dans le chapitre 4 le lien entre l'optimisation de systèmes modulaires et l'optimisation d'une hypersurface NURBS.

3.3.2 Une métaheuristique pour la résolution de problèmes d'optimisation à dimension variable ERASMUS

Introduction

Les métaheuristiques forment une vaste classe de méthodes créées typiquement dans le but de résoudre des CNLPPs non-convexes. Elles sont largement rependues dans la littérature et utilisent des règles empiriques inspirées des phénomènes naturels pour explorer le domaine des variables d'optimisation. On les considère comme des méthodes *globales* d'optimisation parce qu'elles permettent une meilleure exploration que les méthodes déterministes : elles utilisent une population de points plutôt que de baser la recherche sur un seul point. Il faut toutefois noter, qu'en théorie, une métaheuristique est capable d'atteindre un optimum global mais rien ne garantit que la solution trouvée en soit un. Il existe un grand nombre de techniques différentes dont les plus connues sont :

- L'optimisation par *colonies de fourmis* [215]. Cet algorithme simule le comportement d'un certain nombre de fourmis ou d'*agents coopératifs*, qui explore l'espace des solutions à la recherche de zones localement productives.
- L'optimisation par *essaims particulaires* [216]. Cet algorithme simule le comportement d'une population, ou *essaim*, composée de solutions candidates, ou *particules*, qui se déplacent dans l'espace des solutions. Le déplacement est déterminé par la meilleur position atteinte par la particule, la meilleur position de l'ensemble des particules et une composante aléatoire.
- L'optimisation par *système immunitaires artificiel* [217]. Ce type d'algorithme simule le comportement du système immunitaire des vertébrés dans lequel les solutions sont représentées par des anticorps dont les variables de conception sont les gênes. L'évolution des anticorps est régit par le meilleur anticorps généré et une composante aléatoire.
- L'optimisation par *algorithme génétique* [218, 219]. Les algorithmes génétiques sont basés sur le concept de la *sélection naturelle* et les mécanismes de la génétique. Initialement une ou plusieurs populations composées d'individus, dont le génotype représente les variables d'optimisation, sont générées ; puis une opération de sélection permet de choisir les individus qui seront utilisés pour la phase de reproduction. Une composante aléatoire est ajoutée

par des opérations de mutation.

Cette présentation n'a pas pour but de lister de manière exhaustive l'ensemble des métaheuristiques existantes et de leurs variantes, le lecteur intéressé par le domaine est donc renvoyé vers la référence [220]. Le reste de ce chapitre se concentre sur un algorithme génétique particulier développé initialement par Vincenti [15] puis enrichi par Montemurro [14]. Après une brève description des algorithmes génétiques, les spécificités relatives à l'algorithme utilisé sont développées pour terminer par les modifications qui y ont été apportées.

Généralités sur les algorithmes génétiques

Les algorithmes génétiques ont été initialement introduit par Holland, son équipe de recherche et ses étudiants [218, 219]. Ils sont basés, d'une part, sur le concept de l'évolution de Darwin [221], et d'autre part, sur les mécanismes de la reproduction génétique. Ils peuvent être considérés comme des algorithmes d'exploration présentant une grande partie des caractéristiques de la *sélection naturelle*. En effet, naturellement, seuls les individus les mieux adaptés à un environnement survivent et ont une chance de se reproduire, fournissant ainsi aux générations suivantes un patrimoine génétique plus *adapté* que celui de la génération précédente.

Le vocabulaire employé par ce type d'algorithmes est celui de la génétique. Il simule l'évolution d'une *population* de *génération* en *génération* en fonction de la capacité de survie de ses *individus*. Chaque individu représente un point de l'espace d'optimisation et est formé de *chromosomes* dont l'ensemble constitue le *génotype* de l'individu. Les chromosomes sont à leurs tours formés de *gènes* qui représentent les valeurs des variables d'optimisation du problème. Ces valeurs sont en fait codées, généralement par un alphabet binaire, et l'ensemble des valeurs décodées des gènes forme le *phénotype* de l'individu. Le phénotype peut être vu comme l'expression physique du génotype d'un individu, dont le sens est défini de manière externe par l'utilisateur. Dans un organisme, le phénotype inclue des caractérisations physiques comme la couleur des yeux ou des cheveux alors que, dans le cadre des algorithmes génétiques, il représente l'ensemble des valeurs possibles (réelles, discrètes, éparpillés) qu'une variable du problème considéré peut prendre.

L'encodage des gènes à partir des valeurs possibles que peuvent prendre les variables d'optimisation est généralement fait à partir d'un alphabet binaire. Dans un premier temps, chaque variable est discrétisée. Dans le cas d'une variable réelle, le pas de discrétisation est choisi automatiquement par l'algorithme en fonction de la précision machine (i.e. le nombre de bits maximal utilisé pour les calculs). Considérons par exemple une variable $x_i \in [x_{LB}^{(i)}, x_{UB}^{(i)}] \subset \mathbb{R}$ avec un pas de discrétisation Δx_i, les valeurs possibles pour cette variable sont stockés dans le vecteur $\mathbf{x}_i^d$ suivant :

$$\mathbf{x}_i^d = x_{LB} + (\mathbf{I}^{(i)} - 1)\Delta x_i, \tag{3.31}$$

avec $\mathbf{I}^{(i)}$ le vecteur de la forme suivante,

$$\mathbf{I}^{(i)} = \begin{pmatrix} 1 & 2 & \cdots & n_{var}^{(i)} \end{pmatrix}, \tag{3.32}$$

et $n_{var}^{(i)}$ le nombre de valeurs possibles donné par :

$$n_{var}^{(i)} = 1 + \frac{x_{UB}^{(i)} - x_{LB}^{(i)}}{\Delta x_i}. \tag{3.33}$$

Notons qu'il serait également possible d'avoir un pas Δx_i non constant qui permettrait de traiter le cas des valeurs discrètes éparpillées. La relation liant le phénotype au génotype est *bijective* ce qui rend les algorithmes génétiques complètement indépendant du domaine traité. Dans un second temps, l'encodage binaire est fait sur les composantes de $\mathbf{I}^{(i)}$ et le nombre de bits $n_{bit}^{(i)}$ attribué au codage dépend de la valeur de $n_{var}^{(i)}$:

$$n_{bit}^{(i)} = \left\lceil \frac{\ln(n_{var}^{(i)})}{\ln(2)} \right\rceil, \tag{3.34}$$

avec $\lceil \cdot \rceil$ la fonction partie entière par excès.

Considérons un exemple simple de CNLPP non-convexe à résoudre :

$$\begin{aligned} &\min_{x_1,x_2} f(x_1,x_2) = -e^{ka\sqrt{x_1^2+x_2^2}} \sin(ax_1)\cos(2bx_2), \\ &\text{sujet à : } \begin{cases} 0 \leq x_1 \leq 4\pi, \\ 0 \leq x_2 \leq 2\pi, \end{cases} \end{aligned} \tag{3.35}$$

avec des pas de discrétisation $\Delta x_1 = \dfrac{\pi}{20}$ et $\Delta x_2 = \dfrac{\pi}{10}$. Dans ce cas, la structure de l'individu est constituée d'un chromosome de deux gènes, chacun correspondant à une des variables x_i. Les nombres de bits associés pour le codage du génotype sont $n_{bit}^{(1)} = 7$ et $n_{bit}^{(2)} = 5$. Le tableau 3.1 répertorie les informations d'un individu de la population initiale. Le code génétique de cet individu correspond aux indices $I^{(1)} = 35$ et $I^{(2)} = 10$ et se traduit par les valeurs de phénotype $x_1 = 5.3407$ et $x_2 = 3.1416$.

Structure de l'individu	Gène 1	Gène 2
Génotype	0100011	01010
Indices	35	10
Phénotype	5.3407	3.1416

Tableau 3.1 – Structure d'un individu pour le problème (3.35)

La première population est générée aléatoirement, ensuite l'évolution de cette population est perpétrée au travers de 4 opérateurs fondamentaux des algorithmes génétiques jusqu'à ce qu'un critère d'arrêt soit atteint :

- *Adaptation.* Dans un premier temps, la capacité d'adaptation des individus composant la population est évaluée grâce à une fonction nommée *fitness function.* C'est une fonction scalaire qui admet en entrée les valeurs de la fonction objectif et retourne une valeur comprise entre 0 et 1. Il existe plusieurs choix possible [14] mais dans tous les cas, la valeur 0 est attribuée au pire individu de la génération et la valeur 1 est attribuée au meilleur individu de la génération. Il est évident que la fonction objectif ainsi que les fonctions contraintes doivent être évaluées avant la phase d'adaptation.
- *Sélection.* Si l'on suppose que la population est composée de N_{ind} individus, alors l'opération de sélection consiste à choisir dans la population les $N_{ind}/2$ couples de parents pour la

phase de reproduction. Le concept de base est que les individus les mieux adaptés (avec une valeur de *fitness function* élevée) ont plus de probabilité d'être choisis. Le critère de sélection a pour but de reproduire un phénomène naturel très simple : les individus les plus adaptés à survivre, dans un environnement donné, survivront plus longtemps et auront donc plus de chance de se reproduire. En pratique, les valeurs retournées par la *fitness function* permettent d'affecter les probabilités de reproduction aux différents individus. Enfin la sélection est effectuée par une méthode adaptée, généralement la méthode de la *roulette*.

- *Croisement* ou *crossover*. Le but du croisement est de combiner l'héritage génétique des deux parents. Cette étape constitue la première phase de la *reproduction* dont l'objectif est de fournir les nouveaux individus qui constitueront la génération suivante. L'opération de croisement s'effectue gène à gène, i.e. entre les gènes homologues des parents. Le point de coupe, aussi appelé *crossover point*, est déterminé aléatoirement pour chaque gène de chaque couple d'individus. Les gènes des parents sont ensuite combinés comme montré sur la fig. 3.7 où la phase de reproduction de deux individus de la population initiale du problème (3.35) est présentée.
- *Mutation*. La mutation peut être considérée comme un *mécanisme d'adaptation de second ordre* ayant pour objectif d'améliorer la capacité exploratoire de l'algorithme et éviter que les individus ne se retrouvent "coincés" dans un optimum local en début d'optimisation. D'un point de vue pratique, la mutation intervient après la phase de croisement et peut affecter un ou plusieurs gènes. La sélection du bit qui sera muté est aléatoire et la probabilité que ce bit soit muté est p_m. Dans ce cas, la valeur du bit est modifiée de manière à obtenir la valeur opposée à celle qu'il avait au départ (la valeur 0 lui est affectée si sa valeur de départ était 1 et *vice versa*).
- *Critère d'arrêt*. Il existe plusieurs critères pour les algorithmes génétiques. Les plus utilisés sont le nombre maximal de générations ou une condition de non-amélioration de la fonction objectif (i.e. l'algorithme est stoppé si la fonction objectif ne diminue pas pendant un certain nombre d'itérations). Notons que dans le premier cas, une règle empirique existe, fixant le nombre de générations de manière à ce que $N_{gen} \times N_{ind}$ soit au moins égal à 80000 et que, dans le second cas, il est nécessaire d'utiliser un opérateur optionnel des algorithmes génétiques, à savoir l'élitisme dont nous parlerons dans la section suivante. Il peut également être envisagé d'utiliser un critère d'arrêt basé sur la réussite de l'algorithme à atteindre un seuil de la valeur objectif. Par exemple, cela peut se traduire par un gain minimal attendu par rapport à une solution de référence.

La fig. 3.7 montre une phase de reproduction complète, i.e. croisement suivi de mutation, de deux parents de la population initiale du problème (3.35). Dans un premier temps, le point de croisement est choisi aléatoirement pour chaque gène indépendamment (le point de croisement du gène 1 est différent de celui du gène 2). Les codages binaires des parents sont ensuite mélangés au niveau du point de croisement pour chacun des gènes, permettant d'obtenir le génotype intermédiaire (i.e. avant mutation) des enfants. La mutation des gènes des enfants ainsi obtenus est réalisée avec la probabilité p_m après une sélection aléatoire du bit à modifier. Comme on peut le constater, tous les gènes des enfants ne sont pas forcement mutés. En effet, dans cet exemple, seuls le gène 1 de l'enfant 1 et le gène 2 de l'enfant 2 sont mutés. Après cette étape, le génotype,

et le phénotype associé, des enfants sont connus et le couple suivant de parents entre en phase de reproduction. L'opération est répétée jusqu'à que le nombre d'individu N_{ind} de la génération suivante soit atteint.

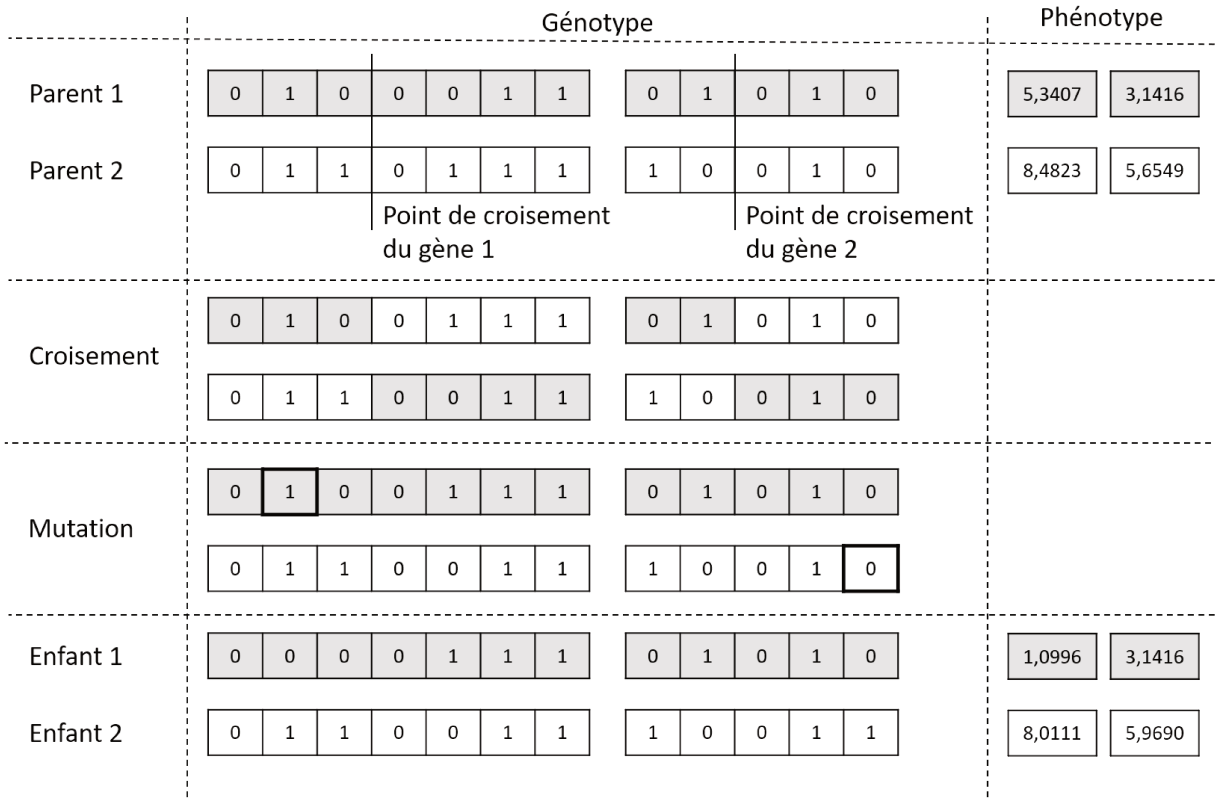


FIGURE 3.7 – Exemple d'une phase de reproduction (croisement + mutation) dans un algorithme génétique standard pour le problème (3.35)

La capacité intrinsèque des algorithmes génétiques à explorer le domaine d'optimisation les rend capables de traiter des problèmes non-convexes. De plus ils peuvent être considérés comme des méthodes d'*ordre zéro*, ne nécessitant pas l'évaluation de dérivées des fonctions objectif ou contraintes. Cependant, ils ne sont pas adaptés aux problèmes de prise de décision et, dans le cas de certains problèmes industriels complexes, le coût de calcul de la fonction objectif d'un individu peut être élevé et rendre impossible l'optimisation par ce type d'algorithme. L'une des solution est alors de fournir un métamodèle adapté diminuant grandement le temps de calcul de la fonction objectif.

Particularités de BIANCA

Le code original de BIANCA (*Biologically Inspired ANalysis of Composite Assemblages*) a été proposé par Vincenti [15] et, comme son nom l'indique, a été conçu dans l'optique de traiter

des problèmes d'optimisation de structures composites multi-couches. Cette première version de BIANCA (implémentée dans le langage FORTRAN) était basée sur les caractéristiques classiques des algorithmes génétiques décrites précedemment. La formulation de BIANCA a été modifiée et enrichie par Montemurro [14] pour être en mesure de traiter une nouvelle classe de problèmes d'optimisation : les problèmes d'optimisation de *systèmes modulaires*. Une stratégie très générale (i.e. indépendante du problème d'optimisation) permettant de traiter les contraintes a également été développée par Montemurro [14] et nommée *Automatic Dynamic Penalisation* (ADP).

Les problèmes industriels ont souvent pour objectif l'optimisation de *systèmes modulaires*. Ces systèmes sont composés d'*unités élémentaires* et *répétitives* (RUs pour *Repetitive Units*). Chaque RU représente un *module* qui est caractérisé par un ensemble de variables de conception de différentes natures, telles que des propriétés géométriques, dimensionnelles, matériaux, etc. Bien évidemment, tous les modules partagent une *structure commune*, i.e. chaque RU est caractérisée par un vecteur de variables de conception de même forme mais dont les valeurs ne sont pas nécessairement identiques. L'optimisation de systèmes/structures modulaires est très complexe car cela implique d'optimiser simultanément les modules composant le système et le nombre de ces modules. Un cas particulier d'optimisation de tels systèmes, beaucoup plus simple à traiter, est rencontré lorsque tous les modules le composant sont identiques (i.e. ils partagent le même vecteur de variables de conception). Cependant, dans le cas le plus général où les modules sont différents les uns des autres, la difficulté augmente significativement. Il n'existe, en général, pas de critère permettant de fixer *a priori* le nombre optimal de modules. C'est pourquoi le nombre de modules doit être inclus parmi les variables d'optimisation avec les valeurs des paramètres les constituant.

Montemurro [14] a montré que le problème d'optimisation lié à ce type de système est défini sur espace de dimension variable. Plus exactement, le problème est caractérisé par un *nombre variable* de variables de conception. Le problème d'optimisation d'un système modulaire peut être posé de la manière suivante :

$$\begin{gathered} \min_{\mathbf{x}} f(\mathbf{x}, n_c), \\ \text{sujet à :} \\ \begin{cases} g_i(\mathbf{x}, n_c) \leq 0, \ i = 1, ..., m_i(n_c), \\ h_j(\mathbf{x}, n_c) = 0, \ j = 1, ..., m_e(n_c), \\ \mathbf{x}_{LB} \leq \mathbf{x} \leq \mathbf{x}_{UB}, \\ (\mathbf{x}_{LB}, \mathbf{x}, \mathbf{x}_{UB}) \in \mathbb{R}^{n(n_c)}, \end{cases} \end{gathered} \tag{3.36}$$

où la dépendance du nombre de variables de conception et du nombre de contraintes par rapport au nombre de modules n_c est explicite. On peut remarquer que, le nombre de modules étant inclus parmi les variables d'optimisation, le concept classique de domaine de conception du problème (3.36) peut être généralisé et réinterprété comme un *multivers* d'espaces de conception sur lesquels l'algorithme évolue simultanément. Ce multivers est ainsi "peuplé" par des points représentant des systèmes modulaires composés de nombres différents de modules. Par conséquent, chaque espace de ce multivers à une dimension différente et le nombre d'inconnues du CNLPP (3.36) est différent pour deux points distincts.

Un exemple typique de problème d'optimisation de système modulaire est l'optimisation de structures réalisées à partir de composite stratifié dans lequel un module est représenté par un pli. Chaque pli est une RU, caractérisée par des variables de conception comme l'orientation des fibres, les propriétés des matériaux qui composent le pli ou encore son épaisseur. L'efficacité de l'algorithme pour ce genre de problèmes a été démontrée dans les références [16–18].

Dans cette version de BIANCA, la résolution de CNLPP du type de l'éq. (3.36) est rendu possible grâce, notamment, à une nouvelle formulation du génotype des individus. En effet, le nombre de chromosomes est variable d'un individu à l'autre et on considère les individus possédant le même nombre de chromosomes comme faisant partie de la même *espèce*. Nous allons maintenant présenter les principales fonctionnalités, et différence, de cet algorithme par rapport aux algorithmes génétiques classiques. Tout d'abord, notons qu'il est composé des mêmes opérateurs de base, modifiés pour s'adapter au nombre de chromosomes variable :

- *Adaptation.* L'adaptation est réalisée grâce à une fonction de la forme suivante :

$$fit(ind_i) = \left(\frac{f_{max}^{(g)} - f(ind_i)}{f_{max}^{(g)} - f_{min}^{(g)}} \right)^{f_{pres}} \tag{3.37}$$

  où $fit(ind_i)$ est la valeur retournée par la *fitness function* pour l'individu i, $f_{max}^{(g)}$ et $f_{min}^{(g)}$ sont respectivement les valeurs maximales et minimales de la fonction objectif pour la génération g en cours, $f(ind_i)$ est la valeur de la fonction objectif de l'individu i et f_{pres} est un paramètre nommée *fitness pressure* à fixer par l'utilisateur. Il existe d'autres choix possibles (cf. [14]).

Numéro de l'individu	Valeur de la *fitness function*	Pourcentage affecté de la roue
1	0.1	10%
2	0.1	10%
3	0.2	20%
4	0.6	60%
Total	1	100%

Tableau 3.2 – Valeur de la *fitness function* et pourcentage pour chacun des individus d'une population

- *Sélection.* Nous employons la méthode de la *roulette* dont le principe va être illustré par un exemple très simple. La sélection par la méthode de la *roulette* est un processus purement aléatoire dans lequel une plus grande probabilité est affectée aux individus ayant une plus grande valeur de *fitness function*. Considérons 4 individus dont les valeurs de la *fitness function* sont données dans le tableau 3.2. Chacun de ces individus reçoit une section de la roue égale au pourcentage de sa *fitness function* par rapport à la *fitness* totale. Dans le cas général d'une population composée de N_{ind} individus, ce pourcentage est donné par la relation suivante :

$$r_k = \frac{fit_k}{\sum_{i=1}^{N_{ind}} fit_i}, \tag{3.38}$$

où fit_k et r_k sont respectivement la valeur de la *fitness* et le pourcentage de la roue affecté au k^{e} individu. La roue correspondante à cet exemple est donnée à la fig. 3.8. Une fois les sections de la roue affectées, la sélection est faite simplement en faisant tourner la roue et en prenant l'individu sur lequel elle s'arrête. Dans notre exemple, l'individu 4 qui possède une section de 60% de la roue à beaucoup plus de chance d'être sélectionné que les autres individus. Notons que les algorithmes génétiques travaillent, en général, à nombre d'individus constant et, de ce fait, la roue doit être lancé N_{ind} fois. Notons également qu'un même individu peut être sélectionner plusieurs fois pour la phase de reproduction.

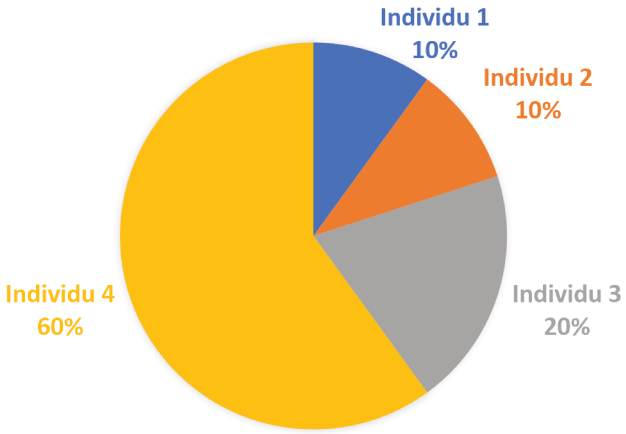


FIGURE 3.8 – Roue correspondante à l'exemple du tableau 3.2

- *Nouvelle représentation de l'individu et introduction du contexte d'espèce.* Ce qui différencie BIANCA des algorithmes génétiques classiques, c'est qu'il a été conçu dans l'optique de traiter des problèmes d'optimisation de systèmes modulaires. Ce genre de système est composé d'unités élémentaires (RU), ou modules, caractérisés par un certain nombre de variables. Prenons l'exemple d'une plaque multicouche réalisée avec des plis unidirectionnels. Dans ce cas, le système modulaire est la plaque, et les modules sont les différents plis composants la plaque. Chaque pli est caractérisé par deux paramètres, à savoir, son orientation et son épaisseur. Classiquement le problème d'optimisation est résolu en fixant le nombre de modules. Seules les valeurs des paramètres les composants sont alors optimisées. L'atout majeur de BIANCA réside dans sa capacité à optimiser à la fois le nombre de modules et les valeurs des paramètres qui les caractérisent. Cette spécificité a été rendue possible grâce à une modification du génotype des individus. Ainsi, chaque module composant un système modulaire est représenté par un chromosome tandis que les paramètres le caractérisant sont représentés par ses différents gènes. La fig. 3.9 présente la matrice permettant de stocker le génotype d'un individu k. Les lignes de cette matrice représentent les n^k différents chromosomes tandis que les colonnes représentent les m gènes. Chaque chromosome est composé par le même nombre de gènes. Enfin, le nombre de chromosomes n^k, indépendant pour chaque individu, est stocké dans la colonne $m+1$, après le dernier gène. Les valeurs e

correspondent aux chromosomes non utilisés pour l'individu k. Le nombre de lignes laissées non affectées est $n_{c,max} - n^k$, où $n_{c,max}$ est le nombre maximal de chromosomes. Le nombre minimal de chromosomes $n_{c,min}$ peut lui aussi être fixé. La fig. 3.10 fournit la structure d'un individu représentant la séquence d'empilement du composite stratifié composant la plaque à optimiser. Dans ce cas, le *chromo-masque* est composé d'un premier gène représentant l'orientation du pli considéré et d'un second gène représentant, lui, l'épaisseur du pli. Le nombre de plis est donné par le nombre de chromosomes de l'individu.

g^k_{11}	g^k_{12}	$\cdots$	g^k_{1m}	n^k
g^k_{21}	g^k_{22}	$\cdots$	g^k_{2m}	
$\vdots$	$\vdots$	$\ddots$	$\vdots$	
g^k_{n1}	g^k_{n2}	$\cdots$	g^k_{nm}	
e	e	e	e	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
e	e	e	e	

FIGURE 3.9 – Structure d'un individu dans BIANCA

- *Croisement* ou *crossover*. L'un des aspects important de BIANCA est sa capacité à faire reproduire des individus appartenant à différentes espèces sans les rendre stériles. En effet, dans la nature, le croisement entre espèces différentes résulte souvent en une espèce incapable de se reproduire, c'est le cas par exemple des *ligres* qui sont obtenus par le croisement entre un lion mâle et une tigresse ou des *tigrons* qui sont le résultat d'un croisement entre une lionne et un tigre mâle. D'autres croisements d'espèces non-stériles existent, c'est le cas des *mulets* (croisement d'un âne et d'une jument) ou des *bardots* (croisement d'une ânesse et d'un étalon). Notons cependant que leur caractéristiques génétiques sont à la base très similaires et que tous les croisements ne sont pas capables d'engendrer un enfant sain. Ce n'est pas le cas dans BIANCA, tous les croisements entre individus de différentes espèces engendrent des enfants sains (correspondant à un point de l'espace d'optimisation) et fertiles (capables d'engendrer à leurs tours des enfants pour la génération suivante). Pour ce faire, une étape de décalage, de probabilité p_{shift}, est ajoutée avant l'opération de croisement gène à gène. La fig. 3.11 présente un exemple de la phase de reproduction modifiée. Dans cet exemple, deux parents, dont les nombres de chromosomes sont respectivement $n_{c,1} = 3$ pour le parent 1 et $n_{c,2} = 5$ pour le parent 2, sont accouplés. L'étape de décalage intervient avant la phase de croisement et permet de choisir les gènes du parent 2 qui seront croisés avec ceux du parent 1 (car ici $n_{c1} < n_{c2}$). Le parent ayant le plus de chromosomes, ici le parent 2, croisera les gènes des indices compris entre $[1 + dec; n_{c,1} + dec]$, avec dec le décalage et respectant la condition $n_{c,1} + dec \leq n_{c,2}$. Le décalage est donc compris dans l'intervalle $[0; n_{c,2} - n_{c,1}]$ (si $n_{c,1} \leq n_{c,2}$). Dans la fig. 3.11, le décalage a une valeur $dec = 1$.

Le croisement gène à gène répond ensuite aux mêmes règles que les algorithmes génétiques standard (cf. fig. 3.7). Les enfants obtenus ont le même nombre de chromosomes que les parents, i.e. l'enfant 1 aura un nombre de chromosomes $n_{c,1} = n_{p,1}$ et l'enfant 2 aura un nombre de chromosomes $n_{c,2} = n_{p,2}$. Comme on peut le voir dans la fig. 3.11, les gènes du parent 2 qui ne sont pas croisés avec ceux du parent 1 sont directement transmis à l'enfant 2.

a_1^k	t_1^k	n^k
a_2^k	t_2^k	
⋮	⋮	
$a_{n^k}^k$	$t_{n^k}^k$	
e	e	
⋮	⋮	
e	e	

FIGURE 3.10 – Structure d'un individu représentant la séquence d'empilement d'un composite stratifié constitué de plis unidirectionnels

- *Mutation.* Le nombre de chromosomes étant variable, la phase de mutation a été modifiée et s'articule en deux étapes. Dans un premier temps, une mutation du nombre de chromosomes se produit avec une probabilité p_{mc} pour chaque individu. Cette mutation entraine l'ajout/suppression (aléatoirement sélectionné) d'un chromosome en une position aléatoirement déterminée. Dans la fig. 3.11, on peut remarquer que le nombre de chromosomes de l'enfant 1, $n_{c,1}$, est augmenté de 1 dans la phase de mutation du nombre de chromosomes et ce nouveau chromosome est introduit aléatoirement en position 2. Le nombre de chromosomes de l'enfant 2, $n_{c,2}$, est, quant à lui, diminué de 1 par la suppression du chromosome en position 3. Évidemment, si le nombre de chromosomes d'un individu est égal au nombre maximal de chromosomes, il n'est possible d'effectuer qu'une suppression de chromosome. A l'inverse, si le nombre de chromosomes est égal au nombre minimum de chromosomes, seul un ajout est possible. Dans un second temps, après une étape de réarrangement des gènes (cf. fig. 3.11), la mutation des gènes bit par bit des algorithmes génétiques standards intervient (cf. fig. 3.7). Dans l'exemple de la fig. 3.11, seul le gène du quatrième chromosome de l'enfant 1 est muté, tandis que l'enfant 2 ne subit pas de mutation de ses gènes. A la fin de cette phase de mutation, les enfants générés peuvent appartenir à des espèces différentes de celles de leurs parents (contrairement à la phase de croisement).

Cette nouvelle phase de reproduction (i.e. croisement + mutation) permet de faire évoluer simultanément les espèces et les individus. C'est pour cette raison que la recherche de solutions des problèmes d'optimisation de systèmes modulaires peut être effectuée de manière efficace par

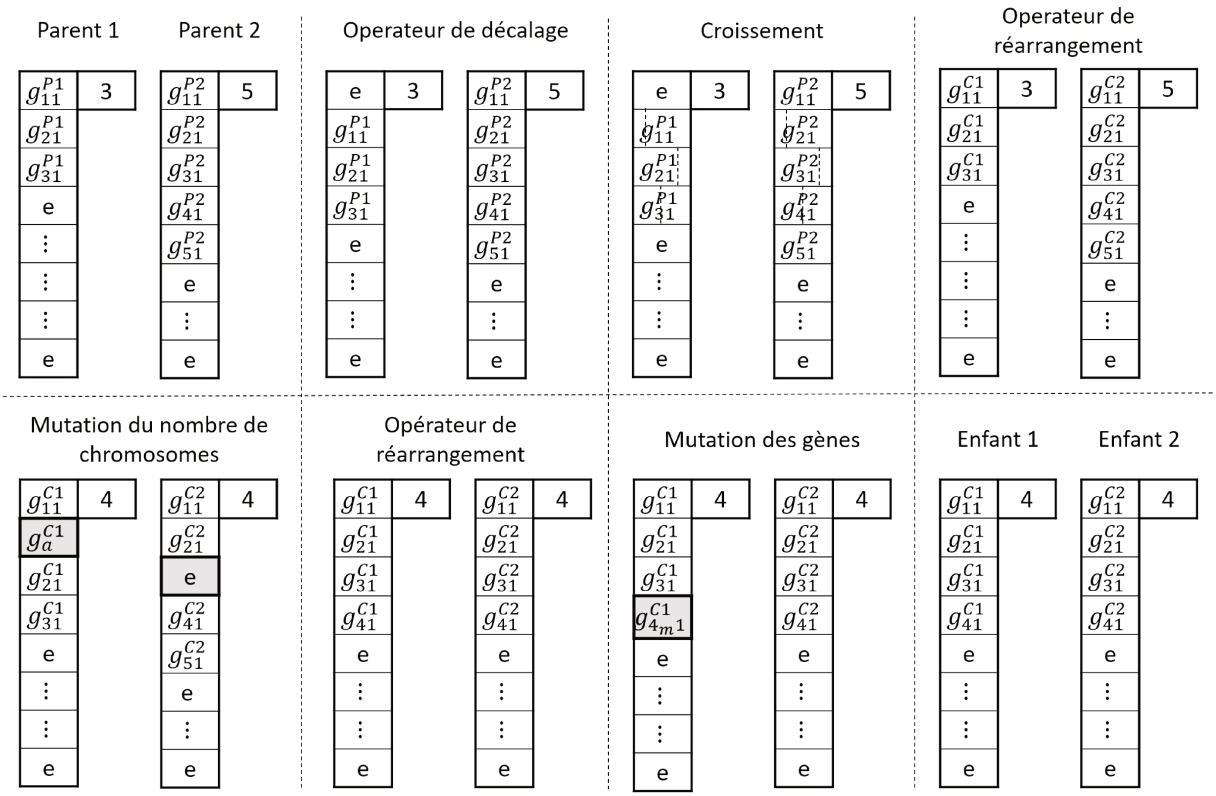


FIGURE 3.11 – Phase de reproduction modifiée entre deux individus appartenant à des espèces différentes

cet algorithme. Outre sa phase de reproduction spécifique, BIANCA possède également d'autres opérateurs et il est capable de faire évoluer plusieurs populations en même temps. Nous détaillons ci-après les différents opérateurs disponibles dans BIANCA :

- *Elitisme.* La phase de croisement des algorithmes génétiques se traduit, comme c'est le cas dans les phases de reproduction d'êtres vivants, par la disparition systématique des parents dans la génération suivante. Cela implique que le meilleur individu (ayant la meilleure valeur de la fonction objectif) peut être perdu d'une génération à l'autre. L'opérateur d'élitisme permet de conserver cet individu en remplaçant le pire individu de la génération actuelle par le meilleur de la précédente. Cette opérateur n'intervient donc qu'à partir de la seconde génération de l'algorithme.
- *Migration.* Le fait de faire évoluer plusieurs populations en même temps permet, en théorie, d'explorer plusieurs zones de l'espace simultanément. Cependant, il peut parfois être utile de croiser ces populations et c'est l'objectif de la migration. Toutes les n_{mig} générations, n_{mig} étant un paramètre fixé par l'utilisateur, le meilleur individu de la population 1 est copié dans la population 2 à la place du pire individu de celle-ci. L'opération est répétée

jusqu'à ce que le meilleur individu de la population $N_{pop} - 1$ soit copié dans la population N_{pop}, N_{pop} étant le nombre de populations évoluant simultanément. Le meilleur individu de la population N_{pop} est lui envoyé dans la population 1.

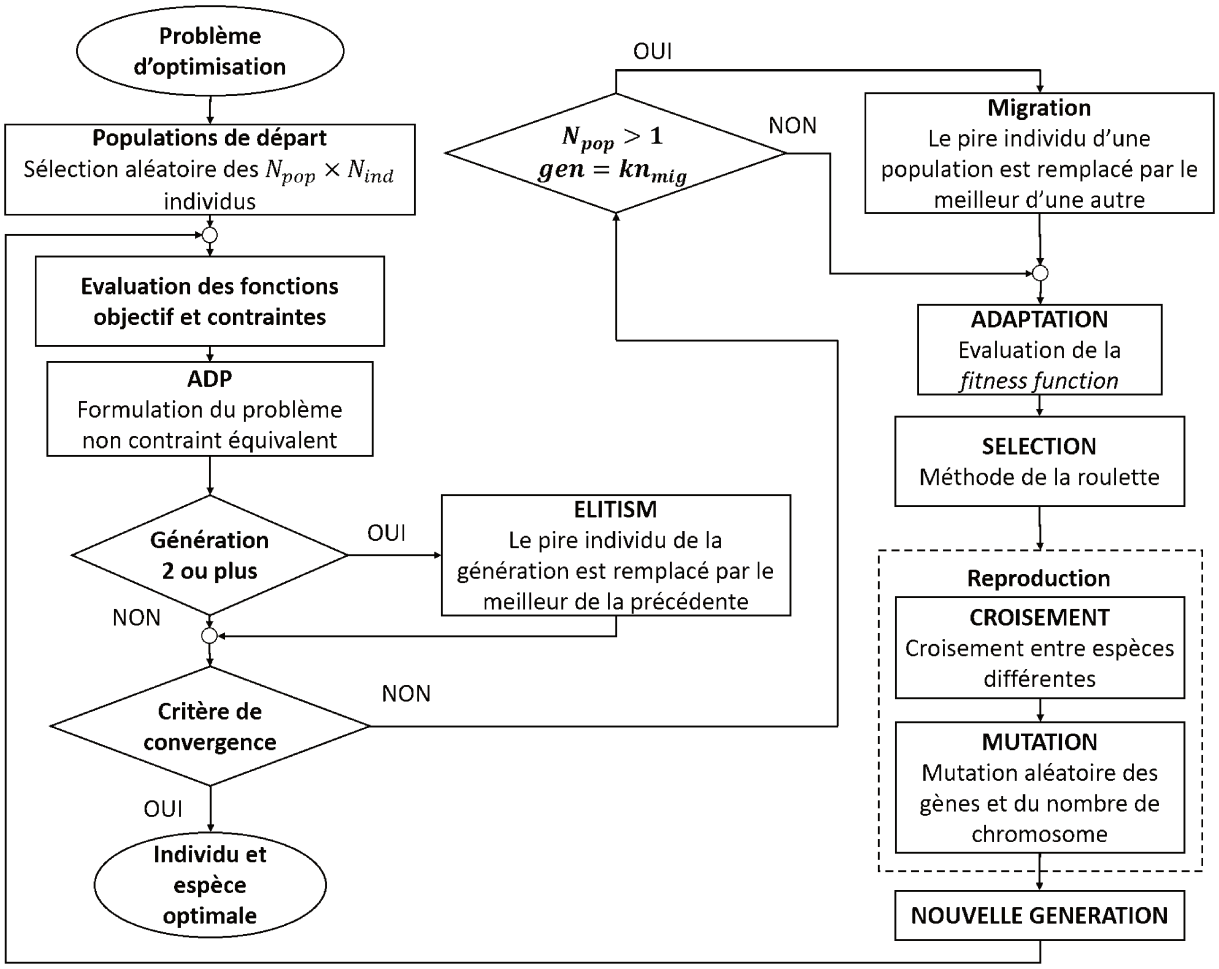


FIGURE 3.12 – Algorithme génétique BIANCA

La capacité de BIANCA à traiter des problèmes de dimension variable est déjà un avantage majeur mais ce n'est pas le seul. En effet, comme nous l'avons dit précédemment, les CNLPPs sont généralement traités en formulant un problème sans contraintes équivalent. La génération de ce problème équivalent est loin d'être une tâche triviale. Dans BIANCA, le problème équivalent est généré par une stratégie très efficace nommée ADP pour *Automatic Dynamic Penalisation* [14, 222]. Son nom expose bien les principaux avantages de cette stratégie. En effet, la pénalisation appliquée à la fonction objectif s'adapte automatiquement au problème traité et est donc indépendante de la formulation de celui-ci. Elle est de plus dynamique car elle s'adapte au fil des générations. Le problème d'optimisation sans contraintes équivalent qui est traité par

BIANCA grâce à l'ADP est de la forme suivante :

$$\min_{\mathbf{x}} f_p(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{m_i} c_i G_i(\mathbf{x}) + \sum_{j=1}^{m_e} q_j H_j(\mathbf{x}), \quad i = 1, ..., m_i, \quad j = 1, ..., m_e, \tag{3.39}$$

où $f_p(\mathbf{x})$ est la fonction objectif pénalisée, $G_i(\mathbf{x})$ est donné par

$$G_i(\mathbf{x}) = \max\left[0, g_i(\mathbf{x})\right], \quad i = 1, ..., m_i, \tag{3.40}$$

tandis que $H_j(\mathbf{x})$ est donné par

$$H_j(\mathbf{x}) = \max\left[0, |h_j(\mathbf{x})| - \epsilon\right], \quad \epsilon \ll 1, \quad j = 1, ..., m_e. \tag{3.41}$$

La définition de $G_i(\mathbf{x})$ et de $H_j(\mathbf{x})$ permet de remarquer que, dans le cas où toutes les contraintes sont respectées (égalités et inégalités), la fonction objectif pénalisée est égale à la fonction objectif non-pénalisée :

$$\text{Si } \forall i = 1, ..., m_i,\ g_i(\mathbf{x}) \leq 0 \text{ et } \forall j = 1, ..., m_e,\ h_j(\mathbf{x}) = 0, \text{ alors } f_p(\mathbf{x}) = f(\mathbf{x}). \tag{3.42}$$

La grande force de l'ADP est de déterminer les coefficients c_i et q_j de manière automatique (sans intervention de l'utilisateur) et adaptative (répété à chaque génération), dans le sens où la pénalisation n'est, ni trop faible, ni trop forte, assurant ainsi un bon compromis entre le fait d'éviter une solution non faisable et une exploration efficace des bords du domaine faisable. A chaque génération, les coefficients sont donnés par les relations suivantes :

$$\begin{cases} c_i = \dfrac{|f^F_{best} - f^{NF}_{best}|}{(G_i)^{NF}_{best}}, & i = 1, ..., m_i, \\ q_j = \dfrac{|f^F_{best} - f^{NF}_{best}|}{(H_j)^{NF}_{best}}, & j = 1, ..., m_e, \end{cases} \tag{3.43}$$

où f^F_{best} et f^{NF}_{best} sont les valeurs de la fonction objectif des meilleurs individus du domaine faisable et du domaine non faisable respectivement. Considérons maintenant le meilleur individu du domaine non faisable, $(G_i)^{NF}_{best}$ et $(H_j)^{NF}_{best}$ représentent alors les quantités de violation des contraintes d'inégalité et d'égalité, respectivement. L'idée est de ne pas trop pénaliser des solutions très performantes, bien qu'elles ne respectent pas les contraintes et qu'elles soient donc non faisables, afin d'en faire des "points d'attraction" pour la solution. L'exploration des bords du domaine faisable est ainsi grandement améliorée. Dans le cas où tous les individus d'une population seraient non faisables, une stratégie adaptée est mise en place. Nous ne la détaillons pas ici mais le lecteur intéressé est renvoyé vers les références [14, 222].

Les différents opérateurs de BIANCA ayant été définis, la fig. 3.12 présente l'algorithme sous forme schématique. Dans cette version, même si l'algorithme est capable de traiter des problèmes modulaires de dimension variable, comme la définition de l'empilement optimal d'un composite stratifié dont le nombre de pli varie, il n'est pas possible de traiter des problèmes présentant plusieurs types de modules différents, et indépendants les uns des autres. Nous allons expliquer dans la prochaine section comment cette difficulté a été surmontée, grâce notamment à un codage différent du génotype.

L'algorithme ERASMUS

BIANCA a d'abord été développé dans le langage FORTRAN afin de favoriser la vitesse de calcul. Malgré la rigidité du formalisme de ce langage, le traitement des problèmes d'optimisation relatifs aux systèmes modulaires a été rendu possible par un stockage approprié de l'information génétique dans des matrices multidimensionnelles. Deux dimensions de cette matrice permettent de définir la population et l'individu. De cette manière, à population et individu fixés, le stockage de l'information génétique pour cet individu peut se formaliser par une matrice $2D$ dont les lignes et les colonnes sont respectivement les chromosomes et les gènes de l'individu comme le montre la fig. 3.9.

Le formalisme de la fig. 3.9 présente cependant une limitation. En effet, même si BIANCA peut traiter des problèmes d'optimisation de systèmes modulaires, ceux-ci ne peuvent être caractérisés que par un seul type de module. Un système modulaire est caractérisé par différents types de modularité lorsque les modules qui le composent peuvent être regroupés en différents ensembles caractérisés par le même vecteur d'inconnues (i.e. les modules d'un ensemble ont le même vecteur de variables de conception). Un exemple de système modulaire possédant différents types de modules est donné à la fig. 3.13. Il s'agit d'une section du fuselage d'un avion dans lequel on trouve deux familles de modules : les lisses et les cadres. Bien évidemment, le module de chaque famille est caractérisé par son propre vecteur de variables de conception.

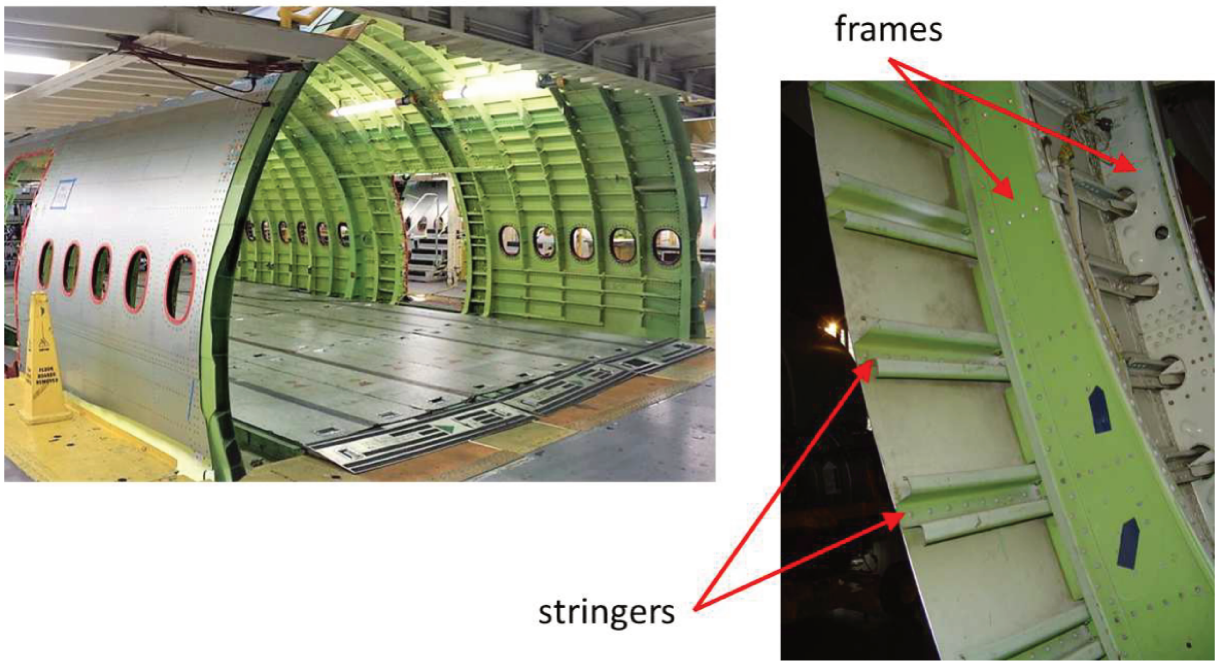


FIGURE 3.13 – Exemple de structure modulaire (section d'un fuselage) avec deux types de modules : les lisses (*stringers*) et les cadres (*frames*)

Afin de répondre à cette limitation, BIANCA a été traduit dans le langage MATLAB et la

structure du génotype des individus a été généralisée afin de traiter des problèmes d'optimisation de systèmes modulaires possédant plusieurs types de modules. La formulation mathématique du problème d'optimisation de tels systèmes peut être mis sous la forme suivante :

$$\begin{gathered}\min_{\mathbf{x}} f(\mathbf{x}, n_c^{(1)}, \ldots, n_c^{(N_{mod})}), \\ \text{sujet à :} \\ \begin{cases} g_i(\mathbf{x}, n_c^{(1)}, \ldots, n_c^{(N_{mod})}) \leq 0, \quad i = 1, ..., m_i(n_c^{(1)}, \ldots, n_c^{(N_{mod})}), \\ h_j(\mathbf{x}, n_c^{(1)}, \ldots, n_c^{(N_{mod})}) = 0, \quad j = 1, ..., m_e(n_c^{(1)}, \ldots, n_c^{(N_{mod})}), \\ \mathbf{x}_{LB} \leq \mathbf{x} \leq \mathbf{x}_{UB}, \\ (\mathbf{x}_{LB}, \mathbf{x}, \mathbf{x}_{UB}) \in \mathbb{R}^{N_v}, \quad N_v = N_v(n_c^{(1)}, \ldots, n_c^{(N_{mod})}), \end{cases} \end{gathered} \tag{3.44}$$

où N_{mod} est le nombre de types de modules différents et où la dépendance du nombre de variables de conception et de contraintes en fonction des nombres de modules $n_c^{(1)}, \ldots, n_c^{(N_{mod})}$, est explicite.

Le résultat de cette opération de généralisation du génotype des individus, ainsi que de l'architecture de l'algorithme, est une nouvelle métaheuristique nommée ERASMUS (*EvolutionaRy Algorithm for optimiSation of ModUlar Systems*) [212]. La nouvelle structure de l'individu, illustrée dans la fig. 3.14, est définie par une section standard et N_{mod} sections modulaires ayant chacune des caractéristiques indépendantes les unes des autres. Chaque individu k (i.e. point de l'espace de conception) est ainsi caractérisé par un génome composé de $N_{mod} + 1$ *sections* ayant une hiérarchie précise. La première section, ou *section standard*, est composée des paramètres non modulaires du problème et est divisée en deux parties. La première partie est composée d'un nombre fixe $n_{c,k}^{stand}$ de chromosomes, contenant chacun n_g^{stand} gènes (i.e. toutes les variables du problème dont le nombre ne varie pas). La seconde partie est composée d'un seul chromosome possédant N_{mod} gènes qui contiennent les nombres de chromosomes $n_{c,k}^{(1)}, \ldots, n_{c,k}^{(N_{mod})}$, des différentes section modulaires de l'individu k. Les N_{mod} sections restantes sont les sections modulaires de l'individu k (i.e. contenant les variables modulaires des différents modules du système à optimiser). Chacune d'entre elles admet un nombre de chromosomes maximal $n_{c,max}^{(mod)}$, un nombre de chromosomes minimal $n_{c,min}^{(mod)}$ et un nombre de gènes $n_g^{(mod)}$ indépendants des autres modules. Bien évidement, la phase de croisement s'effectue entre les gènes des modules homologues, i.e. le module i d'un individu k_1 est croisé avec le module i d'un individu k_2 et seulement avec celui-ci.

Notons que des problèmes beaucoup plus simples que le CNLPP (3.44) peuvent également être traités par ERASMUS. Par exemple, dans le cas où le nombre de paramètres est constant, seul la section *standard* de l'individu sera utilisée. Si en revanche, il n'existe pas de paramètres non modulaires, alors seules les nombres de chromosomes des modules de la section standard et les sections *modulaires* des individus seront utilisés. ERASMUS est donc un outil très puissant capable de s'adapter à la plupart des problèmes d'optimisation. Il peut également s'interfacer avec d'autres logiciels ou langages de programmation. En effet, il est par exemple possible d'utiliser des routines écrites en C++ ou FORTRAN, moyennant quelques adaptations mineures, afin de les utiliser comme si elles étaient des fonctions classiques de MATLAB. Il est également possible de faire appel à des logiciels de calcul par éléments finis commerciaux, tel qu'ANSYS par exemple,

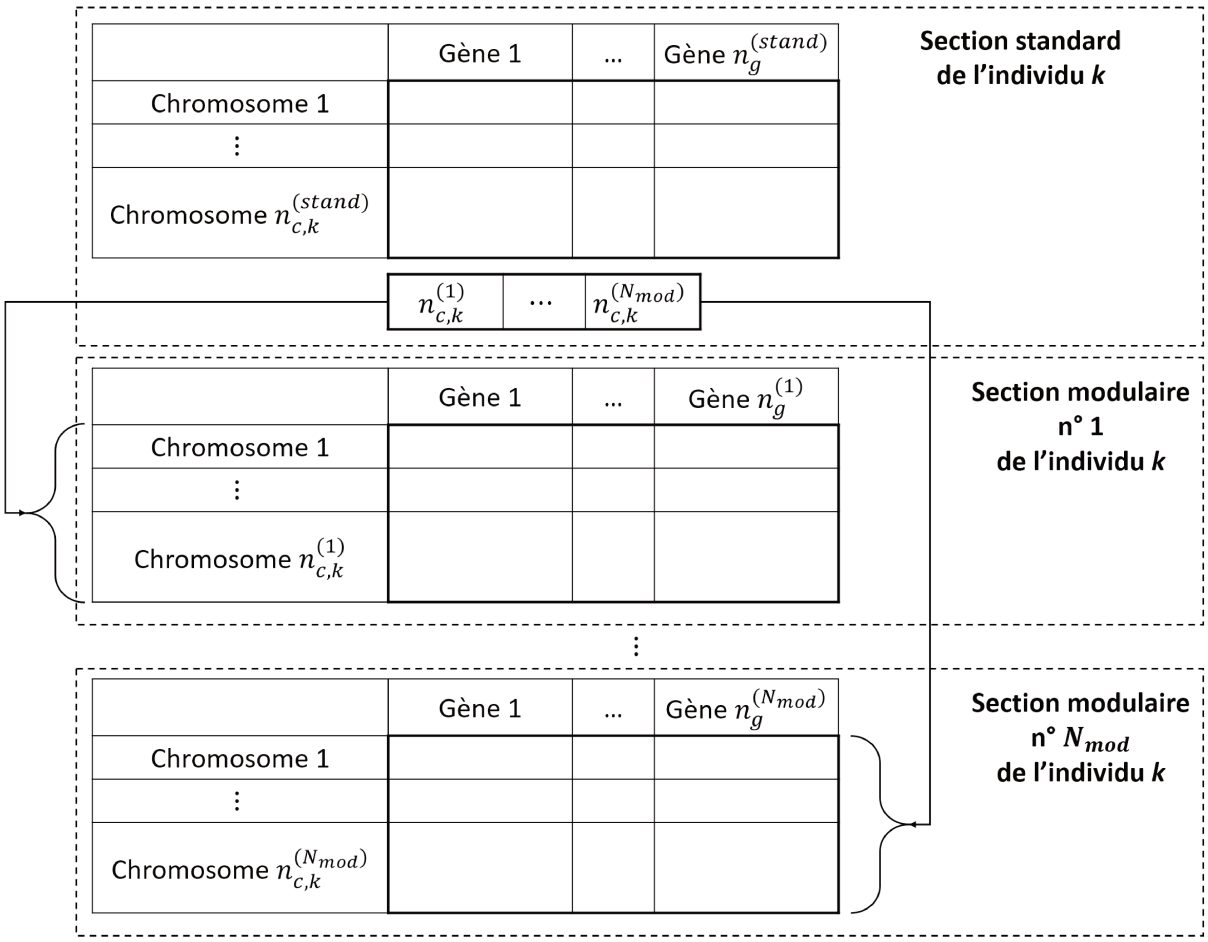


FIGURE 3.14 – Nouvelle structure d'un individu dans ERASMUS

de manière très ergonomique. Cependant, même si sa structure lui permet de s'adapter à un vaste panel de problèmes, cela n'assure en rien la rapidité de convergence et cet algorithme, comme toutes les métaheuristiques, reste limité par la taille du domaine à explorer et le temps de calcul nécessaire à l'obtention des valeurs des fonctions objectif et contraintes des différents individus.

C'est pourquoi nous allons maintenant parler des méthodes déterministes qui sont encore massivement utilisées sur des problèmes d'optimisation pourtant non-convexes. Elles présentent une réponse à la problématique soulevée par la limitation précédente, ces méthodes n'utilisant les informations que d'un seul point au lieu d'une population. Il serait cependant naïf de croire qu'il s'agit d'une réponse appropriée. En effet, ne se focalisant que sur un seul point, la réponse d'une méthode déterministe est entièrement dépendante du point de départ choisi. Afin d'atteindre un minimum global il est donc nécessaire d'avoir une connaissance profonde des phénomènes qui régissent le problème, permettant ainsi de choisir un point de départ approprié. En pratique, il est très rare que ce soit le cas et deux solutions peuvent être envisagées :

- Choisir des points de départ aléatoirement, dont le nombre est à définir, afin d'utiliser la méthode déterministe à partir de ces points et ainsi espérer trouver dans les réponses un minimum global,
- Déterminer un point de départ à partir d'une métaheuristique.

La seconde option est au cœur de la méthode que nous avons développée et c'est pourquoi, après une brève introduction sur les méthodes déterministes, nous présentons la méthode utilisée pour notre approche décrite au chapitre 4.

3.3.3 Méthodes déterministes de résolution de problèmes d'optimisation

Généralités sur les méthodes au gradient

Les méthodes déterministes d'optimisation sont basées sur un schéma itératif, mettant à jour les variables d'optimisation $\mathbf{x}_{k+1}$, à l'itération $k+1$, à partir des informations disponibles à l'itération k. C'est un domaine d'étude bien ancré comme en attestent les références [223, 224] et on les qualifie également de *méthodes basées sur le gradient*. Cela est dû au fait que l'information permettant de mettre à jour les variables d'optimisation d'une itération à l'autre est essentiellement basée sur l'opérateur de gradient.

Comme nous l'avons vu précédemment pour les métaheuristiques, un problème d'optimisation avec contraintes est généralement résolu en formulant un problème d'optimisation sans contraintes équivalent. C'est pourquoi nous allons d'abord introduire les méthodes de résolution des problèmes sans contraintes. Il existe principalement deux méthodes permettant de mettre à jour les paramètres d'optimisation d'une itération à l'autre [213] :

- Méthodes *line search*. Dans un premier temps, une direction de descente $\mathbf{p}_k$ appropriée, permettant de réduire la fonction objectif f, est déterminée. Ensuite un pas d'avancement α dans cette direction est calculé en résolvant le problème mono-dimensionnel suivant :

$$\min_{\alpha>0} f(\mathbf{x}_k + \alpha \mathbf{p}_k). \tag{3.45}$$

  Les algorithmes existants diffèrent essentiellement dans le choix de la direction de descente. La direction permettant la descente la plus raide est :

$$\mathbf{p}_k = -\nabla f(\mathbf{x}_k), \tag{3.46}$$

  car cette direction maximise la réduction de la fonction objectif à chaque itération. Cependant, et même si seules les informations sur le gradient de la fonction sont nécessaires, ce choix conduit souvent à un algorithme très lent pour des problèmes complexes.
  Bien évidemment, ce n'est pas la seule direction possible et n'importe quelle direction faisant un angle strictement inférieur à $\frac{\pi}{2}$ avec $-\nabla f(\mathbf{x}_k)$ permet d'atteindre le minimiseur de la fonction. Parmi les autres choix possibles, la direction de Newton, particulièrement efficace pour des fonctions objectifs proches de leurs approximations quadratiques localement autour de $\mathbf{x}_k$, s'appuie également sur les informations des dérivées secondes de la fonction objectif pour donner la direction de descente. Cette direction est donnée par :

$$\mathbf{p}_k = -\mathbf{H}_k^{-1} \nabla f(\mathbf{x}_k), \tag{3.47}$$

où $\mathbf{H}_k = \nabla^2 f(\mathbf{x}_k)$ est la matrice hessienne de la fonction objectif à l'itération k. Ce choix de direction de descente assure un taux de convergence quadratique mais la matrice hessienne doit être calculée pour chaque itération.

Une alternative à la direction de Newton est la direction quasi-Newton. Dans cette approche, la matrice hessienne n'est pas calculée explicitement et une approximation adaptée est utilisée :

$$\mathbf{p}_k = -\mathbf{B}_k^{-1} \nabla f(\mathbf{x}_k), \tag{3.48}$$

où $\mathbf{B}_k$ est l'approximation de la matrice hessienne, généralement donnée par la formule BFGS (Broyden-Fletcher-Goldfarb-Shanno). Le taux de convergence n'est pas quadratique dans ce cas mais super-linéaire.

La dernière direction fréquemment utilisée découle de l'utilisation du *gradient conjugué*. L'idée est de déterminer la direction $\mathbf{p}_k$ à l'itération actuelle en fonction de la direction de l'étape précédente $\mathbf{p}_{k-1}$ et en exploitant l'idée de la direction conjuguée. Les directions sont dites conjuguées si :

$$\mathbf{p}_i^T \mathbf{A} \mathbf{p}_j = 0, \quad \forall i \neq j, \tag{3.49}$$

où $\{\mathbf{p}_1, ..., \mathbf{p}_l\}$ est un ensemble de directions et $\mathbf{A}$ une matrice symétrique définie positive. Dans ce cas, on peut démontrer que la méthode des gradients conjugués permet de converger en n itération au plus, n étant la dimension du vecteur des variables d'optimisation. Le taux de convergence n'est pas aussi élevé que pour la direction de Newton ou de quasi-Newton mais cette méthode ne nécessite pas de stocker des matrices de grandes tailles.

- Méthodes *trust region*. Contrairement à la méthode précédente, dans celle-ci c'est d'abord le pas d'avancement qui est choisi et ensuite une direction de descente est calculée. La direction étant inconnue initialement, on parle, en début d'itération, de rayon de région de confiance (ou de rayon de boule) Δ_k plutôt que de longueur de pas d'avancement. L'idée est de résoudre une approximation quadratique du problème original, dans la région de confiance, pour donner l'itération suivante. Concernant la direction de descente, les mêmes choix que pour la méthode *line search* sont disponibles, à l'exception du gradient conjugué qui n'a pas d'équivalent dans le cadre de cette méthode. Il est important de noter que les algorithmes basés sur une méthode de *trust region* sont très sensibles aux modèles de mauvaises échelles, que l'on trouve dans la littérature sous la dénomination *poorly scaled*. Ce genre de modèles peut admettre une variation très importante de la valeur de la fonction objectif (de plusieurs ordre de grandeur parfois) par rapport aux variables d'optimisation. Cette limitation les rend très souvent bien moins robustes que les algorithmes utilisant les méthodes *line search* intrinsèquement non sensibles aux modèles *poorly scaled*.

Bien qu'un problème dépourvu de contraintes présente rarement un intérêt dans des applications concrètes du monde industriel, les méthodes de résolution associées, après quelques améliorations et modifications, forment la base de résolution des CNLPPs, souvent résolus par la formulation d'un problème sans contraintes équivalent [213]. Cette formulation équivalente fait appel au lagrangien du problème de minimisation de la forme de l'eq. (3.27) dont la fonction L est de la forme suivante :

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T g(\mathbf{x}) + \boldsymbol{\mu}^T h(\mathbf{x}), \tag{3.50}$$

où $\boldsymbol{\lambda}$ et $\boldsymbol{\mu}$ sont respectivement les vecteurs des multiplicateurs de Lagrange pour les contraintes d'inégalité ($\lambda_i \geq 0,\ \forall i = 1, ..., m_i$) et les contraintes d'égalité ($\mu_j \geq 0,\ \forall j = 1, ..., m_e$). Nous introduisons maintenant deux théorèmes à la base des méthodes de résolution des problèmes de type CNLPP. Pour la démonstration de ces théorèmes, le lecteur est renvoyé vers les références [213, 223, 224].

Théorème 3.1 (Conditions nécessaires de premier ordre)
Supposons les propriétés suivantes vérifiées :

1. $\mathbf{x}^*$ *est une solution locale du problème (3.27).*
2. *les fonctions* f, g_i *et* h_j *sont continuellement différentiables.*
3. *les conditions LICQ (*Linear Independence Constraint Qualification*) sont vérifiées au point* $\mathbf{x}^*$*, i.e. les gradients de chacune des fonctions contraintes d'égalité et d'inégalité sont linéairement indépendantes.*

alors, il existe deux vecteurs de multiplicateurs de Lagrange $\boldsymbol{\lambda}^* \geq 0$ *et* $\boldsymbol{\mu}^* \geq 0$ *et les conditions suivantes, connues sous le nom de conditions de Karush-Kuhn-Tucker (KKT), sont vérifiées :*

$$\begin{cases} \nabla_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = 0, \\ \lambda_i^* g_i(\mathbf{x}^*) = 0, \quad \forall i = 1, ..., m_i, \\ h_j(\mathbf{x}^*) = 0, \quad \forall j = 1, ..., m_e. \end{cases} \tag{3.51}$$

Dans l'éq. (3.51), $\nabla_{\mathbf{x}}$ représente l'opérateur gradient par rapport à la variable x. Le point $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ est appelé *point KKT*.

Théorème 3.2 (Conditions suffisantes de second ordre)
Étant données les fonctions f, g_i *et* h_j *deux fois continuellement différentiables. Supposons que le point* $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ *soit un point KKT et que la matrice hessienne du lagrangien* $\nabla^2 L(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ *soit définie positive. Alors,* $\mathbf{x}^*$ *est une solution locale stricte du problème (3.27).*

Ces deux théorèmes sont à la base des méthodes que nous présentons dans la prochaine section.

Algorithmes de résolution des CNLPPs

Ce manuscrit n'a pas vocation à dresser une liste exhaustive des différents algorithmes existants aussi, comme nous l'avons expliqué précédemment, le choix du langage MATLAB pour la programmation de notre algorithme génétique nous a conduit à considérer les méthodes d'optimisation au gradient de l'*optimization toolbox* [214] de ce logiciel. Les méthodes présentées ici sont donc les méthodes qu'il est possible d'utiliser dans la fonction *fmincon* du logiciel. Cependant une méthode a été écartée, malgré son efficacité, car elle présente des limitations trop importantes. En effet, l'algorithme *trust-region* peut être utilisé seulement quand le gradient de la fonction objectif est connu de manière analytique et, soit avec des variables bornées, soit avec des contraintes linéaires d'égalité, mais pas les deux. Cette limitation ne permettant pas d'utiliser la méthode dans la résolution du problème de minimisation qui nous intéresse, nous ne présentons ici que les autres algorithmes disponibles, à savoir :

- *Sequential Quadratic Programming* (SQP).
- *Active Set* (AS).
- *Interior Point* (IP).

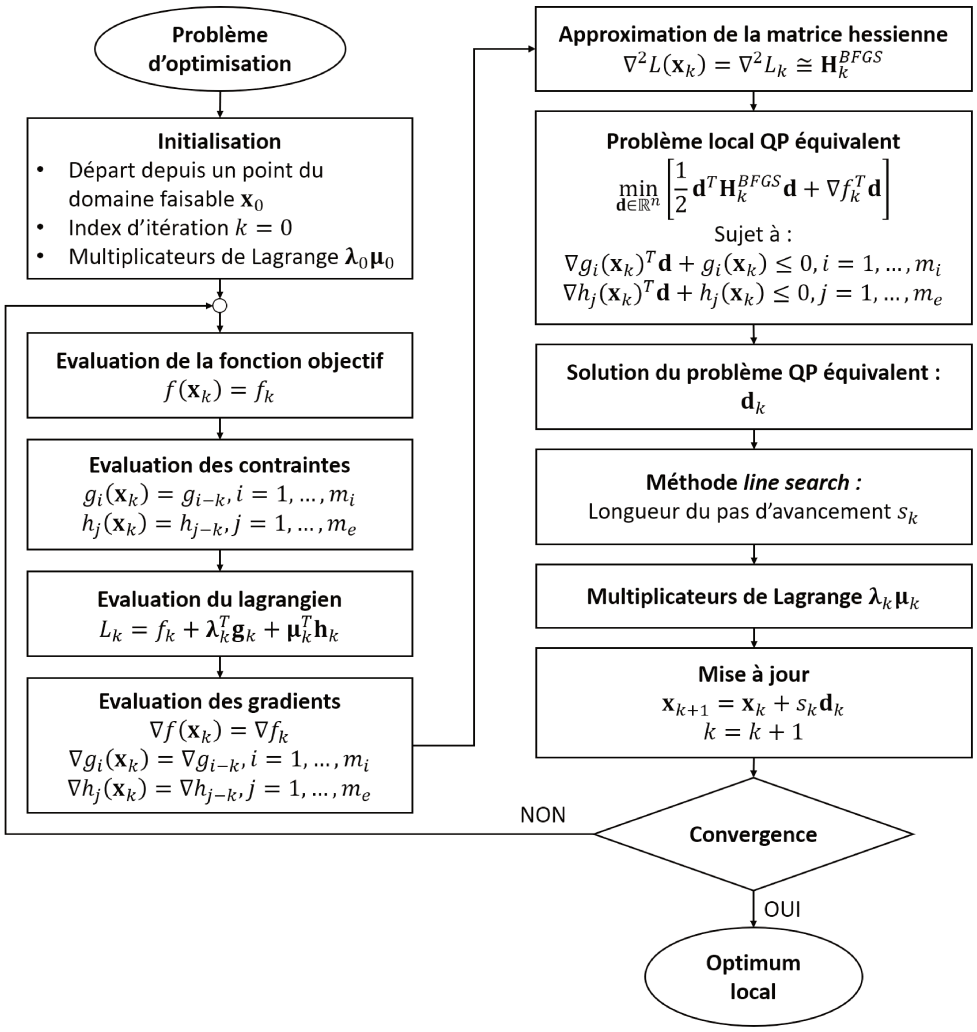


FIGURE 3.15 – Algorithme SQP (*Sequential Quadratic Programming*)

Algorithmes SQP et AS Les algorithmes SQP et AS peuvent être traités ensembles car ils reposent sur les mêmes fondements. En fait, la méthode AS est un cas particulier de la méthode SQP dans laquelle les contraintes sont traitées de manière plus efficace. Les premières étapes étant similaires, le descriptif qui suit est valable pour les deux approches. L'idée principale derrière ces méthodes est d'approximer le CNLPP initial par une séquence de problèmes de programmation

quadratiques QP (*quadratic programming*). Les conditions définissant un problème QP sont les suivantes :

- la fonction objectif f est une fonction quadratique des variables d'optimisation.
- Les fonctions contraintes d'égalité h_j et d'inégalité g_i sont des fonctions linéaires des variables d'optimisation.

Une solution peut toujours être fournie en utilisant une technique QP. Le principal désavantage de cette méthode est que le coût de calcul dépend de la fonction objectif et du nombre de contraintes. La vue d'ensemble de l'algorithme est fournie à la fig. 3.15.

Après que le problème ait été mis sous la forme du problème (3.27), il faut fournir en entrée de l'algorithme un point $\mathbf{x}_0$ "adapté". Nous l'avons vu précédemment, le choix de ce point initial influence la solution renvoyée par l'algorithme si le problème est non-convexe. Après initialisation des multiplicateurs de Lagrange, le lagrangien du problème est évalué à partir de l'éq. (3.50). L'étape suivante nécessite l'évaluation des fonctions objectif et contraintes qui peut se faire de manière analytique ou par évaluation numérique (approximation par différence finie des dérivées par exemple). Suite à cela, la matrice hessienne du lagrangien est approximée et utilisée avec le gradient précédemment calculé pour former le problème quadratique équivalent :

$$\begin{aligned} &\min_{\mathbf{d}} Q_k(\mathbf{d}) = \min_{\mathbf{d}} \tfrac{1}{2}\mathbf{d}^T \mathbf{H}_k^{BFGS} \mathbf{d} + \nabla f_k^T \mathbf{d}, \\ &\text{sujet à :} \\ &\bar{\mathbf{A}}_k \mathbf{d} \leq \mathbf{b}_k. \end{aligned} \tag{3.52}$$

Dans l'éq. (3.52), la matrice hessienne $\mathbf{H}_k^{BFGS}$ est approximée par la formule BFGS (Broyden-Fletcher-Goldfarb-Shanno) [213]. Le vecteur $\mathbf{d}$ constitue les variables d'optimisation du problème équivalent et peut être interprété comme la direction de recherche pour la k^{e} itération de l'algorithme SQP. Comme nous l'avons vu précédemment, le problème (3.52) étant un problème QP, les contraintes doivent être linéaires. Si ce n'est pas le cas, les contraintes du problème original (3.27) sont linéarisées et leurs gradients sont stockés dans la matrice $\bar{\mathbf{A}}_k$. La principale différence entre les méthodes SQP et AS réside dans la forme de cette matrice. En particulier, dans le cas de l'algorithme AS, seules les contraintes d'inégalité n'étant pas respectées apportent une contribution à la matrice $\bar{\mathbf{A}}_k$ (les contraintes d'égalité étant toujours incluses). De manière analogue, les coefficients de l'approximation de premier ordre des contraintes sont rassemblés dans le vecteur $\mathbf{b}_k$. Le problème équivalent est ensuite résolu par les méthodes courantes [213]. Dans le cas de la méthode AS, quelques contrôles internes peuvent être nécessaires, notamment pour vérifier si les contraintes ont bien été évaluées et, éventuellement, mettre à jour la matrice $\bar{\mathbf{A}}_k$. La solution $\mathbf{d}_k$ est obtenue par une méthode de type *line search* et le pas d'avancement dans cette direction $\mathbf{s}_k$ peut alors être calculé. Enfin, les multiplicateurs de Lagrange et les variables d'optimisation $\mathbf{x}_{k+1}$ sont mis à jour. Cette présentation des algorithmes SQP et AS est relativement succincte et le lecteur intéressé par ces méthodes est renvoyé vers les références [213,214]. Nous ajouterons seulement que les algorithmes AS, contrairement aux SQP, peuvent tolérer que quelques itérations soient en dehors du domaine faisable. Cela permet une meilleure exploration des limites du domaine.

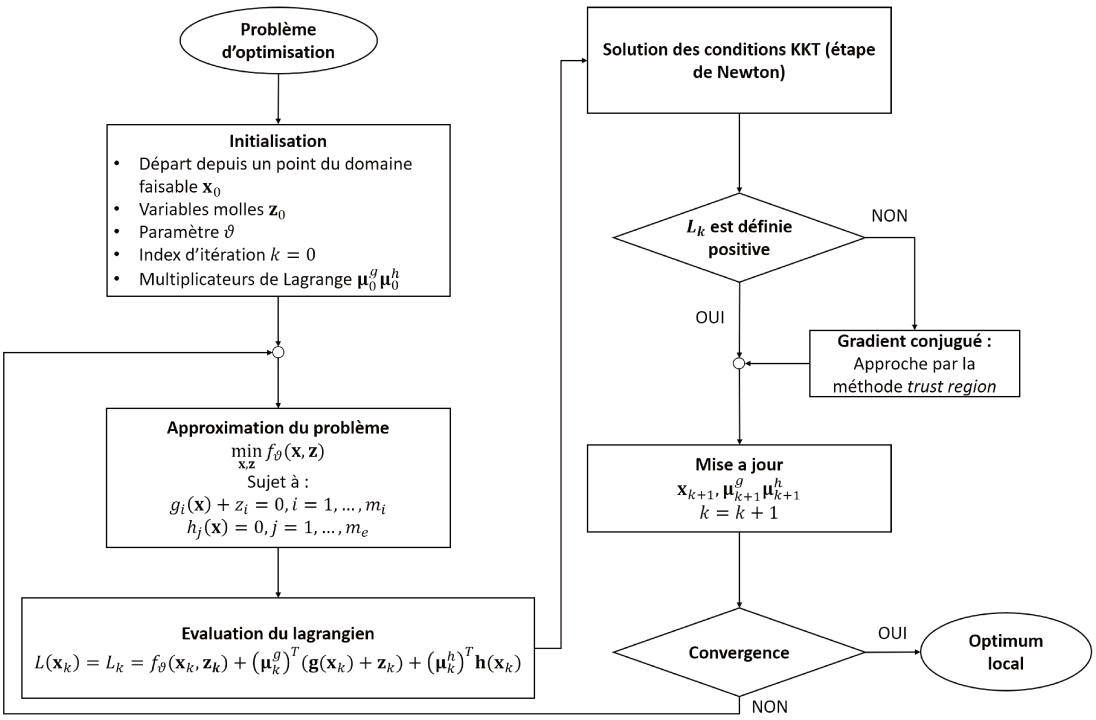


FIGURE 3.16 – Algorithme IP (*Interior Point*)

Algorithme IP (*Interior Point*) Avec les algorithmes SQP, les algorithmes IP font partis des plus efficaces pour la résolution des CNLPPs. De plus, dans certaines conditions, les algorithmes IP sont plus efficaces que les SQP et moins d'itérations sont nécessaires pour converger. Cependant, ils sont un peu moins robustes que les méthodes SQP, particulièrement dans l'exploration des bornes du domaine faisable. L'idée principale de ces méthodes est de changer le CNLPP initial en une séquence de problèmes d'optimisation plus simples, dans lesquels seules les contraintes d'égalité sont prises en compte, comme on peut le voir sur la fig. 3.16 qui présente la vue d'ensemble de l'algorithme IP. A l'itération k, le problème de minimisation approché est formulé comme suit, pour chaque $\vartheta > 0$:

$$\begin{aligned} &\min_{\mathbf{x},\mathbf{z}} f_\vartheta(\mathbf{x}, \mathbf{z}) = f(\mathbf{x}) - \vartheta \sum_{i=1}^{m_i} \ln(z_i) \\ &\text{sujet à :} \\ &\begin{cases} h_j(\mathbf{x}) = 0, & j = 1, ..., m_e, \\ g_i(\mathbf{x}) + z_i = 0, & i = 1, ..., m_i. \end{cases} \end{aligned} \tag{3.53}$$

Cette méthode est souvent appelée *méthode des barrières* en raison du terme logarithmique ajouté au problème (3.53) qui porte le nom de *fonction barrière*. Les variables z_i sont appelées variables molles (*slack variables*) et il y en a autant que de contraintes d'inégalité du problème original. Elles sont strictement positives pour assurer que $\ln(z_i)$ soit bornée. A mesure que ϑ

approche de zéro, le minimum de f_ϑ devrait s'approcher du minimum de f. La séquence de problèmes possédant seulement des contraintes d'égalité est plus simple à résoudre que le problème initial, et la résolution est faite en utilisant principalement deux méthodes. Dans un premier temps, les conditions KKT (éq. (3.51)) sont imposées sur une approximation linéaire du problème considéré. Cette étape est également appelée *Newton step*. Suite à cela, il est possible de vérifier que la matrice hessienne du lagrangien est définie positive. Si c'est le cas, la résolution peut poursuivre et les variables d'optimisation ainsi que les multiplicateurs de Lagrange sont mis à jour. Dans le cas contraire, il est nécessaire de résoudre le problème à partir d'une seconde méthode de résolution, la méthode du *gradient conjugué*. Le problème est ici approximé de manière quadratique et résolu dans le cadre de la méthode *trust region*. Ainsi, par l'une ou l'autre des méthodes, les valeurs des variables d'optimisation $\mathbf{x}_{k+1}$ de l'itération suivante sont obtenues et le critère de convergence peut être vérifié. Notons que le défaut principal de cette méthode est qu'elle a tendance à garder les itérations "éloignées" des limites du domaine faisable et une stratégie doit être mise en place pour l'exploration de ces zones.

Quelle que soit la méthode employée, l'implémentation de ces algorithmes dans MATLAB utilise quatre critères de convergences simultanément [214] :

- *Nombre maximal d'itération* : $k+1 \leq K_{max}$,
- *Faible amélioration de la fonction objectif* : $|f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| < \sigma_f$, avec $0 < \sigma_f \ll 1$,
- *Variation négligeable des valeurs des variables d'optimisation* : $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| < \sigma_x$, avec $0 < \sigma_x \ll 1$,
- *Norme du gradient de la fonction de Lagrange proche de 0* : $\|\nabla L(\mathbf{x}_{k+1})\| < \sigma_\nabla$, avec $0 < \sigma_\nabla \ll 1$.

Lorsqu'au moins un de ces critères est atteint, l'algorithme est stoppé. Il est important de noter que dans cette formulation, la fonction objectif ainsi que les fonctions contraintes doivent être données sans dimension.

3.3.4 Conclusions sur les méthodes d'optimisation

Cette section a présentée quelques principes de base de la formulation d'un problème d'optimisation de type CNLPP et évoquée les principaux outils de résolution disponibles. La principale conclusion qui en résulte est qu'il n'existe pas de *meilleur algorithme* qui serait capable de résoudre le plus efficacement possible tous les problèmes. Le choix de la méthode dépend, en grande partie, du problème à traiter. Nous avons donc focalisé la présentation sur deux méthodes particulières, utilisées dans le cadre de l'optimisation des hypersurfaces NURBS pour la réduction de modèle. Nous avons notamment vu que parmi l'ensemble des métaheuristiques, ERASMUS se démarque par sa capacité à traiter des problèmes extrêmement généraux à nombre de variables d'optimisation non constant. De plus, l'utilisation du langage MATLAB rend l'algorithme très simple à interfacer avec d'autres logiciels ou d'autre langages de programmation. Il faut cependant noter que, malgré la généralité des problèmes pouvant être traités, le nombre de paramètres peut rendre l'optimisation très coûteuse en temps de calcul, notamment si chaque individu doit faire appel à un calcul par éléments finis. Dans de tels cas, des techniques de parallélisation peuvent être mises en place mais il est certain que ERASMUS peut être limité, comme les autres

métaheuristiques, par un trop grand nombre d'individus à traiter (i.e. trop de points de l'espace des paramètres d'optimisation nécessaires à une exploration convenable du domaine). Ceci explique que beaucoup de problèmes soient encore résolus par des méthodes déterministes alors qu'ils sont non-convexes. Même si, dans de tels cas, l'utilisation d'une métaheuristique ne s'avère pas être un bon choix pour des raisons de temps, l'utilisation d'une méthode déterministe n'en est pas un non plus, à moins qu'une connaissance profonde des phénomènes qui régissent le problème considéré permette de choisir un point de départ pertinent. Ceci étant rarement possible, une méthode hybride, couplant une recherche par métaheuristique avec une méthode déterministe, semblerait être une réponse appropriée. La connaissance des phénomènes physiques est alors remplacé par une exploration métaheuristique du domaine d'optimisation afin d'en extraire un point de départ pertinent pour l'algorithme déterministe. C'est le fondement de la méthode que nous avons développée ici et qui sera détaillée dans le chapitre 4.

3.4 Conclusions

Même si l'utilisation des courbes et surfaces NURBS est vastement répandue, notamment dans un contexte de CAO où elle représente la norme pour la représentation des courbes et surfaces depuis les années 90, la notion d'hypersurface NURBS introduite dans ce chapitre est relativement nouvelle. Ces entités bénéficient pourtant de toutes les propriétés des courbes et surfaces et fournissent un outils très versatile pour la réduction de modèle. Cependant, comme nous l'avons vu, cette versatilité a un coût et les méthodes actuelles basées sur les hypersurfaces NURBS pour la réduction de modèle font appel à un grand nombre de règles empiriques pour fixer la valeurs des différents paramètres qui modifient l'hypersurface. La solution fournie par ces méthodes est donc souvent sous-optimale. La recherche de ces paramètres peut être mise sous la forme d'un CNLPP, intrinsèquement non-convexe et avec un domaine d'optimisation de dimension variable, dont la formulation est fournie au chapitre 4. Après une présentation succincte des différentes méthodes de résolution existantes pour les CNLPPs, il s'est avéré qu'aucune méthode ne pouvait être qualifiée de *méthode idéale*. Ce constat nous a conduit à créer une méthode hybride d'optimisation permettant de combiner les avantages de deux grandes classes existantes, à savoir les métaheuristiques et les algorithmes déterministes. Le reste de cette discussion s'est donc focalisée sur les deux méthodes employées par notre approche hybride. L'utilisation de la métaheuristique ERASMUS a grandement était motivée par sa capacité à traiter des problèmes dont le nombre de variables d'optimisation est variable. En particulier, comme nous le verrons dans le chapitre 4, la première étape d'optimisation à l'aide de ERASMUS permet d'obtenir un nombre de points de contrôle optimal pour l'hypersurface NURBS tandis que l'optimisation par la fonction *fmincon* de MATLAB permet de raffiner les valeurs des *knot vectors*.

Chapitre 4

Une nouvelle stratégie de réduction de modèle

4.1 Introduction

L'objectif principal de la réduction de modèle est de réaliser un *métamodèle* nécessitant *moins de ressources* pour obtenir une réponse que le modèle dont il est issu. La notion de *ressources* peut être différente d'un cas d'application à l'autre. Par exemple, dans le cas de la réduction d'image, c'est le nombre de données nécessaires à la restitution d'une certaine qualité que l'on cherche à minimiser. En revanche, dans un problème d'optimisation, c'est le temps de restitution du métamodèle (i.e. le temps nécessaire au métamodèle pour évaluer la, ou les, sorties à partir des paramètres d'entrée) que l'on veut minimiser en priorité. Dans certains cas, réduire le temps de restitution passe également par la réduction du nombre de données nécessaires à cette restitution. Nous verrons que, dans le cadre des hypersurfaces NURBS, diminuer le nombre total de points de contrôle diminue le nombre de données nécessaires au calcul de la réponse du métamodèle tandis que diminuer les degrés dans chaque direction permet d'améliorer le temps de restitution.

Supposons un système de type MIMO (*Multiple Inputs Multiple Outputs*), caractérisé par un nombre d'entrées N et un nombre de sorties M. Ce système admet pour fonction de transfert $\mathbf{z}(\mathbf{X})$ telle que :

$$\begin{array}{rccc} & \mathbb{R}^N & \rightarrow & \mathbb{R}^M, \\ \mathbf{z}: & \mathbf{X} & \rightarrow & \mathbf{z}\left(\mathbf{X}\right), \end{array} \tag{4.1}$$

avec $\mathbf{X} \in \mathbb{R}^N$ le vecteur contenant les N entrées du système, $\mathbf{z}\left(\mathbf{X}\right) \in \mathbb{R}^M$ le vecteur contenant les M sorties du système. Dans certains cas, la fonction $\mathbf{z}$ est connue explicitement et ne demande pas beaucoup de ressources pour être calculée. Cependant, dans le cas de problèmes industriels, la fonction $\mathbf{z}$ est rarement connue explicitement. De plus, la réponse ne peut souvent être obtenue qu'en un certain nombre limité de points, dû a un coup en ressources important pour les obtenir. La réduction de modèle a pour objectif de déterminer une fonction $\hat{\mathbf{z}}$ telle que :

$$\hat{\mathbf{z}}\left(\mathbf{X}\right) = \mathbf{z}\left(\mathbf{X}\right) + \boldsymbol{\epsilon}\left(\mathbf{X}\right), \tag{4.2}$$

avec $\hat{\mathbf{z}}\left(\mathbf{X}\right)$ la fonction du métamodèle permettant d'approximer la valeur de la fonction $\mathbf{z}\left(\mathbf{X}\right)$ et $\boldsymbol{\epsilon}\left(\mathbf{X}\right)$ l'erreur commise au point $\mathbf{X}$. L'intérêt de la fonction $\hat{\mathbf{z}}\left(\mathbf{X}\right)$ est de nécessiter beaucoup moins

de ressources pour être calculée que la fonction $\mathbf{z}(\mathbf{X})$ qu'elle approxime. Il est bien évidemment très compliqué de quantifier $\boldsymbol{\epsilon}(\mathbf{X})$ pour toutes les valeurs de $\mathbf{X}$ et, en général, on s'intéresse aux bornes de cette fonction $\boldsymbol{\epsilon}_{\max}$ et $\boldsymbol{\epsilon}_{\min}$. La fonction $\mathbf{z}(\mathbf{X})$ n'étant pas connue explicitement, ces bornes sont souvent approximées sur un ensemble de points ayant, ou non, été utilisés pour paramétrer la fonction $\hat{\mathbf{z}}(\mathbf{X})$. Dans le cas d'un métamodèle réalisant une interpolation, l'erreur d'approximation aux points ayant servis à paramétrer la fonction $\hat{\mathbf{z}}(\mathbf{X})$ est nulle. Dans ce cas, il est nécessaire d'estimer $\boldsymbol{\epsilon}_{\max}$ et $\boldsymbol{\epsilon}_{\min}$ à partir d'un ensemble de points n'ayant pas servis à paramétrer la fonction $\hat{\mathbf{z}}(\mathbf{X})$.

Plusieurs méthodes permettent de calculer la fonction $\hat{\mathbf{z}}(\mathbf{X})$ (cf. chapitre 2). Elles s'appuient, majoritairement, sur un ensemble de points cibles $\{\mathbf{Q}_s = \mathbf{z}(\mathbf{X}_s)\}$ où la fonction $\mathbf{z}$ a été évaluée de manière exacte ou approchée pour le vecteur d'entrée $\mathbf{X}_s$. Cet ensemble permet d'identifier les différents paramètres de la fonction $\hat{\mathbf{z}}(\mathbf{X})$ minimisant $\boldsymbol{\epsilon}_{\max}$, lié à la précision du métamodèle, et $\boldsymbol{\epsilon}_{moy}$ (l'erreur moyenne), lié à la qualité globale d'approximation. D'autres paramètres peuvent être utilisés comme le coefficient de corrélation de Pearson [1] par exemple. Dans cette thèse, on approxime la fonction $\mathbf{z}(\mathbf{X})$ par une hypersurface NURBS :

$$\begin{array}{rccc} & [0,1]^N & \rightarrow & \mathbb{R}^M, \\ \mathbf{H}: & \mathbf{u} & \rightarrow & \mathbf{H}(\mathbf{u}), \end{array} \tag{4.3}$$

où $\mathbf{H}(\mathbf{u}) \in \mathbb{R}^M$ est le point de l'hypersurface NURBS aux coordonnées dans l'espace paramétrique $\mathbf{u} = \left(u^{(1)}, \ldots, u^{(N)}\right) \in \mathbb{R}^N$. Les coordonnées de $\mathbf{H}(\mathbf{u})$ représentent les M différentes sorties du système tandis que les coordonnées paramétriques $\mathbf{u}$ représentent les N entrées. En lien avec les définitions introduites au chapitre 3, et les pratiques usuelles du cadre des courbes et surfaces NURBS [2], les valeurs de $u^{(k)}$, $k = 1, \ldots, N$, sont bornées dans l'intervalle $[0,1]$. L'espace de départ de la fonction $\mathbf{z}(\mathbf{X})$ étant un sous espace $E \subset \mathbb{R}^N \neq [0,1]^N$, on détermine une fonction *bijective* f telle que :

$$\begin{array}{rccc} & E & \rightarrow & [0,1]^N, \\ \mathbf{f}: & \mathbf{X} & \rightarrow & \mathbf{f}(\mathbf{X}) = \mathbf{u}. \end{array} \tag{4.4}$$

Cette transformation des coordonnées n'est pas nécessaire, les entités NURBS pouvant être définies directement dans l'espace E. Cependant, elle permet de s'affranchir des effets d'échelles des différents types de variables d'entrée. En effet, dans le cas des courbes et surfaces, les coordonnées paramétriques, comme les coordonnées homogènes, représentent le même type de grandeur (i.e. une longueur). Ce n'est pas obligatoirement le cas lorsqu'on modélise un système de type MIMO, et cette transformation permet donc d'éliminer les effets d'échelles dus aux paramètres d'entrée qui n'ont pas forcement la même nature. Une telle transformation n'est pas nécessaire pour les coordonnées homogènes de $\mathbf{H}(\mathbf{u})$. La fonction $\hat{\mathbf{z}}(\mathbf{X})$ permettant d'approximer la fonction $\mathbf{z}(\mathbf{X})$ est donc définie de la manière suivante :

$$\begin{array}{rccc} & E & \rightarrow & \mathbb{R}^M, \\ \hat{\mathbf{z}}: & \mathbf{X} & \rightarrow & \mathbf{H}(\mathbf{f}(\mathbf{X})). \end{array} \tag{4.5}$$

La construction de la fonction $\hat{\mathbf{z}}$ implique de déterminer les différents paramètres de l'hypersurface NURBS permettant d'obtenir une erreur d'approximation inférieure à un certain seuil. Pour cela, les méthodes actuelles basées sur les hypersurfaces NURBS se focalisent sur la détermination du nombre de points de contrôle garantissant que l'erreur d'approximation soit en

dessous du seuil fixé par l'utilisateur. Tous les autres paramètres de l'hypersurface sont fixés par des règles empiriques qui bride les possibilités offertes par le formalisme des entités NURBS. De plus, ces règles sont difficilement justifiables puisqu'elles ne reposent sur aucun principe physique. Leur intérêt principal est qu'elles simplifient le calcul des coordonnées homogènes des points de contrôle, obtenus par la résolution d'un système d'équations linéaires. Elles assurent notamment que le déterminant de la matrice à inverser soit non-nul. Elles permettent également de fortement limiter le nombre de paramètres à fixer pour l'utilisateur. Même si elles dérivent des règles bien connues du formalisme des courbes et surfaces, leur utilisation peut être remise en cause dans le cadre de la réduction de modèle. En effet, dans le cas des courbes et surfaces, la taille des matrices à inverser ainsi que les ressources nécessaires (stockage des points de contrôle et temps de calcul des points de la courbe ou de la surface) ne limite pas l'utilisation des NURBS. En revanche, dans le contexte de la réduction de modèle, il est très pertinent de réduire le nombre de données nécessaire (réduction d'image par exemple) ou le temps d'évaluation des points de l'hypersurface (boucle d'optimisation par exemple). C'est dans cette optique que nous avons développé une méthode capable de fournir *tous* les paramètres optimaux de l'hypersurface NURBS et pas seulement le nombre et les coordonnées homogènes des points de contrôle.

Pour ce faire, le problème de réduction de modèle (i.e. la détermination de la fonction $\hat{\mathbf{z}}$) est traité comme un problème d'*hypersurface fitting* par une hypersurface NURBS sur un ensemble de points cibles. Ce problème d'optimisation est traité sans introduire aucune hypothèse simplificatrice sur les paramètres de l'hypersurface, contrairement aux méthodes actuelles. Par conséquent, le problème d'optimisation qui en résulte est défini sur un domaine à dimension variable et nous verrons qu'il peut être formulé comme un problème d'optimisation de systèmes modulaires avec différents types de modules, dont la notion a été introduite au chapitre 3. Dans la suite ce chapitre, nous allons introduire la stratégie permettant d'obtenir les paramètres optimaux du modèle réduit. Cette stratégie s'appuie sur l'utilisation d'une métaheuristique, l'algorithme génétique ERASMUS [212], et d'une méthode déterministe implémentée dans MATLAB, notamment l'algorithme AS implémenté dans la fonction *fmincon*, introduites au chapitre 3. La méthode que nous proposons est utilisée dans le cadre de l'*approximation* à partir d'une base de données de points connus, aussi appelés *points cibles*, mais elle est tout à fait transposable au cas de l'*interpolation* (i.e. le nombre de points de contrôle est le même que le nombre de points cibles). Les coordonnées homogènes des points de contrôle ne font pas partie des variables d'entrée de notre métaheuristique car elles peuvent être obtenues par l'inversion d'un système d'équations linéaires. Toutefois, cette matrice peut atteindre des tailles importantes et les algorithmes permettant de diminuer la charge de calcul seront présentés dans la section 4.3.

4.2 Formulation mathématique du problème de réduction de modèle comme problème d'optimisation d'un système modulaire

4.2.1 Variables d'optimisation

Le formalisme des hypersurfaces NURBS a été introduit au chapitre précédent et nous avons mis en évidence qu'un certain nombre de paramètres permettait de modifier l'entité obtenue. Les notations introduites restent valables et notamment nous rappelons que l'hypersurface NURBS

est une application : $\mathbf{H}(\mathbf{u}) : \mathbb{R}^N \rightarrow \mathbb{R}^M$ où N désigne le nombre de paramètres d'entrée du métamodèle et M représente la dimension de l'espace des sorties du modèle réduit. L'hypersurface est ajustée à partir d'une base de données de points cibles,

$$\left\{ \mathbf{Q}_{s_1,...,s_N} = \mathbf{z}(\mathbf{X}_{s_1,...,s_N}), \mathbf{z}(\mathbf{X}_{s_1,...,s_N}) = \left(z^{(1)}(\mathbf{X}_{s_1,...,s_N}), ..., z^{(M)}(\mathbf{X}_{s_1,...,s_N}) \right) \in \mathbb{R}^M \right\}, \quad (4.6)$$

avec

$$\mathbf{X}_{s_1,...,s_N} = \left(X_{s_1}^{(1)}, ..., X_{s_N}^{(N)} \right) \in \mathbb{R}^N, \quad s_k = 0, ..., r_k, \quad (4.7)$$

où $\mathbf{X}_{s_1,...,s_N}$ est le vecteur d'entrée de la fonction $\mathbf{z}$ et r_k+1 est le nombre de points cibles mesurés dans la direction k, $k = 1, ..., N$. Dans la suite du chapitre, on notera

$$z^{(j)}(\mathbf{X}_{s_1,...,s_N}) = Q_{s_1,...,s_N}^{(j)}, \quad j = 1, \ldots, M. \quad (4.8)$$

Le nombre total de points cibles est donné par :

$$n_{TP} = \prod_{k=1}^{N} (r_k + 1), \quad (4.9)$$

tandis que le nombre de points de contrôle est donné par la relation :

$$n_{CP} = \prod_{k=1}^{N} (n_k + 1), \quad (4.10)$$

$n_k + 1$ étant le nombre de points de contrôle dans la direction k.

L'eq. (3.21), qui définit une hypersurface NURBS, permet de remarquer que les paramètres qui régissent la forme de l'hypersurface sont de différentes natures.

- *Variables entières.* Parmi les variables appartenant à l'ensemble des entiers, nous dénombrons les nombres de points de contrôle n_k+1, les nombres de *knots* des *knot vectors* m_k+1 ainsi que les degrés p_k dans chaque direction de l'espace $\mathbb{R}^N$.
- *Variables continues.* Les variables appartenant à l'ensemble des réels sont plus nombreuses. Elles comptent les coordonnées paramétriques $u_{s_k}^{(k)}$, $s_k = 0, ..., r_k$, $k = 1, ..., N$, auxquelles l'hypersurface est évaluée, les différentes valeurs des composantes des *knot vertors* $U_{l_k}^{(k)}$, $l_k = 0, ..., m_k + 1$, $k = 1, ..., N$, dans chaque direction de l'espace paramétrique, les valeurs des coordonnées homogènes des points de contrôle $P_{i_1,...,i_N}^{(j)}$, $j = 1, ..., M$, $i_k = 0, ..., n_k$, $k = 1, ..., N$, pour chaque direction de l'espace des sorties du modèle et les poids qui leurs sont associés $\omega_{i_1,...,i_N}$, $i_k = 0, ..., n_k$.

Certains de ces paramètres sont interdépendants, tandis que d'autres peuvent être intelligemment choisis. Comme nous l'avons vu au chapitre 3, le nombre de *knots* $m_k + 1$ dépend du nombre de points de contrôle $n_k + 1$ et du degré p_k dans la direction k, $k = 1, ..., N$. Ils sont liés par l'éq. (3.6) que nous rappelons ici :

$$m_k = n_k + p_k + 1, \quad k = 1, ..., N. \quad (4.11)$$

De plus, les $m_k + 1$ valeurs des *knot vectors* ne sont pas toutes inconnues, puisque les $p_k + 1$ premières valeurs sont fixées à 0 tandis que les $p_k + 1$ dernières sont fixées à 1. Il n'y a donc en fait que $n_k - p_k$ valeurs inconnues pour chacun des *knot vectors*. Cette quantité ne pouvant pas être négative, il est important de noter que, dans chaque direction k, le nombre de points de contrôle $n_k + 1$ doit être strictement supérieur au degré p_k.

Pour le calcul des coordonnées paramétriques auxquelles seront évaluées les coordonnées de l'hypersurface NURBS, la fonction $\mathbf{f}(\mathbf{X})$ est définie de la manière suivante :

$$\begin{array}{rccl} & E & \rightarrow & [0,1]^N, \\ \mathbf{f}: & \mathbf{X} & \rightarrow & \mathbf{f}(\mathbf{X}) = \left(f^{(1)}\left(X^{(1)}\right), \ldots, f^{(N)}\left(X^{(N)}\right)\right) = \mathbf{u}, \end{array} \tag{4.12}$$

avec $\mathbf{u}$ les coordonnées paramétriques liées aux coordonnées d'entrée du métamodèle $\mathbf{X}$, E l'espace de départ de la fonction $\mathbf{f}$ défini par :

$$E = \left[X_{\min}^{(1)}, X_{\max}^{(1)}\right] \times \ldots \times \left[X_{\min}^{(N)}, X_{\max}^{(N)}\right], \tag{4.13}$$

et $f^{(k)}\left(X^{(k)}\right)$ la fonction bijective dans la direciton k définie de la manière suivante :

$$\begin{array}{rccl} & \left[X_{\min}^{(k)}, X_{\max}^{(k)}\right] & \rightarrow & [0,1], \\ f^{(k)}: & X^{(k)} & \rightarrow & f^{(k)}\left(X^{(k)}\right) = \dfrac{X^{(k)} - X_{min}^{(k)}}{X_{max}^{(k)} - X_{min}^{(k)}} = u^{(k)}. \end{array} \tag{4.14}$$

La fonction $\mathbf{f}(\mathbf{X})$ permet de réaliser la bijection de l'espace d'entrée du métamodèle E sur l'espace paramétrique de l'hypersurface NURBS $[0,1]^N$. Les coordonnées paramétriques $u_{s_k}^{(k)}$, $s_k = 0, ..., r_k$, $k = 1, ..., N$, sont ensuite obtenues à partir des coordonnées des points cibles dans l'espace des entrées du métamodèle $X_{s_k}^{(k)}$ par la relation suivante :

$$u_{s_k}^{(k)} = \frac{X_{s_k}^{(k)} - X_{min}^{(k)}}{X_{max}^{(k)} - X_{min}^{(k)}}, \quad s_k = 0, ..., r_k, \quad k = 1, ..., N, \tag{4.15}$$

où $X_{s_k}^{(k)}$ est la k^{e} coordonnée du $s_k{}^{\text{e}}$ point cible dans la direction k, $X_{min}^{(k)}$ et $X_{max}^{(k)}$ représentent les bornes du domaine de définition de la variable $\mathbf{X}$ dans la direction k et $r_k + 1$ représente le nombre de points cibles mesurés dans cette direction. Par définition, une entité NURBS passe exactement par les bords du domaine [2] et il est donc obligatoire d'avoir ces points parmi les points cibles.

L'utilisation des poids $\omega_{i_1,\ldots,i_N}$, $i_k = 0, ..., n_k$, augmente grandement le nombre de données nécessaires pour l'évaluation du métamodèle. De plus, le nombre de poids peut atteindre un nombre conséquent, limitant l'utilisation de l'algorithme génétique. Dans cet optique, notre approche utilise dans un premier des B-Splines (i.e. tous les poids sont égaux à 1) et dans un second temps, seuls les poids dans les zones où l'erreur d'approximation dépasse le seuil fixé par l'utilisateur sont optimisés. Cela est rendu possible par le caractère intrinsèquement local des hypersurfaces NURBS (i.e. le fait qu'un point de contrôle agit uniquement dans la zone définie par son support local).

Lorsque tous les paramètres précédents sont fixés, les valeurs des coordonnées homogènes des points de contrôle $\mathbf{P}_{i_1,...,i_N}$, $i_k = 0, ..., n_k$, sont obtenues par la résolution d'un système d'équations linéaires. Même s'il s'agit d'un problème "simple", la taille de la matrice à inverser lorsque la dimension de l'espace N, ou que les nombres de points cibles dans chaque direction $r_k + 1$, augmentent peut s'avérer une limitation importante. En effet, l'espace de stockage nécessaire peut facilement atteindre les téraoctets. C'est pour cette raison que des algorithmes plus performants, et ne nécessitant pas le stockage de cette matrice, ont été développés dans le cadre des courbes et surfaces NURBS. La généralisation de ces algorithmes sera présentée dans ce chapitre à la section 4.3.

En résumé, tous les paramètres de la NURBS ne font pas partie des variables d'optimisation de l'hypersurface. Seuls les paramètres indépendants de cette hypersurface sont optimisés, les autres étant liés ou fournis par la résolution d'un système d'équations linéaires. Les variables d'optimisation de notre métamodèle sont donc :

- les degrés p_k, $k = 1, ..., N$,
- les tailles des *knot vectors* $m_k + 1$, $k = 1, ..., N$,
- les valeurs internes (i.e. non prédéterminées) des *knot vectors* $U^{(k)}_{l_k}$, $l_k = p_k + 1, ..., m_k - p_k - 1$, $k = 1, ..., N$,
- les poids $\omega_{i_1,...,i_N}$, $i_k = 0, ..., n_k$.

Le nombre de degrés p_k est N, ce qui correspond à la dimension de l'espace des entrées du métamodèle. Il y a également N tailles de *knot vector* $m_k + 1$ à déterminer. Le nombre de points de contrôle est, quant à lui, déterminé par l'éq. (4.11) avec $p_k \leq n_k \leq r_k$. Contrairement aux paramètres précédents, dont le nombre est fixe, le nombre de valeurs des composantes à définir pour chaque *knot vector* n'est pas constant et varie d'une direction de l'espace à l'autre. De plus, ces nombres sont liés à la fois aux degrés p_k et aux dimensions des *knot vectors* $m_k + 1$ (ou aux nombres de points de contrôle $n_k + 1$) :

$$\begin{gathered} n^{(k)}_{knots} = n_k - p_k, \quad k = 1, ..., N, \\ \text{ou} \\ n^{(k)}_{knots} = m_k - 2p_k - 1, \end{gathered} \tag{4.16}$$

avec $n^{(k)}_{knots}$ le nombre de valeurs des composantes à définir pour le *knot vector* $\mathbf{U}^{(k)}$ dans la direction de l'espace k, $k = 1, ..., N$. On peut donc remarquer que la dimension du vecteur des paramètres d'optimisation $\mathbf{x}$ est donnée par :

$$n_{var} = 2N + \sum_{k=1}^{N} (m_k - 2p_k - 1) + \prod_{k=1}^{N} (n_k + 1), \tag{4.17}$$

avec n_{var} le nombre de variables du problème d'optimisation. Ce nombre n'est pas constant et dépend de certaines des variables d'optimisation. La stratégie de résolution employée repose sur l'utilisation de l'algorithme génétique ERASMUS présenté au chapitre 3 et sera détaillée en fin de chapitre.

Il est toutefois important de préciser en quoi la recherche des paramètres optimaux d'une hypersurface NURBS peut être vue comme le problème d'optimisation d'un système modulaire

possédant plusieurs types de modules. Pour cela notons, dans un premier temps, que le nombre de certaines variables d'optimisation ne varie pas. En effet, quelle que soit l'hypersurface NURBS générée, il est nécessaire de fixer N paramètres m_k+1 correspondant à la longueur des *knot vectors* dans chaque direction k, et N paramètres p_k correspondants aux degrés des fonctions de base dans les différentes directions k de l'espace. L'autre partie des variables d'optimisation est lié aux valeurs internes des différents *knot vectors*, dont le nombre dépend des variables d'optimisation précédentes (i.e. les degrés p_k et les tailles des *knot vectors* $m_k + 1$). Chacun de ces *knot vectors* $\mathbf{U}^{(k)}$, $k = 1, ..., N$, peut être considéré comme un type de module à optimiser, chaque *knot vector* étant indépendant des autres. L'hypersurface NURBS peut donc être vue comme un système modulaire comportant N *knot vectors*, ou types de module, dont les valeurs sont à optimiser au même titre que les N degrés p_k et les N tailles $m_k + 1$ des différents *knot vectors*. La résolution de ce problème est rendu possible par l'utilisation de la métaheuristique ERASMUS, capable d'optimiser à la fois le nombre de modules ainsi que leurs valeurs pour chaque type de module simultanément.

4.2.2 Méthodes itératives existantes de recherche des paramètres de l'hypersurface NURBS

Actuellement, les méthodes permettant de définir les valeurs des différentes variables d'optimisation de l'hypersurface NURBS sont basées sur des méthodes itératives. Elles peuvent être classées en deux catégories, à savoir, les méthodes par ajout ou par suppression itératif de points de contrôle. Dans le premier cas, à partir d'un nombre minimal de points de contrôle, d'autres sont ajoutés de manière itérative jusqu'à ce qu'un certain critère soit satisfait. Dans le second cas, à partir d'une hypersurface d'interpolation (i.e. nombre de points de contrôle égal au nombre de points cibles), des points de contrôle sont successivement supprimés jusqu'à ce qu'un seuil d'erreur soit atteint. Certaines méthodes utilisent les deux approches afin de déterminer le nombre de points de contrôle à attribuer à l'hypersurface. La principale limitation de ces approches est qu'elles ne se focalisent que sur la recherche du nombre de points de contrôle, tandis que les autres paramètres doivent être fixés par l'utilisateur. Nous détaillons, ci-après, trois méthodes basées sur l'ajout successif de points de contrôle, les principes utilisés étant facilement transposables à l'approche par suppression.

Quelle que soit l'approche itérative employée, l'utilisateur doit fixer, en premier lieu, les degrés p_k des fonctions de base dans chacune des directions $k = 1, ..., N$ de l'espace. Ce choix peut être facilité dans le cas où la continuité souhaitée pour l'hypersurface dans la direction k est connue. Dans le cas des courbes et surfaces NURBS, par exemple, des degrés $p_k = 3$ sont souvent utilisés car cela permet d'imposer des contraintes sur les tangences et courbures de l'entité NURBS. L'utilisateur doit également fixer les valeurs des *knot vectors*. Pour cela, il existe principalement deux règles empiriques [2] :

- *Knot vectors* uniformes. Les valeurs des nœuds sont espacées de manière régulière, i.e.

$$\begin{cases} U_{l_k} = 0, & l_k = 1, \ldots, p_k, \\ U^{(k)}_{l_k+p_k} = \dfrac{l_k}{n_k - p_k + 1}, & l_k = 1, \ldots, n_k - p_k, \\ U_{l_k} = 1, & l_k = m_k - p_k, \ldots, m_k. \end{cases} \tag{4.18}$$

Ce type de *knot vector* est plus adapté à des points cibles répartis uniformément.

- *Knot vector* moyennés par rapport aux points cibles. Les valeurs des nœuds, dans ce cas, sont données par :

$$\begin{cases} U_{l_k} = 0, & l_k = 1, \ldots, p_k, \\ U^{(k)}_{l_k+p_k} = \dfrac{1}{p_k} \sum_{j=l_k}^{l_k+p_k-1} u_j^{(k)}, & l_k = 1, \ldots, n_k - p_k, \\ U_{l_k} = 1, & l_k = m_k - p_k, \ldots, m_k. \end{cases} \tag{4.19}$$

Ce type de *knot vector* reflète beaucoup mieux la répartition des points cibles. De plus, associé à la méthode du *chord length*, elle assure qu'au moins un point cible est présent entre deux nœuds [2], ce qui facilite le calcul des coordonnées des points de contrôle.

Toutes les stratégies d'optimisation d'hypersurface NURBS existantes étant axées uniquement sur le nombre de points de contrôle, les poids doivent également être fixés par l'utilisateur. Une technique répendue consiste à fixer la valeur de tous les poids à 1 et, ainsi, utiliser des entités B-Splines. Dans le contexte des courbes et surfaces, les poids sont introduits essentiellement pour représenter des coniques, ne pouvant être représentées exactement qu'avec des NURBS et non des B-Splines [2]. Il existe également des règles empiriques permettant de fixer la valeur des poids. Nous détaillons brièvement ici la pondération utilisée par Turner [2]. Le poids $\omega_{i_1,\ldots,i_N}$ affecté au point de contrôle $\mathbf{P}_{i_1,\ldots,i_N}$ est donné par la relation suivante :

$$\omega_{i_1,\ldots,i_N} = \omega_{\min} + (\omega_{\max} - \omega_{\min}) \left(\mathbf{r}^T_{i_1,\ldots,i_N} \mathbf{R}^{-1} \mathbf{r}_{i_1,\ldots,i_N} \right), \tag{4.20}$$

avec $\omega_{\min}$ et $\omega_{\max}$ les poids minimum et maximum respectivement, tandis que $\mathbf{r}_{i_1,\ldots,i_N}$ et $\mathbf{R}$ sont définis à partir d'une fonction de corrélation spatiale (cf. chapitre 2) de la manière suivante :

$$\mathbf{r}^T_{i_1,\ldots,i_N} = \begin{pmatrix} R(\mathbf{u}_{i_1,\ldots,i_N}, \mathbf{u}_0) & \cdots & R(\mathbf{u}_{i_1,\ldots,i_N}, \mathbf{u}_{n_{CP}}) \end{pmatrix}, \tag{4.21}$$

$$\mathbf{R} = \begin{pmatrix} R(\mathbf{u}_0, \mathbf{u}_0) & \cdots & R(\mathbf{u}_0, \mathbf{u}_{n_{CP}}) \\ \vdots & \ddots & \vdots \\ R(\mathbf{u}_{n_{CP}}, \mathbf{u}_0) & \cdots & R(\mathbf{u}_{n_{CP}}, \mathbf{u}_{n_{CP}}) \end{pmatrix}, \tag{4.22}$$

où $\mathbf{u}_{i_1,\ldots,i_N} = \left(u^{(1)}_{i_1}, \ldots, u^{(N)}_{i_N} \right)$ sont les coordonnées dans l'espace paramétrique du point de contrôle $P_{i_1,\ldots,i_N}$, $\mathbf{u}_s = \left(u^{(1)}_s, \ldots, u^{(N)}_s \right)$ sont les coordonnées dans l'espace paramétrique du points cible s et $R(\cdot,\cdot)$ est une fonction de corrélation spatiale dont l'expression en dimension $N = 1$ est la suivante :

$$R(u_{s_1}, u_{s_2}) = \exp^{-\theta |u_{s_1} - u_{s_2}|^\tau}. \tag{4.23}$$

Dans l'éq. (4.23), θ représente l'intervalle d'influence des points cibles et τ définit la rapidité à laquelle l'influence décroît avec la distance. Turner [1] a défini que ces deux paramètres pouvait être choisis en fonction du poids minimum $\omega_{\min}$ et du nombre de points de contrôle n_{CP} par les relations suivantes :

$$\theta = \ln(\omega_{\min}), \tag{4.24}$$

$$\tau = \frac{\ln\left(\ln\left(C\right)\right)}{\ln\left(\dfrac{1}{n_{CP}}\right)}, \tag{4.25}$$

où C ($C > 1$) est un coefficient définissant l'influence minimal du poids près des frontières du voisinage. Turner [1] fournit des valeurs par défaut de $\omega_{\min} = 0.1, \omega_{\max} = 1$ et $C = 2$. L'utilisation de poids, dans le contexte de la réduction de modèle, permet d'estimer la "confiance" en un point de contrôle. Plus ce point de contrôle est proche des points cibles, plus la confiance en ce point sera grande. Ces résultats sont cependant empiriques et il n'existe pas de justification physique permettant de fixer la valeur des paramètres. De plus, la fonction de corrélation spatiale donnée dans l'éq. (4.23) est valable dans le cas $N = 1$. En dimension N quelconque, la pratique courante consiste, comme dans le cas du krigeage [9], à utiliser la règle du produit de corrélation. Dans ce cas, la fonction de corrélation spatiale en dimension N quelconque est donnée par le produit des fonctions de dimension 1 :

$$R(\mathbf{u}_{s_1}, \mathbf{u}_{s_2}) = \prod_{k=1}^{N} R\left(u_{s_1}^{(k)}, u_2^{(k)}\right), \tag{4.26}$$

où $R\left(\cdot,\cdot\right)$ est la fonction de corrélation spatiale en dimension 1 définie par l'éq. (4.23). Notons qu'il est possible de définir une fonction $R^{(k)}\left(\cdot,\cdot\right)$, dépendant de la direction k de l'espace paramétrique, en définissant des coefficients $\theta^{(k)}$ et $\tau^{(k)}$ différents dans chaque direction. Cependant, avec la définition de Turner, les coefficients $\theta^{(k)}$ serait les mêmes pour toutes les directions ($\omega_{\min}$ étant le même pour toutes les directions), et les différents paramètres de lissage $\tau^{(k)}$ seraient donnés par :

$$\tau^{(k)} = \frac{\ln\left(\ln\left(C\right)\right)}{\ln\left(\dfrac{1}{n_k + 1}\right)}, \tag{4.27}$$

où $n_k + 1$ est le nombre de points de contrôle dans la direction de l'espace paramétrique k.

L'ensemble des paramètres précédents étant définis, les méthodes itératives se focalisent sur la recherche du nombre de points de contrôle minimal permettant de garantir un certain seuil d'erreur d'approximation. Pour cela, le nombre de points de contrôle évolue à chaque itération et leurs coordonnées sont calculées par la résolution d'un système d'équations linéaires. La stratégie de résolution utilisée est détaillée plus tard dans le chapitre. Nous explicitons simplement ici que le système d'équations linéaires à résoudre est obtenu en réalisant une approximation des moindres carrés dont la formulation est la suivante :

$$\min g_h = \sum_{s=1}^{n_{TP}} \left(\mathbf{Q}_s - \mathbf{H}(\mathbf{u}_s)\right)^2, \tag{4.28}$$

où $\mathbf{Q}_s$ est le s^{e} point cible des n_{TP} points cibles et $\mathbf{H}(\mathbf{u}_s)$ le point de l'hypersurface NURBS calculé aux coordonnées paramétriques $\mathbf{u}_s$ du point cible $\mathbf{Q}_s$. Le système d'équations linéaires qui résulte de la minimisation de la fonction g_h, ainsi que la manière de l'obtenir, sont détaillés plus loin dans le chapitre à la section 4.3.

Les deux premières stratégies itératives présentées utilisent l'approximation des moindres carrés pour le calcul des coordonnées des points de contrôle. Elles sont des généralisations du

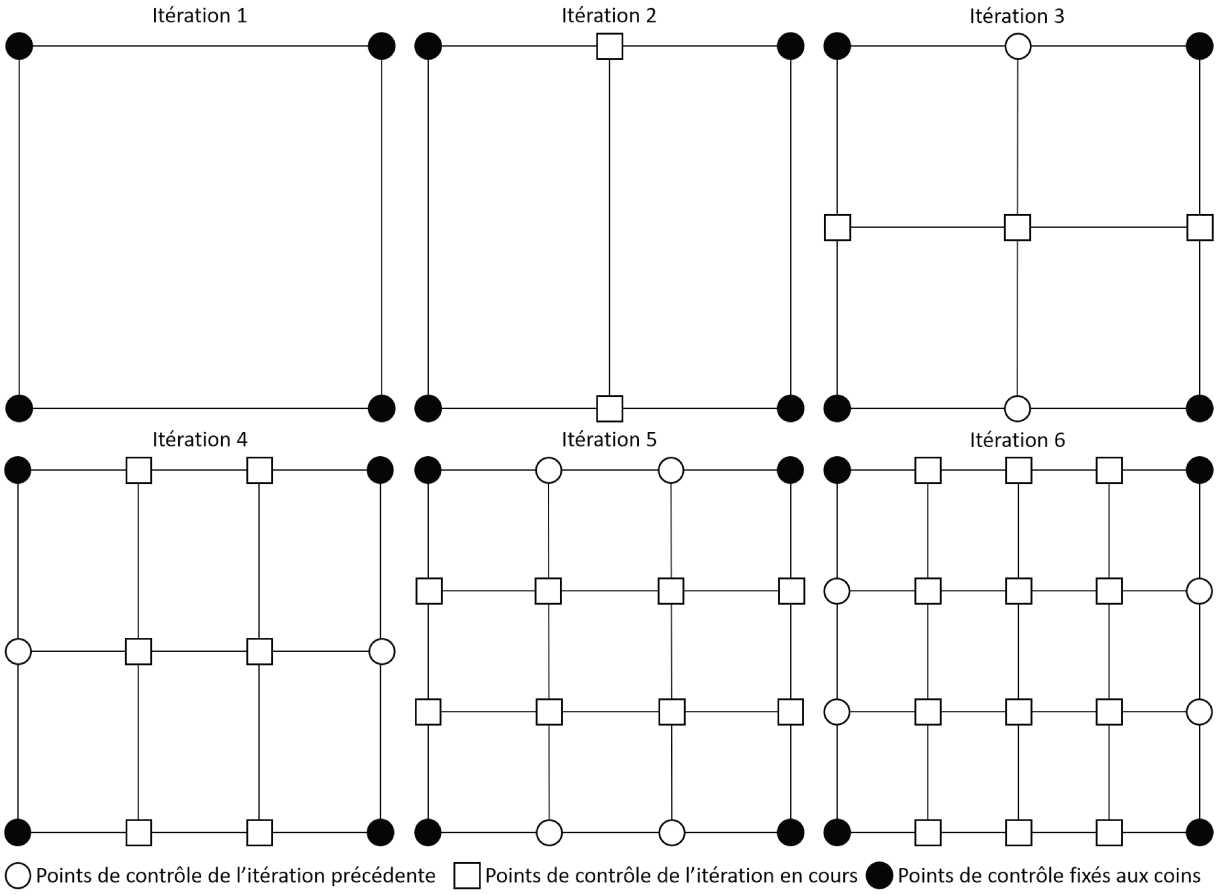


FIGURE 4.1 – Stratégie itérative d'ajout de points de contrôle dimension par dimension

cas des courbes et surfaces NURBS [2]. Une première méthode, dont le cas en dimension $N = 2$ est présenté à la fig. 4.1, consiste à ajouter des points de contrôle en augmentant itérativement le nombre de points de contrôle $n_k + 1$ dans une direction k de l'espace paramétrique. Initialement, le nombre minimal de points de contrôle (dépendant des degrés p_k fixés par l'utilisateur) est attribué dans chaque direction. Si l'erreur d'approximation dépasse le seuil d'erreur fixé par l'utilisateur alors, à l'itération suivante, l'indice maximale n_k est incrémenté de 1 dans la direction k. Cela implique de choisir la direction k dans laquelle le nombre de points de contrôle sera augmenté. La stratégie la plus simple consiste à incrémenter chacune des directions successivement (i.e. direction 1, puis 2, et ainsi de suite jusqu'à la direction N). Une autre possibilité est de calculer l'erreur maximale d'approximation pour un ajout dans chaque direction, puis de choisir la direction dans laquelle l'erreur est la plus faible. Cependant, cela implique un temps de calcul beaucoup plus important. Enfin, il est possible de choisir une direction reflétant la distribution

des points cibles, en incrémentant la direction k dans laquelle le ratio suivant est le plus faible :

$$q_k = \frac{n_k + 1}{r_k + 1}, \quad k = 1, \ldots, N, \tag{4.29}$$

où $n_k + 1$ est le nombre de points de contrôle et $r_k + 1$ est le nombre de points cibles dans la direction k. Dans la fig. 4.1, la stratégie employée est d'incrémenter le nombre de points de contrôle dans chaque direction l'une après l'autre. Dans un premier temps, la direction horizontale voit son nombre de points de contrôle augmenter. C'est ensuite dans la direction verticale que le nombre de points de contrôle est augmenté. L'opération est répétée jusqu'à ce que l'erreur d'approximation ne dépasse plus le seuil fixé par l'utilisateur. Cette méthode permet une évolution progressive du nombre total de points de contrôle mais leurs position peut varier d'une itération à l'autre, et il est donc possible que l'erreur d'approximation augmente entre deux itérations. Notons, toutefois, que l'erreur d'approximation admet une asymptote à 0 correspondant au cas de l'interpolation (i.e. autant de points de contrôle que de points cibles).

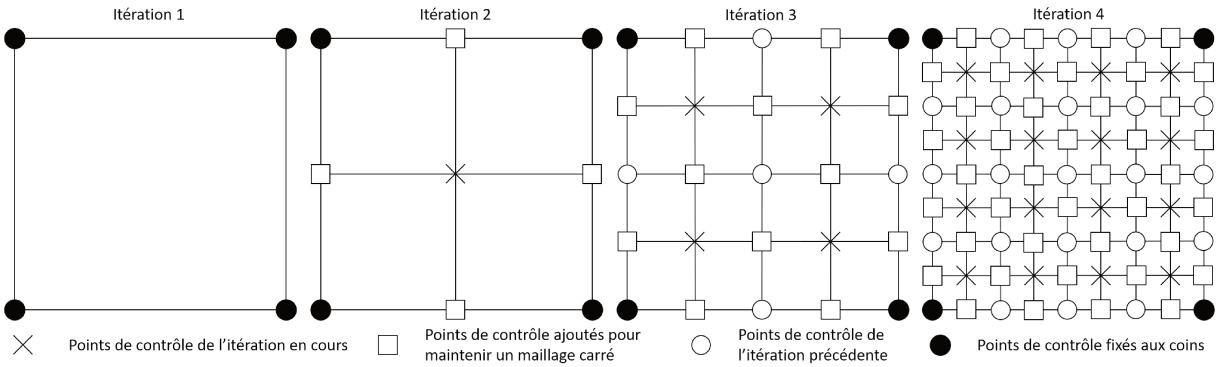


FIGURE 4.2 – Stratégie itérative d'ajout de points de contrôle par subdivision de l'espace

Une deuxième méthode, dont le cas $N = 2$ est présenté à la fig. 4.2, consiste à ajouter des points de contrôle simultanément dans toutes les directions, à chaque itération, en subdivisant l'espace. Cela permet de conserver la position des points de contrôle des itérations précédentes. Elle permet également de s'affranchir du choix de la position des points à ajouter. Cependant, comme on peut le voir sur la fig. 4.2, le nombre total de points de contrôle croît très rapidement et la croissance est d'autant plus rapide que la dimension N augmente. Afin de limiter cette croissance, il est possible de mettre en place des stratégies de choix de position des points de contrôle à ajouter. Notamment, il est possible d'envisager des stratégies ne conservant pas un maillage uniforme de points de contrôle. Ainsi, la fig. 4.3 montre une approche basée également sur la subdivision de l'espace, mais l'ajout de points n'est pas systématique dans chaque subdivision de l'espace. Un point est ajouté au barycentre de la subdivision ayant la plus grande erreur d'approximation. Les autres points qui s'ajoutent permettent de maintenir un maillage "carré". Comme on peut le voir sur la fig. 4.3, la croissance du nombre de points de contrôle est bien plus faible que dans le cas de la fig. 4.2. Il n'est cependant plus possible, dans ce cas, d'utiliser une approximation des moindres carrés pour calculer la valeur des points de contrôle.

En effet, la position des points de contrôle dans l'espace paramétrique est implicite lorsqu'on utilise l'approximation des moindres carrés.

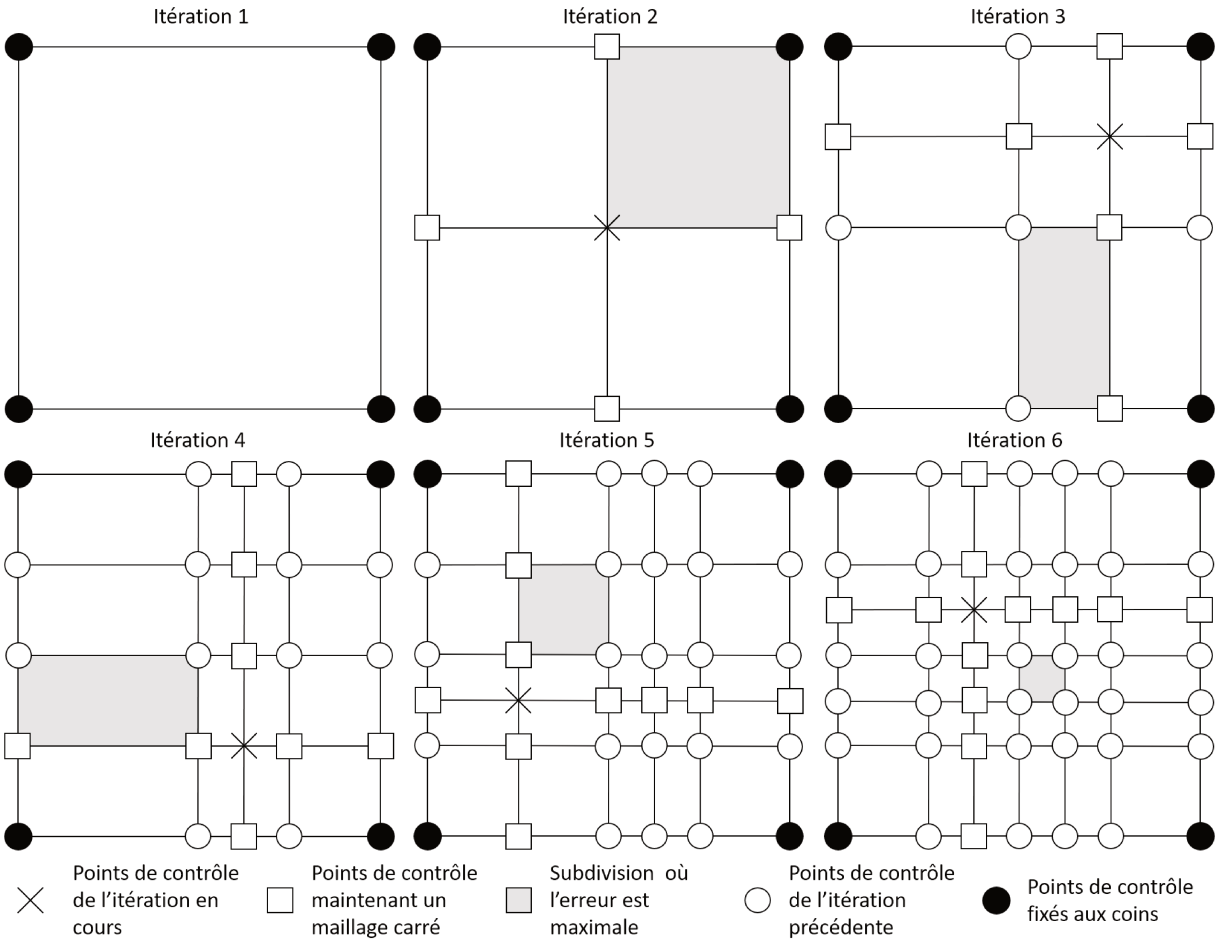


FIGURE 4.3 – Stratégie itérative d'ajout de points de contrôle dans la subdivision de l'espace ayant la plus grande erreur d'approximation

La méthode itérative développée par Turner [1, 139] est également une méthode dont le positionnement des points de contrôle est explicite dans l'espace paramétrique. Pour cela, le système d'équations linéaires résolu est similaire au système généré par l'interpolation de points cibles. Dans ce dernier cas, il y a autant de points cibles que de points de contrôle (i.e. $n_{CP} = n_{TP}$) et l'hypersurface passe par tous les points cibles, ce qui se traduit par :

$$\mathbf{Q}_s = \mathbf{H}(\mathbf{u}_s) = \sum_{i_1=0}^{n_1} \cdots \sum_{i_N=0}^{n_N} R_{i_1,\ldots,i_N}\left(u_s^{(1)},\ldots,u_s^{(N)}\right)\mathbf{P}_{i_1,\ldots,i_N}, \quad s = 1,\ldots,n_{TP}, \tag{4.30}$$

avec $\mathbf{Q}_s$ le s^{e} point cible, $\mathbf{H}(\mathbf{u}_s)$ le points de l'hypersurface NURBS correspondant aux coordonnées paramétriques du s^{e} point cible $\mathbf{u}_s = \left(u_s^{(1)},\ldots,u_s^{(N)}\right)$, $\mathbf{P}_{i_1,\ldots,i_N}$ le $(i_1,\ldots,i_N)^{\text{e}}$ point de

contrôle et $R_{i_1,\ldots,i_N}\left(u_s^{(1)},\ldots,u_s^{(N)}\right)$ les fonctions rationnelles par morceaux introduites dans l'éq. (3.23). Le système d'équations linéaires qui en résulte peut être mis sous la forme suivante :

$$\mathbf{NP} = \mathbf{Q}, \tag{4.31}$$

où $\mathbf{N}$ est la matrice contenant les valeurs des fonctions rationnelles par morceaux :

$$\mathbf{N} = \begin{pmatrix} R_{0,\ldots,0}\left(u_1^{(1)},\ldots,u_1^{(N)}\right) & \cdots & R_{n_1,\ldots,n_N}\left(u_1^{(1)},\ldots,u_1^{(N)}\right) \\ \vdots & \ddots & \vdots \\ R_{0,\ldots,0}\left(u_{n_{TP}}^{(1)},\ldots,u_{n_{TP}}^{(N)}\right) & \cdots & R_{n_1,\ldots,n_N}\left(u_{n_{TP}}^{(1)},\ldots,u_{n_{TP}}^{(N)}\right) \end{pmatrix}, \tag{4.32}$$

$\mathbf{P}$ est la matrice contenant les coordonnées des points de contrôle "rangées" de la même manière que les fonctions rationnelles par morceaux (i.e. les indices $i_1,\ldots,i_N$ correspondent) :

$$\mathbf{P} = \begin{pmatrix} P_{0,\ldots,0}^{(1)} & \cdots & P_{0,\ldots,0}^{(M)} \\ \vdots & \ddots & \vdots \\ P_{n_1,\ldots,n_N}^{(1)} & \cdots & P_{n_1,\ldots,n_N}^{(M)} \end{pmatrix}, \tag{4.33}$$

et $\mathbf{Q}$ est la matrice contenant les coordonnées des points cibles :

$$\mathbf{Q} = \begin{pmatrix} Q_1^{(1)} & \cdots & Q_1^{(M)} \\ \vdots & \ddots & \vdots \\ Q_{n_{TP}}^{(1)} & \cdots & Q_{n_{TP}}^{(M)} \end{pmatrix}. \tag{4.34}$$

Notons que l'éq. (4.31) reste valable dans le cas de l'approximation mais que la matrice $\mathbf{N}$ est rectangulaire dans ce cas. Il est alors possible de déterminer la pseudo-inverse de cette matrice qui est équivalente à une approximation des moindres carrés.

Le calcul des coordonnées des points de contrôle, lorsque leur position dans l'espace paramétrique est explicite, met en jeu un système d'équations linéaires très similaire de la forme suivante :

$$\mathbf{N}^{(CP)}\mathbf{P} = \mathbf{Q}^{(NN)}, \tag{4.35}$$

où, cette fois-ci, la matrice à inverser $\mathbf{N}^{(CP)}$ regroupe les valeurs des fonctions rationnelles par morceaux aux coordonnées paramétriques des points de contrôle,

$$\mathbf{N}^{(CP)} = \begin{pmatrix} R_{0,\ldots,0}\left(u_0^{(1)},\ldots,u_0^{(N)}\right) & \cdots & R_{n_1,\ldots,n_N}\left(u_0^{(1)},\ldots,u_0^{(N)}\right) \\ \vdots & \ddots & \vdots \\ R_{0,\ldots,0}\left(u_{n_1}^{(1)},\ldots,u_{n_N}^{(N)}\right) & \cdots & R_{n_1,\ldots,n_N}\left(u_{n_1}^{(1)},\ldots,u_{n_N}^{(N)}\right) \end{pmatrix}, \tag{4.36}$$

avec $\mathbf{u}_{i_1,\ldots,i_N} = \left(u_{i_1}^{(1)},\ldots,u_{i_N}^{(N)}\right)$ les coordonnées dans l'espace paramétrique du point de contrôle $\mathbf{P}_{i_1,\ldots,i_N}$ [1]. Cette notion de coordonnées paramétriques des points de contrôle est utilisée par Turner [1] mais dans un soucis de clarté nous ne l'utiliserons pas dans le reste de ce manuscrit.

En effet, par définition, les points de contrôle ne possèdent pas de coordonnées paramétriques. En revanche ce sont les points cibles qui en possèdent. De plus, $\mathbf{u}_{i_1,\ldots,i_N}$ représente en réalité le vecteur des paramètres auxquels sont calculées les fonctions rationnelles $R_{i_1,\ldots,i_N}\left(u_{i_1}^{(1)},\ldots,u_{i_N}^{(N)}\right)$ qui permettent ensuite de déterminer les coordonnées des points de contrôle.

Avec cette formulation, il est possible que le vecteur des paramètres $\mathbf{u}_{i_1,\ldots,i_N}$, auxquels sont calculées les coordonnées d'un point de contrôle, ne possède pas de point cible correspondant (i.e. un point cible ayant été obtenu aux coordonnées paramétriques $\mathbf{u}_{i_1,\ldots,i_N}$). Pour cela, la matrice $\mathbf{Q}^{(NN)}$ est de la forme,

$$\mathbf{Q}^{(NN)} = \begin{pmatrix} \mathbf{Q}\left(\mathbf{u}_{NN}\left(\mathbf{u}_{0,\ldots,0}\right)\right) \\ \vdots \\ \mathbf{Q}\left(\mathbf{u}_{NN}\left(\mathbf{u}_{n_1,\ldots,n_N}\right)\right) \end{pmatrix}, \tag{4.37}$$

avec $\mathbf{u}_{NN}\left(\mathbf{u}_{i_1,\ldots,i_N}\right)$ les coordonnées du point cible le plus proche des coordonnées paramétriques $\mathbf{u}_{i_1,\ldots,i_N}$.

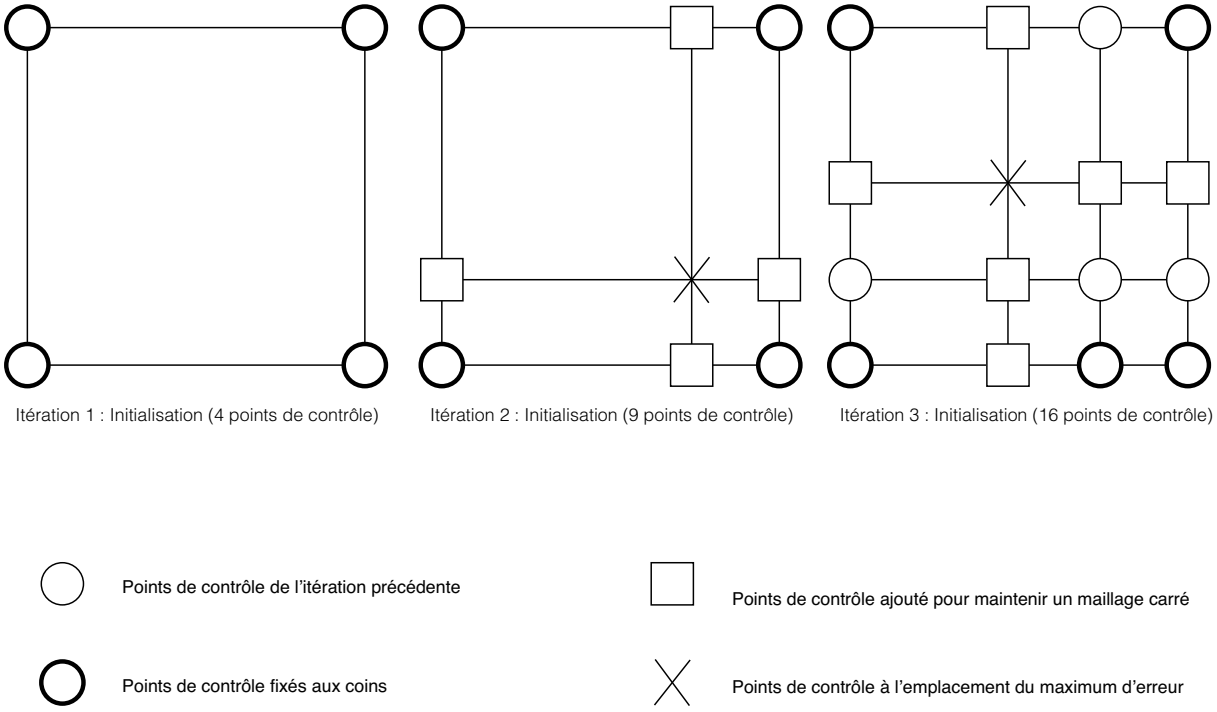


FIGURE 4.4 – Stratégie itérative d'ajout de points de contrôle de Turner [1]

Cette approche a plusieurs avantages, et notamment, comme le souligne Turner [1], elle s'affranchit de la nécessité d'une répartition régulière des points cibles. Elle permet également de diminuer la taille des matrices à stocker pour la résolution du système d'équations linéaires. En effet, nous verrons plus tard dans le chapitre que, dans le cas de l'approximation des moindres carrés, il est nécessaire de stocker une matrice ayant une dimension liée au nombre de points cibles et une dimension liée au nombre de points de contrôle. Avec cette approche, les dimensions

de la matrice sont seulement liées au nombre de points de contrôle qui est inférieur au nombre de points cibles ($n_{CP} \leq n_{TP}$). Nous n'utilisons cependant pas cette approche et nous verrons, plus tard dans le chapitre, qu'il existe une méthode plus efficace que l'inversion directe de la matrice $\mathbf{N}$, basée sur l'inversion de plusieurs matrices $\mathbf{N}_k$ de tailles inférieures à celle de $\mathbf{N}$.

L'obtention des coordonnées des points de contrôle étant explicitée, la stratégie de positionnement des points de contrôle de Turner [1] est donnée à la fig. 4.4. A la première itération, le degré des fonctions de base des NURBS est fixé à 1 (i.e. $p_k = 1$, $k = 1, ..., N$) et seulement deux points de contrôle sont affectés dans chaque direction (i.e. $n_k = 1$, $k = 1, ..., N$). Sur la fig. 4.4, qui présente le principe en dimension $N = 2$, cela se traduit par des points de contrôle placés aux quatre coins du domaine. A l'itération suivante, le degré des fonctions de forme est fixé à 2 et ne varie plus au cours du processus (i.e. pour toutes les autres itérations $p_k = 2$, $k = 1, ..., N$). Au cours de cette même itération, un point de contrôle est ajouté à l'endroit où l'erreur maximale d'approximation est calculée. Des points de contrôle, permettant de maintenir un maillage carré, sont également introduits et ne dépendent que de la positions du point de contrôle ajouté à l'endroit où l'erreur d'approximation est maximale, et des positions des points de contrôle des itérations précédentes. Notons qu'aux itérations 1 et 2, les *knot vectors* sont respectivement $\{0, 0, 1, 1\}$ et $\{0, 0, 0, 1, 1, 1\}$ dans toutes les directions de l'espace. Les différents *knot vectors*, pour les itérations suivantes, sont obtenus en fonction des points de contrôle ajoutés aux itérations précédentes. Supposons qu'un point de contrôle ait été ajouté dans la direction k à l'itération l. La coordonnée paramétrique de ce point de contrôle est $u_l^{(k)}$. A l'itération $l+1$, si des points sont ajoutés à la coordonnée paramétrique $u_{l+1}^{(k)}$, dans la direction k, alors le nouveau nœud ajouté au *knot vector* sera :

$$U_{l+1}^{(k)} = \frac{u_l^{(k)} + u_{l+1}^{(k)}}{2}, \tag{4.38}$$

où $U_{l+1}^{(k)}$ est le nouveau nœud introduit dans le *knot vector* $\mathbf{U}^{(k)}$. Ce choix s'apparente à réaliser des *knot vectors* uniformément repartis, ce qui n'est pas pertinent lorsque les points cibles ne sont pas uniformément répartis [2].

Cette méthode, proposée par Turner [1], et ayant montré son efficacité sur des problèmes de caractérisation inverse [30], présente des avantages indéniables dans le cadre de la réduction de modèle. En effet, la stratégie d'obtention des coordonnées des points de contrôle permet d'utiliser des techniques d'échantillonnage de l'espace n'utilisant pas un schéma régulier. En particulier, un point cible de l'espace peut être utilisé pour calculer la valeur de plusieurs points de contrôle. Elle présente cependant une limitation importante car, comme les techniques utilisées dans le cadre des courbes et surfaces NURBS, elle n'intègre dans les variables d'optimisation du métamodèle que le nombre de points de contrôle. De plus, un nombre non négligeable de paramètres doit être fixé par l'utilisateur. Même s'il existe des règles empiriques permettant de déterminer ces paramètres, aucune justification physique ne permet de valider ces règles. Notamment, l'utilisation de *knot vectors* uniformes est connue pour donner de moins bons résultats dans le cas où les points cibles ne sont pas également uniformément répartis. A cela s'ajoute le fait que les degrés des fonctions de base des NURBS sont fixés à 2, ne permettant pas d'obtenir les dérivées du métamodèle d'ordre supérieur ou égal à 2. Dans l'optique de surmonter ces limitations, la

méthode que nous proposons intègre tous les paramètres de l'hypersurface NURBS dans l'optimisation du métamodèle et ne s'appuie sur aucune hypothèse simplificatrice pour fixer la valeur de ces paramètres.

4.2.3 Fonction objectif et fonctions contraintes

Nous avons détaillé les différents paramètres à déterminer pour définir complètement une hypersurface NURBS. Ces paramètres correspondent aux variables d'optimisation. Nous allons maintenant définir la fonction objectif de ce problème. On pourrait être tenté, dans un premier temps, d'envisager une fonction objectif ayant pour but de minimiser l'erreur d'approximation. En effet, dans beaucoup de méthodes de réduction de modèle, la phase de calibration a pour objectif principal de diminuer l'erreur d'approximation du métamodèle réalisé. Supposons donc un problème de minimisation de l'erreur des moindres carrés, de la forme suivante :

$$\begin{array}{c} \min_{\mathbf{x}} \Phi(\mathbf{x}), \\ \Phi(\mathbf{x}) = \sum_{s_1=0}^{r_1} \cdots \sum_{s_N=0}^{r_N} \left(\mathbf{H}(\mathbf{u}_{s_1,\ldots,s_N}) - \mathbf{Q}_{s_1,\ldots,s_N}\right)^2, \end{array} \tag{4.39}$$

avec $\mathbf{u}_{s_1,\ldots,s_N} = \left(u_{s1}^{(1)}, \ldots, u_{sN}^{(N)}\right)$ le vecteur des paramètres d'entrée sans dimension correspondant au point cible $\mathbf{Q}_{s_1,\ldots,s_N}$.

Supposons dans un premier temps qu'il n'y ait pas de contraintes d'optimisation autre que les nombres maximales de points de contrôle dans chaque direction (i.e. $n_k \leq r_k$, $k = 1, \ldots, N$), contraintes nécessaires au calcul des coordonnées des points de contrôle. Le problème de type CNLPP est, dans ce cas, de la forme :

$$\begin{array}{c} \min_{\mathbf{x}} \Phi(\mathbf{x}) = \sum_{s_1=0}^{r_1} \cdots \sum_{s_N=0}^{r_N} \left(\mathbf{H}(\mathbf{u}_{s_1,\ldots,s_N}) - \mathbf{Q}_{s_1,\ldots,s_N}\right)^2, \\ \text{sujet à :} \\ \left\{ n_k \leq r_k, \quad k = 1, \ldots, N, \right. \end{array} \tag{4.40}$$

où $\mathbf{x}$ est le vecteur regroupant les variables d'optimisation.

Nous l'avons évoqué précédemment, les entités NURBS peuvent être utilisées en approximation mais également en interpolation. Il apparaît donc que toute solution du problème remplissant la condition $n_{CP} = n_{TP}$ est un minimum local de la fonction Φ (qui est convexe dans l'espace des points de contrôle mais pas dans celui des composantes des *knot vectors*). Il en existe une infinité puisqu'il existe une infinité de *knot vectors* $\mathbf{U}^{(k)}$ pour un même nombre de points de contrôle $n_k + 1$, $k = 1, \ldots, N$. Il semble donc évident que le problème (4.40) ne permet pas de déterminer les composantes des *knot vectors*. En effet, rien ne permet de différencier deux minima et le reste des paramètres (i.e. autre que n_k, $k = 1, \ldots, N$) doivent être déterminés par une autre méthode. De plus, dans le cadre de la réduction de modèle, certaines applications ont pour objectif de diminuer l'espace de stockage nécessaire, ce qui n'est clairement pas possible avec la formulation du problème (4.40). Dans cet optique, il est possible d'imaginer des contraintes supplémentaires sur le nombre maximal de points de contrôle $n_{CP} \leq n_{CP,max}$. L'utilisateur pourrait le paramétrer en fonction d'un gain minimal attendu sur le nombre de données à stocker de la forme suivante :

$$\tau = \frac{n_{CP,max}}{n_{TP}}. \tag{4.41}$$

Une telle approche nécessite une certaine expertise de l'utilisateur. En effet, il n'est pas possible de garantir un seuil d'erreur maximal avec une telle formulation du problème. Il est donc obligatoire pour l'utilisateur de "juger", à l'avance, de l'erreur commise en fonction du nombre de points. Ce n'est bien sûr pas l'objectif de notre méthode, l'algorithme génétique utilisé ayant pour vocation principale de trouver le nombre minimal de points de contrôle nécessaire à garantir un certain seuil d'erreur.

La différence principale entre l'approche proposée ici et les méthodes itératives décrites précédemment, repose sur le fait que la recherche de la solution ne se focalise pas uniquement sur le nombre de points de contrôle pour garantir un certain seuil d'erreur d'approximation (les autres paramètres de l'hypersurface NURBS étant fixés par des règles empiriques non justifiées). En effet, notre approche permet également de minimiser le temps de restitution du métamodèle en incluant notamment les degrés p_k comme variables d'optimisation. La fonction objectif de notre problème d'optimisation représente donc cette double condition de nombre de points contrôle et de degrés à minimiser. Elle est de la forme suivante :

$$\phi(\mathbf{x}) = a \times \frac{1}{N}\sum_{k=1}^{N}\frac{n_k}{n_k^{max}} + (1-a) \times \frac{1}{N}\sum_{k=1}^{N}\frac{p_k}{p_k^{max}}, \quad 0 < a < 1, \tag{4.42}$$

avec n_k^{max} et p_k^{max} les nombre de points de contrôle et degré maximum dans la direction $k = 1, ..., N$, respectivement, et a un paramètre de pondération permettant de favoriser l'optimisation du temps d'évaluation par rapport au nombre de points de contrôle et *vice versa*.

Les fonctions de base des NURBS sont définies récursivement et l'augmentation du degré augmente le nombre de récursions nécessaires à leur calcul. En effet, l'éq. (3.4) du chapitre précédent mets en évidence le fait que la fonction de base de degré p_k, $N_{i_k,p_k}(u^{(k)})$, dépend des valeurs des fonctions de base des degrés inférieurs, jusqu'au degré 0. Pour une fonction de degrés p_k, il y a donc $p_k + 1$ récursions nécessaires pour la calculer. De plus, le nombre de fonctions de base non-nulles pour un point de contrôle est $p_k + 1$ dans la direction $k = 1, ..., N$. Pour ces deux raisons, des degrés élevés augmentent le nombre de fonctions de base à calculer, pour un même point de contrôle, ainsi que le temps de calcul de chacune de ces fonctions de base. La nécessité de ne pas surestimer ces paramètres est donc importante. Le paramètre a introduit dans l'éq. (4.42) permet d'affecter plus d'importance à la minimisation du nombre de points de contrôle ($a > 0.5$) ou à la minimisation des degrés ($a < 0.5$). Dans le cas où $a = 0.5$, l'importance est égale pour les deux.

Cette fonction objectif ne fait pas intervenir l'erreur d'approximation. En effet, le seuil d'erreur d'approximation maximale est, quant à lui, garanti par une fonction contrainte de la forme :

$$\epsilon_j^{(max)}(\mathbf{x}) \leq \epsilon_{j,max}^{(max)}, \quad j = 1, ..., M, \tag{4.43}$$

où $\epsilon_j^{(max)}(\mathbf{x})$ est l'erreur maximale d'approximation sur la j^{e} coordonnée de l'hypersurface (i.e. la j^{e} sortie du métamodèle) générée à partir des variables d'optimisation $\mathbf{x}$ et $\epsilon_{j,max}^{(max)}$ est le seuil d'erreur maximale fixé par l'utilisateur sur cette même sortie. Il y a donc autant de contraintes que de sorties du métamodèle. Il est tout à fait envisageable d'utiliser une erreur unique $\epsilon^{(max)}(\mathbf{x})$, basée sur la distance euclidienne dans l'espace des sorties du métamodèle par exemple. Cependant,

notre choix est déterminé par le fait que les coordonnées des points cibles de la base de données $Q^{(j)}_{s_1,...,s_N}$ peuvent avoir été obtenues par des moyens différents avec des précisions non similaires. Les différents seuils $\epsilon^{(max)}_{j,max}$ peuvent en particulier être fixés à partir de l'erreur commise par les moyens d'obtention des points de la base de données. Dans le cas de données expérimentales, par exemple, la précision des capteurs utilisés pour obtenir la sortie j peut fixer la valeur du seuil d'erreur $\epsilon^{(max)}_{j,max}$. En effet, il n'est pas utile d'obtenir un métamodèle dont l'erreur d'approximation est très inférieur à l'erreur de mesure ayant permis de l'obtenir. Dans notre cas, nous avons choisi une erreur maximale de la forme :

$$\epsilon^{(max)}_j = \max_{\mathbf{u}} \left(\frac{|H^{(j)}(\mathbf{u}) - Q^{(j)}|}{Q^{(j)}_{max} - Q^{(j)}_{min}} \right), \quad j = 1, ..., M, \tag{4.44}$$

avec $H^{(j)}(\mathbf{u})$ la j^{e} coordonnée de l'hypersurface NURBS (i.e. la j^{e} sortie du métamodèle) au point $\mathbf{u} = \left(u^{(1)}_{s_1}, ..., u^{(N)}_{s_N}\right)$, $s_k = 0, ..., r_k$, $Q^{(j)}$ le point cible correspondant et $Q^{(j)}_{max}$ (resp. $Q^{(j)}_{min}$) la valeur maximale (resp. minimale) de la sortie $Q^{(j)}$. Ces erreurs sont données sans dimensions et non signées. Notons qu'un autre choix de calcul de l'erreur peut être envisagé par l'utilisateur, notamment pour obtenir une erreur signée (i.e. dépassement ou non de la valeur à estimer). Dans le cas de l'approximation de contraintes par exemple, il peut être préférable d'avoir une légère surestimation de la part du métamodèle plutôt qu'une sous-estimation.

De même que pour les méthodes itératives, qui peuvent utiliser plusieurs critères de convergence pour trouver le nombre de points de contrôle, il est possible d'utiliser plusieurs contraintes afin de trouver les paramètres optimaux de l'hypersurface NURBS. Nous utilisons notamment des contraintes sur l'erreur moyenne d'approximation de la forme suivante :

$$\epsilon^{(moy)}_j(\mathbf{x}) \leq \epsilon^{(moy)}_{j,max}, \quad j = 1, ..., M, \tag{4.45}$$

où $\epsilon^{(moy)}_j(\mathbf{x})$ est l'erreur d'approximation moyenne de l'hypersurface NURBS générée à partir des variables d'optimisation $\mathbf{x}$ et $\epsilon^{(moy)}_{j,max}$ est le seuil d'erreur moyenne maximale fixé par l'utilisateur sur la sortie $j = 1, ..., M$. Ces contraintes permettent de vérifier que le métamodèle obtenu est "globalement" une bonne approximation des points cibles de la sortie $j = 1, ..., M$. Le terme d'erreur moyenne pouvant être ambigu, nous précisons l'erreur utilisée dans nos applications. Il s'agit d'une erreur relative sans dimension, non signée, de la forme suivante :

$$\epsilon^{(moy)}_j = \frac{1}{n_{TP}} \sum_{s_1=0}^{r_1} \cdots \sum_{s_N=0}^{r_N} \frac{|H^{(j)}(\mathbf{u}) - Q^{(j)}|}{Q^{(j)}_{max} - Q^{(j)}_{min}}, \quad j = 1, ..., M, \tag{4.46}$$

où $H^{(j)}(\mathbf{u})$ est la j^{e} coordonnées de l'hypersurface NURBS (i.e. la j^{e} sortie du métamodèle) au point $\mathbf{u} = \left(u^{(1)}_{s_1}, ..., u^{(N)}_{s_N}\right)$, $s_k = 0, ..., r_k$, $Q^{(j)}$ est le point cible correspondant et $Q^{(j)}_{max}$ (resp. $Q^{(j)}_{min}$) est la valeur maximale (resp. minimale) de la sortie $Q^{(j)}$. Notons que le coefficient de corrélation de Pearson pourrait également être utilisé, comme c'est le cas dans la méthode de Turner [1].

4.2.4 Formulation du CNLPP

Les variables d'optimisation, la fonction objectif ainsi que les fonctions contraintes étant définies, le problème CNLPP d'*hypersurface fitting* peut être posé de la façon suivante :

$$\min_{\mathbf{x}} \phi(\mathbf{x}) = a \times \frac{1}{N} \sum_{k=1}^{N} \frac{n_k}{n_k^{max}} + (1-a) \times \frac{1}{N} \sum_{k=1}^{N} \frac{p_k}{p_k^{max}}, \quad 0 < a < 1,$$

$$\text{sujet à :}$$

$$\begin{cases} \epsilon_j^{(max)}(\mathbf{x}) \leq \epsilon_{j,max}^{(max)}, \quad j = 1, ..., M, \\ \epsilon_j^{(moy)}(\mathbf{x}) \leq \epsilon_{j,max}^{(moy)}, \quad j = 1, ..., M, \\ 0 < U_{l_k}^{(k)} < 1, \quad l_k = p_k + 1, ..., m_k - p_k + 1, \quad k = 1, ..., N, \\ U_{l_k}^{(k)} \leq U_{l_k+1}^{(k)}, \quad l_k = p_k + 1, ..., m_k - p_k, \quad k = 1, ..., N, \\ \omega_{i_1,...,i_N} \geq 0, \quad i_k = 0, ..., n_k \\ n_k - p_k \geq 0, \quad k = 1, ..., N \\ n_{CP} \leq n_{TP}, \end{cases} \tag{4.47}$$

où $\mathbf{x}$ est le vecteur contenant les variables d'optimisation, i.e. les N degrés p_k, les N tailles des *knot vectors* $m_k + 1$ et les $m_k - 2p_k - 1$ valeurs non prédéfinies des *knot vectors* ainsi que les $(n_1 + 1) \times \cdots \times (n_N + 1)$ poids $\omega_{i_1,...,i_N}$, $i_k = 0, \ldots, n_k$. Le vecteur $\mathbf{x}$ regroupe les variables dans l'ordre suivant :

$$\begin{aligned} \mathbf{x} = \quad & \Big(p_1, \ldots, p_N, m_1, \ldots, m_N, U_{p_1+1}^{(1)}, \ldots, U_{m_1-p_1-1}^{(1)}, \ldots, U_{p_N+1}^{(N)}, \ldots, U_{m_N-p_N-1}^{(N)}, \\ & \omega_{0,...,0}, \ldots, \omega_{n_1,...,n_N}\Big). \end{aligned} \tag{4.48}$$

Comme évoqué dans le paragraphe précédent, la fonction objectif $\phi(\mathbf{x})$ est paramétrable par l'utilisateur, au travers du paramètre a, permettant d'accentuer la minimisation des données nécessaires à la restitution du métamodèle ($a > 0.5$) ou le temps d'évaluation de celui-ci ($a < 0.5$). Même s'il est théoriquement possible d'affecter une valeur nulle ou unitaire à a, n'accordant ainsi d'importance qu'à l'un ou l'autre des objectifs d'optimisation, il n'est pas recommandé de le faire. En effet, le problème serait alors mal posé puisque deux solutions, respectant les contraintes, peuvent ne pas être différenciées. Dans le cas où $a = 0$ (i.e. minimisation des degrés seulement), deux solutions respectant les contraintes et possédant les mêmes degrés, mais un nombre total de points de contrôle différent, seront équivalentes du point de vue de la fonction objectif alors que du point de vue de l'utilisateur, la solution ayant moins de points de contrôle est préférable à l'autre. A l'inverse, dans le cas où $a = 1$ (minimisation du nombre de points de contrôle seulement), ce sont deux solutions respectant les contraintes et possédant le même nombre de points de contrôle, mais des degrés différents, qui seront équivalentes du point de vue de la fonction objectif alors que pour l'utilisateur, la solution ayant des degrés plus faibles est préférable. Dans les deux cas, la fonction objectif n'est plus en mesure de différencier correctement toutes les hypersurfaces NURBS générées à partir des variables d'optimisation $\mathbf{x}$. Cela peut

conduire à un comportement singulier de l'algorithme de résolution. Dans la pratique, il est préférable de conserver un paramètre $a \in [0.1, 0.9]$.

Les erreurs maximales d'approximation des j sorties du métamodèle $\epsilon_{j,max}^{(max)}$ doivent, elles aussi, être paramétrées par l'utilisateur. Notons cependant que, même s'il faut paramétrer M valeurs, nous ne considérons pas ces paramètres comme des paramètres d'entrée de la méthode que nous proposons. En effet, quelle que soit la méthode de réduction employée, il est toujours nécessaire de fixer l'erreur maximale d'approximation tolérée. Ces valeurs correspondent à la précision d'approximation du métamodèle et sont donc, normalement, connues à l'avance par l'utilisateur. Nous avons vu, de plus, qu'il était possible de régler ces paramètres en fonction des moyens d'obtention qui ont permis de générer la base de données des points cibles. Le choix de ces paramètres ne repose, en aucun cas, sur une quelconque expertise du formalisme des hypersurfaces NURBS ou des algorithmes de résolution des CNLPP.

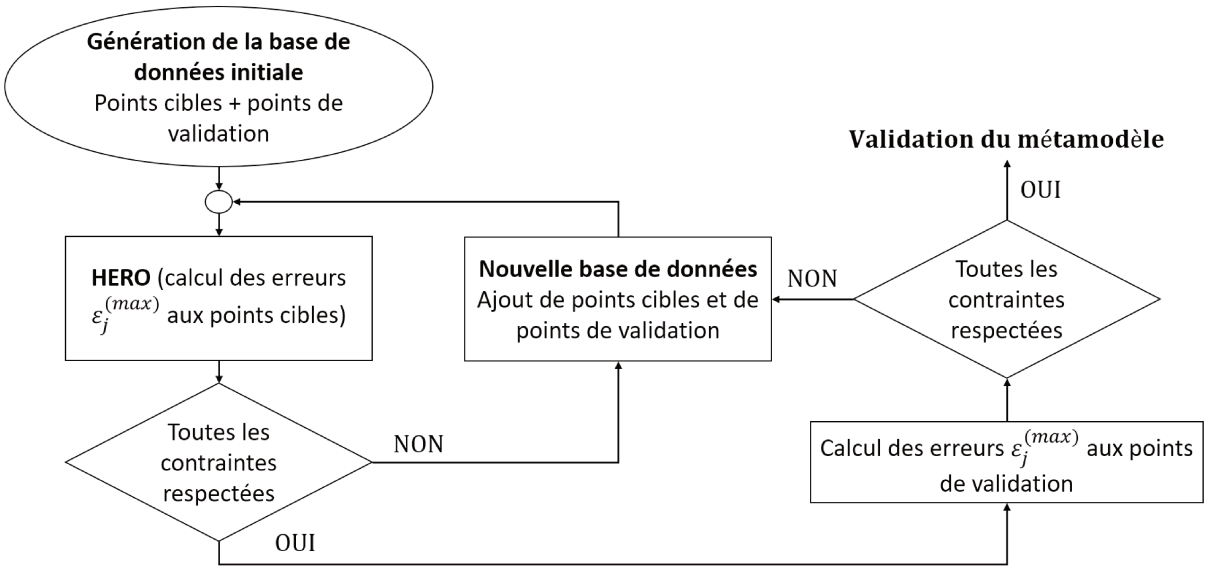


FIGURE 4.5 – Stratégie de validation du métamodèle pour une utilisation en approximation

Dans la phase de recherche des paramètres optimaux de l'hypersurface, les erreurs maximales d'approximation $\epsilon_j^{(max)}$, $j = 1, ..., M$, sont calculées sur l'ensemble des points cibles utilisés pour calculer les coordonnées des points de contrôle de l'hypersurface. En effet, il est nécessaire de s'assurer que le métamodèle réalise une bonne approximation des points cibles avant de le valider sur un ensemble de points n'ayant pas servi au calcul des coordonnées des points de contrôle. Dans le cas où notre méthode serait utilisée en interpolation (i.e. $n_k = r_k$, $k = 1, ..., N$), l'erreur devrait être calculée directement sur un ensemble de points de validation puisque l'hypersurface passerait exactement par les points cibles (i.e. erreur d'approximation nulle en chaque point cible). Les fig. 4.5 et fig. 4.6 présentent les stratégies de validation du métamodèle dans le cas de l'approximation et de l'interpolation, respectivement.

Utilisé en approximation (i.e. $n_{CP} < n_{TP}$), la stratégie de validation du métamodèle passe

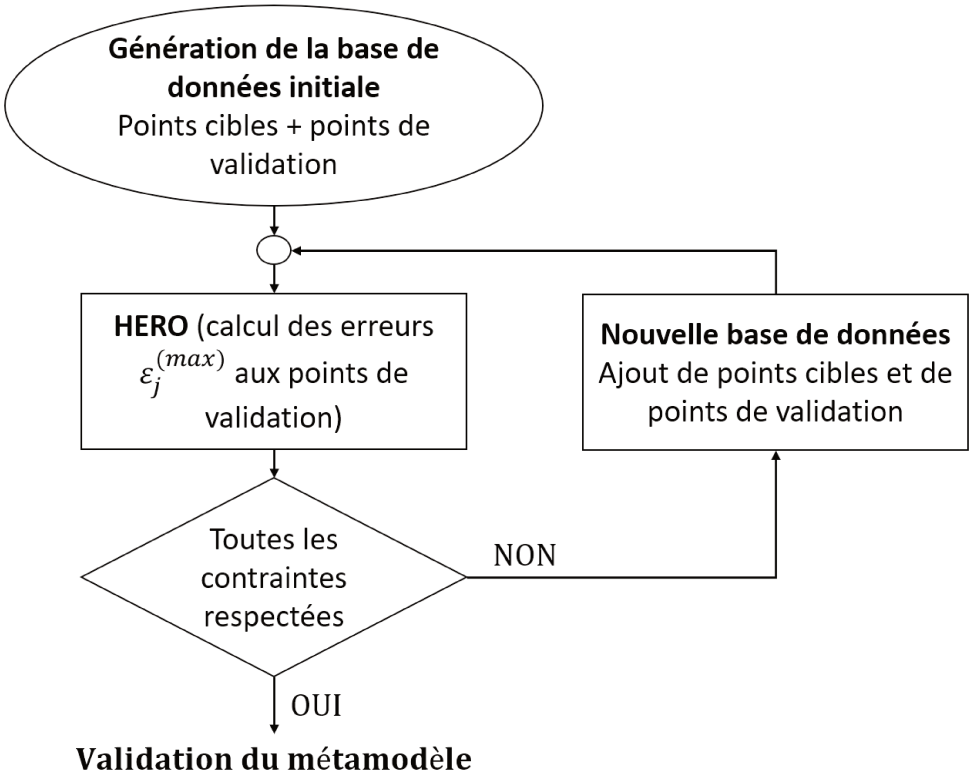


FIGURE 4.6 – Stratégie de validation du métamodèle pour une utilisation en interpolation

par deux étapes (cf. fig. 4.5). Dans un premier temps, l'optimisation du métamodèle est réalisée en calculant les erreurs maximales d'approximation sur les points cibles de la base de données. Si, quels que soient les paramètres de l'hypersurface, une, ou plusieurs, erreurs maximales d'approximation $\epsilon_j^{(max)}$ dépassent les seuils d'erreur maximale autorisés par l'utilisateur $\epsilon_{j,max}^{(max)}$ (i.e. l'algorithme ne parvient pas à trouver de solution faisable), l'interpolation est utilisée (i.e. le nombre de points de contrôle est égal au nombre de points cibles). Notons toutefois que, dans ce cas, les nombres de points de contrôle n_k+1 étant fixés, seuls les degrés p_k des fonctions de base, les valeurs internes $U_{l_k}^{(k)}$, $l_k = p_k+1, ..., m_k-p_k+1$, $k = 1, ..., N$, des *knot verctors* et les poids $\omega_{i_1,...,i_N}$, $i_k = 0, ..., n_k$, font partis des variables d'optimisation. Ce problème reste cependant de dimension variable puisque les nombres de valeurs internes des *knot vectors* sont liés aux degrés p_k. De plus, l'interpolation ne permettant pas de calculer l'erreur d'approximation au niveau des points cibles, les erreurs maximales d'approximation sont calculées directement sur les points de validation pendant la phase d'optimisation des paramètres de l'hypersurface NURBS (cf. fig. 4.6).

Si l'interpolation ne permet toujours pas d'obtenir un modèle réduit avec la précision souhaitée, il est nécessaire de générer plus de points dans la base de données de points cibles. En effet, un manque de points cibles peut conduire, quelle que soit la méthode de réduction de modèle em-

ployée, à une mauvaise approximation du phénomène que l'on cherche à modéliser. L'obtention des points cibles pouvant être coûteuse, il pourrait être intéressant de choisir les points cibles à partir de méthodes d'échantillonnage telles que l'hypercube latin ou le quasi-Monte-Carlo par exemple. Cependant, pour des raisons de calcul des coordonnées des points de contrôle, il est préférable d'avoir un répartition ordonnée des points cibles, ce qui limite les choix pour les méthodes d'échantillonnage. Il existe toutefois des moyens pour contourner cette limitation, comme par exemple l'utilisation du *plus proche voisin* comme c'est le cas dans [1]. De nouveaux points cibles doivent être ajoutés jusqu'à ce que le métamodèle obtenu satisfasse les contraintes de précision fixées par l'utilisateur. Dans un second temps, lorsque la base de données des points cibles est assez conséquente pour permettre de trouver des solutions respectant les contraintes, les erreurs maximales d'approximation du métamodèle obtenu sont calculées sur un ensemble de points de validation différent des points cibles (i.e. n'ayant pas servi au calcul des coordonnées des points de contrôle de l'hypersurface). Si, après cette étape, les contraintes d'erreur maximale d'approximation fixées par l'utilisateur sont respectées, le métamodèle est validé et peut être utilisé.

Les M seuils d'erreur moyenne d'approximation $\epsilon_{j,max}^{(moy)}$ sont également à paramétrer. Contrairement aux seuils d'erreur maximale d'approximation $\epsilon_{j,max}^{(max)}$, facilement paramétrables, ces valeurs peuvent ne pas être connues à l'avance par l'utilisateur. Ces contraintes revêtent cependant un caractère "optionnel". En effet, les erreurs $\epsilon_j^{\mathbf{u}},\ j = 1, ..., M$, calculées en un point $\mathbf{u}$, sont inférieures aux erreurs maximales $\epsilon_j^{(max)}$. De ce fait, la précision du métamodèle est fixée par les erreurs maximales d'approximation. La plupart du temps, cette information peut être suffisante pour l'utilisateur. Cependant, les erreurs moyennes d'approximation $\epsilon_j^{(moy)}$ peuvent fournir une indication supplémentaire sur la "qualité globale" d'approximation de la solution. En effet, si les erreurs moyennes d'approximation $\epsilon_j^{(moy)}$ et les erreurs maximales d'approximation $\epsilon_j^{(max)}$ sont du même ordre de grandeur, cela signifie que les erreurs d'approximation commises par le métamodèle sont du même ordre de grandeur pour toute l'hypersurface. À l'inverse, si les erreurs moyennes d'approximation $\epsilon_j^{(moy)}$ sont d'un ordre de grandeur inférieur à celui des erreurs maximales $\epsilon_j^{(max)}$, alors les erreurs d'approximation du métamodèle sont "globalement meilleures" que sa précision minimale (fixée par les erreurs maximales du métamodèle). Dans la pratique, les erreurs moyennes $\epsilon_j^{(moy)}$ sont souvent inférieures, d'au moins, un ordre de grandeur par rapport aux erreurs maximales d'approximation $\epsilon_j^{(max)}$. Cette "règle" peut être appliquée pour fixer les M valeurs des $\epsilon_{j,max}^{(moy)}$ en fonction des M valeurs $\epsilon_{j,max}^{(max)}$.

4.3 Algorithmes de calcul des coordonées des points de contrôle

4.3.1 Formulation générale du système d'équations linéaires

Comme nous l'avons dit précédemment, les coordonnées des points de contrôle de l'hypersurface NURBS ne font pas partie des variables d'optimisation du CNLPP (4.47). En effet, elles sont obtenues par la résolution d'un système d'équations linéaires et nécessitent que tous les autres paramètres de l'hypersurface soient connus. Nous allons, dans un premier temps, introduire le

système d'équations linéaires permettant de calculer les coordonnées des points de contrôle d'une courbe, ou d'une hyper-courbe, NURBS (i.e. $N = 1$, $M > 1$). On suppose pour cela :

- Approximer $r+1$ points cibles $\mathbf{Q}_s = \left(Q_s^{(1)}, \ldots, Q_s^{(M)}\right) \in \mathbb{R}^M$, $s = 0, ..., r$,
- à la coordonnée paramétrique u_s, $s = 0, ..., r$,
- par une courbe NURBS $\mathbf{C}(u) : \mathbb{R} \to \mathbb{R}^M$,
- composée de $n+1$ points de contrôle,
- dont les fonctions de base sont de degré p,
- dont le *knot vector* $\mathbf{U}$ est connu,
- et dont les poids ω_i affectés aux points de contrôle $\mathbf{P}_i = \left(P_i^{(1)}, ..., P_i^{(M)}\right)$ sont connus.

On pose :

$$\begin{aligned} g_c &= \textstyle\sum_{s=0}^{r} \left(\mathbf{Q}_s - \mathbf{C}(u_s)\right)^2 \\ &= \textstyle\sum_{s=0}^{r} \left(\mathbf{Q}_s - \frac{\sum_{i=0}^{n} N_{i,p}(u_s)\omega_i \mathbf{P}_i}{\sum_{i=0}^{n} N_{i,p}(u_s)\omega_i}\right)^2 \\ &= \textstyle\sum_{s=0}^{r} \left(\mathbf{Q}_s - \sum_{i=0}^{n} R_{i,p}(u_s)\mathbf{P}_i\right)^2. \end{aligned} \tag{4.49}$$

La fonction g_c est une fonction à valeur scalaire des $n+1$ variables $\mathbf{P}_0, ..., \mathbf{P}_n$ et $R_{i,p}(u_s)$ est la fonction de base rationnelle par morceaux introduite dans l'éq. (3.2). Cette fonction nous permet d'appliquer la technique classique d'approximation par les moindres carrées (*least square fitting*) [2]. Le minimum de g_c est obtenu en imposant le gradient de g_c par rapport à $\mathbf{P}_l$ égal à 0. La l^{e} dérivée est donnée par :

$$\frac{\partial g_c}{\partial \mathbf{P}_l} = \sum_{s=0}^{r} \left(-2R_{l,p}(u_s)\mathbf{Q}_s + 2R_{l,p}(u_s) \sum_{i=0}^{n} R_{i,p}(u_s)\mathbf{P}_i\right), \tag{4.50}$$

ce qui implique :

$$\sum_{i=0}^{n} \left(\sum_{s=0}^{r} R_{l,p}(u_s)R_{i,p}(u_s)\mathbf{P}_i\right) = \sum_{s=0}^{r} R_{l,p}(u_s)\mathbf{Q}_s. \tag{4.51}$$

L'éq. (4.51) est une équation linéaire des inconnues $\mathbf{P}_0, ..., \mathbf{P}_n$. L'ensemble de toutes ces conditions ($l = 0, ..., n$) forment un système de $n+1$ équations linéaires avec $n+1$ inconnues :

$$\left(\mathbf{N}^T\mathbf{N}\right)\mathbf{P} = \mathbf{N}^T\mathbf{Q}, \tag{4.52}$$

avec $\mathbf{N}$ la matrice dont les $r+1$ lignes contiennent les fonctions de base rationnelles par morceaux des $n+1$ points de contrôle,

$$\mathbf{N} = \begin{pmatrix} R_{0,p}(u_0) & \cdots & R_{n,p}(u_0) \\ \vdots & \ddots & \vdots \\ R_{0,p}(u_r) & \cdots & R_{n,p}(u_r) \end{pmatrix}, \tag{4.53}$$

$\mathbf{P}$ est la matrice contenant les $M \times (n+1)$ coordonnées $P_i^{(j)}$ des points de contrôle,

$$\mathbf{P} = \begin{pmatrix} P_0^{(1)} & \cdots & P_0^{(M)} \\ \vdots & \ddots & \vdots \\ P_n^{(1)} & \cdots & P_n^{(M)} \end{pmatrix}, \tag{4.54}$$

et $\mathbf{Q}$ est la matrice contenant les $M \times (r+1)$ coordonnées $Q_s^{(j)}$ des points cibles,

$$\mathbf{Q} = \begin{pmatrix} Q_0^{(1)} & \cdots & Q_0^{(M)} \\ \vdots & \ddots & \vdots \\ Q_r^{(1)} & \cdots & Q_r^{(M)} \end{pmatrix}. \tag{4.55}$$

Comme on peut le voir dans l'éq. (4.52), il est nécessaire de calculer la matrice $\mathbf{N}$, de taille $(r+1)\times(n+1)$, afin de calculer les coordonnées des points de contrôle. Il est toutefois important de constater que cette matrice est calculée une seule fois pour les M seconds membres de $\mathbf{Q}$ et les M inconnues (i.e. les M coordonnées des points de contrôle). La propriété de support local des entités NURBS permet également de remarquer que seules $p+1$ valeurs, au plus, sont non-nulles sur chacune des lignes de la matrice $\mathbf{N}$ et il peut être pertinent de la stocker de manière à tirer parti de cette spécificité. Il est intéressant de noter le cas particulier où tous les poids $\omega_i = 1$ sont fixés à 1 (i.e. courbes et hyper-courbes B-Spline). Dans ce cas précis, l'éq. (4.52) reste valable avec :

$$\mathbf{N} = \begin{pmatrix} R_{0,p}(u_0) & \cdots & R_{n,p}(u_0) \\ \vdots & \ddots & \vdots \\ R_{0,p}(u_r) & \cdots & R_{n,p}(u_r) \end{pmatrix} = \begin{pmatrix} N_{0,p}(u_0) & \cdots & N_{n,p}(u_0) \\ \vdots & \ddots & \vdots \\ N_{0,p}(u_r) & \cdots & N_{n,p}(u_r) \end{pmatrix}. \tag{4.56}$$

Le problème consistant à déterminer à la fois les poids et les coordonnées des points de contrôle étant non-linéaire, il n'est pas rare dans les domaines de *curve* et *surface fitting* d'utiliser l'approximation B-Spline comme point de départ de l'optimisation des poids (i.e. les coordonnées des points de contrôle sont fixés par une approximation B-Spline et seules les valeurs des poids sont optimisées) comme c'est le cas dans [13, 225].

Nous l'avons énoncé précédemment, les entités NURBS sont également capables de réaliser l'interpolation de points cibles. Dans ce cas particulier, on suppose :

- Interpoler $r+1 = n+1$ points cibles $\mathbf{Q}_s = \left(Q_s^{(1)}, \ldots, Q_s^{(M)}\right) \in \mathbb{R}^M$, $s = 0, ..., n$,
- à la coordonnée paramétrique u_s, $s = 0, ..., n$,
- par une courbe NURBS $\mathbf{C}(u) : \mathbb{R} \to \mathbb{R}^M$,
- composée de $n+1$ points de contrôle,
- dont les fonctions de base sont de degré p,
- dont le *knot vector* $\mathbf{U}$ est connu,
- et dont les poids ω_i affectés aux points de contrôle $\mathbf{P}_i = \left(P_i^{(1)}, ..., P_i^{(M)}\right)$ sont connus.

Comme nous souhaitons interpoler les points cibles, il est possible d'écrire

$$\begin{array}{rcl} \mathbf{Q}_s & = & \mathbf{C}(u_s) \\ & = & \sum_{i=0}^{n} R_{i,p}(u_s)\mathbf{P}_i. \end{array} \quad . \tag{4.57}$$

Ces $n+1$ équations peuvent être mises sous la forme :

$$\mathbf{NP} = \mathbf{Q}, \tag{4.58}$$

avec $\mathbf{N}$, $\mathbf{P}$ et $\mathbf{Q}$ les matrices précédemment définies. Notons toutefois, qu'en interpolation, la matrice $\mathbf{N}$ est carrée, de dimension $n+1$ (ou $r+1$).

Il est possible de généraliser ces expressions pour une dimension N quelconque. Pour cela, on suppose :

- Approximer n_{TP} points cibles $\mathbf{Q}_s = \left(Q_s^{(1)}, \ldots, Q_s^{(M)}\right) \in \mathbb{R}^M$, $s = 1, ..., n_{TP}$,
- aux coordonnées paramétriques $\mathbf{u}_s = \left(u_s^{(1)}, \ldots, u_s^{(N)}\right)$,
- par une hypersurface NURBS $\mathbf{H}(\mathbf{u}) : \mathbb{R}^N \to \mathbb{R}^M$, $\mathbf{u} = \left(u^{(1)}, ..., u^{(N)}\right)$,
- composée de $n_k + 1$ points de contrôle dans la direction $k = 1, ..., N$,
- dont les fonctions de base sont de degrés p_k dans la direction $k = 1, ..., N$,
- dont les *knot vectors* $\mathbf{U}^{(k)}$ sont connus,
- et dont les poids $\omega_{i_1,...,i_N}$ affectés aux points de contrôle $\mathbf{P}_{i_1,...,i_N} = \left(P^{(1)}_{i_1,...,i_N}, ..., P^{(M)}_{i_1,...,i_N}\right)$ sont connus.

On pose cette fois-ci :

$$\begin{array}{rcl} g_h & = & \sum_{s=1}^{n_{TP}} \left(\mathbf{Q}_s - \mathbf{H}(\mathbf{u}_s)\right)^2 \\ & = & \sum_{s=1}^{n_{TP}} \left(\mathbf{Q}_s - \sum_{i_1=0}^{n_1} \cdots \sum_{i_N=0}^{n_N} R_{i_1,...,i_N}(u_s^{(1)}, \ldots, u_s^{(N)})\mathbf{P}_{i_1,...,i_N}\right)^2. \end{array} \tag{4.59}$$

La fonction g_h est une fonction à valeur scalaire des $(n_1+1) \times \cdots \times (n_N+1)$ variables $\mathbf{P}_{0,...,0}$, ..., $\mathbf{P}_{n_1,...,n_N}$ et $R_{i_1,...,i_N}(u_s^{(1)}, \ldots, u_s^{(N)})$ est la fonction rationnelle par morceaux introduite dans l'éq. (3.23). De même que dans le cas précédent, cette fonction g_h nous permet d'appliquer la technique classique d'approximation par les moindres carrées. Le minimum de g_h est obtenu en imposant le gradient de g_h par rapport à $\mathbf{P}_{l_1,...,l_N}$, $l_k = 0, ..., n_k$, égal à 0. Cette dérivée est donnée par :

$$\begin{array}{rcl} \dfrac{\partial g_h}{\partial \mathbf{P}_{l_1,...,l_N}} & = & \sum_{s=1}^{n_{TP}} \left(-2R_{l_1,...,l_N}\left(u_s^{(1)}, \ldots, u_s^{(N)}\right)\mathbf{Q}_s + 2R_{l_1,...,l_N}\left(u_s^{(1)}, \ldots, u_s^{(N)}\right)\right. \\ & & \left. \times \sum_{i_1=0}^{n_1} \cdots \sum_{i_N=0}^{n_N} R_{i_1,...,i_N}\left(u_s^{(1)}, \ldots, u_s^{(N)}\right)\mathbf{P}_{i_1,...,i_N}\right), \end{array} \tag{4.60}$$

Ce qui implique :

$$\begin{array}{c} \sum_{i_1=0}^{n_1} \cdots \sum_{i_N=0}^{n_N} \left(\sum_{s=1}^{n_{TP}} R_{l_1,...,l_N}\left(u_s^{(1)}, \ldots, u_s^{(N)}\right) \times R_{i_1,...,i_N}\left(u_s^{(1)}, \ldots, u_s^{(N)}\right)\right)\mathbf{P}_{i_1,...,i_N} \\ = \sum_{s=1}^{n_{TP}} R_{l_1,...,l_N}\left(u_s^{(1)}, \ldots, u_s^{(N)}\right)\mathbf{Q}_s. \end{array} \tag{4.61}$$

L'éq. (4.61) est une équation linéaire des inconnues $\mathbf{P}_{i_1,...,i_N}$, $i_k = 0, ..., n_k$. L'ensemble de toutes ces conditions ($l_k = 0, ..., n_k$) forment un système de $(n_1+1) \times \cdots \times (n_N+1)$ équations linéaires avec $(n_1+1) \times \cdots \times (n_N+1)$ inconnues qui peut être mis sous la forme de l'éq. (4.52) avec, dans ce cas, la matrice $\mathbf{N}$, dont les n_{TP} lignes contiennent les $n_{CP} = \prod_{k=1}^{N} (n_k+1)$ fonctions de base rationnelles des points de contrôle, de la forme suivante :

$$\mathbf{N} = \begin{pmatrix} R_{0,...,0}\left(u_1^{(1)}, \ldots, u_1^{(N)}\right) & \cdots & R_{n_1,...,n_N}\left(u_1^{(1)}, \ldots, u_1^{(N)}\right) \\ \vdots & \ddots & \vdots \\ R_{0,...,0}\left(u_{n_{TP}}^{(1)}, \ldots, u_{n_{TP}}^{(N)}\right) & \cdots & R_{n_1,...,n_N}\left(u_{n_{TP}}^{(1)}, \ldots, u_{n_{TP}}^{(N)}\right) \end{pmatrix}, \tag{4.62}$$

$\mathbf{P}$ est la matrice contenant les $M \times n_{CP}$ coordonnées $P^{(j)}_{i_1,...,i_N}$ des points de contrôle,

$$\mathbf{P} = \begin{pmatrix} P^{(1)}_{0,0,...,0} & \cdots & P^{(M)}_{0,0,...,0} \\ \vdots & \ddots & \vdots \\ P^{(1)}_{n_1,0,...,0} & \cdots & P^{(M)}_{n_1,0,...,0} \\ P^{(1)}_{0,1,...,0} & \cdots & P^{(M)}_{0,1,...,0} \\ \vdots & \ddots & \vdots \\ P^{(1)}_{n_1,1,...,0} & \cdots & P^{(M)}_{n_1,1,...,0} \\ \vdots & \ddots & \vdots \\ P^{(1)}_{n_1,n_2,...,n_N} & \cdots & P^{(M)}_{n_1,n_2,...,n_N} \end{pmatrix}, \tag{4.63}$$

et $\mathbf{Q}$ est la matrice contenant les $M \times n_{TP}$ coordonnées Q_s^j des points cibles,

$$\mathbf{Q} = \begin{pmatrix} Q_1^{(1)} & \cdots & Q_1^{(M)} \\ \vdots & \ddots & \vdots \\ Q_{n_{TP}}^{(1)} & \cdots & Q_{n_{TP}}^{(M)} \end{pmatrix}. \tag{4.64}$$

De même que dans le cas des courbes, la matrice $\mathbf{N}$ doit être calculée et stockée de manière à utiliser la propriété de support local des entités NURBS. Dans ce cas, chaque ligne contient au plus $(p_1+1) \times \cdots \times (p_N+1)$ valeurs non-nulles. Le cas des hypersurfaces B-Spline est également intéressant et dans ce cas la matrice $\mathbf{N}$ est de la forme :

$$\mathbf{N} = \begin{pmatrix} N_{0,p_1}\left(u_1^{(1)}\right) \times \cdots \times N_{0,p_N}\left(u_1^{(N)}\right) & \cdots & N_{n_1,p_1}\left(u_1^{(1)}\right) \times \cdots \times N_{n_N,p_N}\left(u_1^{(N)}\right) \\ \vdots & \ddots & \vdots \\ N_{0,p_1}\left(u_{n_{TP}}^{(1)}\right) \times \cdots \times N_{0,p_N}\left(u_{n_{TP}}^{(N)}\right) & \cdots & N_{n_1,p_1}\left(u_{n_{TP}}^{(1)}\right) \times \cdots \times N_{n_N,p_N}\left(u_{n_{TP}}^{(N)}\right) \end{pmatrix}, \\ \omega_{i_1,...,i_N} = 1, \quad i_k = 0, \ldots, n_k. \tag{4.65}$$

Cette méthode de résolution présente un avantage majeur. En effet, contrairement aux notations introduites en début de chapitre (i.e. $\mathbf{u}_{s_1,...,s_N} = \left(u_{s_1}^{(1)}, ..., u_{s_N}^{(N)}\right)$, $s_k = 0, ..., r_k$), le nombre

de points cibles r_k+1 dans une direction $k=1,...,N$ de l'espace n'a pas besoin d'être défini. Les points cibles peuvent avoir une répartition quelconque dans l'espace des paramètres. En particulier, il est possible de traiter des données fournies par un modèle basé sur les éléments finis avec un maillage libre. Il est cependant possible dans ce cas que la matrice $\mathbf{N}$ soit mal conditionnée, avec notamment des points de contrôle dont le support local ne serait affecté d'aucun point cible. Cela se traduit par une matrice $\mathbf{N}^T\mathbf{N}$ de déterminant nul et une étape de filtrage peut être nécessaire afin de ne pas prendre en considération de tels points de contrôle. La matrice $\mathbf{N}$ étant composée d'un grand nombre de valeurs nulles, il peut être judicieux de ne stocker que les valeurs non-nulles, en utilisant par exemple le format CSR (*Compressed Sparse Row*) ou CSC (*Compressed Sparse Column*) qui sont des formats standardisés pour le stockage de matrices creuses (*sparse matrix*). Le nombre maximal de valeurs non-nulles est donnée par :

$$n_{nzv} = n_{TP} \prod_{k=1}^{N} p_k + 1, \tag{4.66}$$

avec n_{nzv} le nombre maximal de valeurs non-nulles (*Non-Zero Values*).

Cette technique de résolution directe présente cependant une limitation importante. Même en utilisant un stockage approprié de la matrice $\mathbf{N}$, celle-ci peut atteindre des tailles très importantes lorsque la dimension de l'espace paramétrique N, ou que le nombre de points cibles n_{TP}, augmentent. Prenons un exemple simple, d'une base de données de points cibles générée à partir de différents calculs par éléments finis, générant 10 201 points par calcul, pour 5 différentes valeurs de deux paramètres d'entrées (dans cet exemple $N=4$). Le nombre de points cibles, dans ce cas, est de $n_{TP}=255\,025$. Pour un nombre total de points de contrôle raisonnable $n_{CP}=12\,480$, la matrice $\mathbf{N}$ nécessiterait 25To de mémoire pour être stockée en précision *double* entièrement (i.e. valeurs nulles comprises), 165Go en ne stockant que les valeurs non-nulles de cette matrice pour des degrés $p_k=2$ et 522Go pour des degrés $p_k=3$ (à noter que les formats de stockage des matrices creuses nécessitent plus d'espace que le stockage des valeurs non-nulles, les plus simples nécessitant trois fois l'espace de stockage des valeurs non-nulles). Cette limitation était déjà connue dans le cadre des surfaces NURBS, des techniques de calcul plus performantes existent. L'une d'elle consiste notamment à utiliser des méthodes itératives (i.e. utilisant des produits plutôt que l'inversion directe d'une matrice) de résolution de systèmes d'équations linéaires. Ces méthodes peuvent, en particulier, résoudre le système d'équations sans avoir à stocker la matrice $\mathbf{N}^T\mathbf{N}$ à inverser. Dans le cas particulier des B-Splines (i.e. $\omega_{i_1,...,i_N}=1$), le problème de *surface fitting* peut se ramener à une succession de problèmes de *curve fitting* [2], nécessitant, cette fois-ci, le stockage de matrices de tailles bien inférieures. Cette technique est très intéressante car il est courant, dans le cas des surfaces NURBS, de calculer les coordonnées des différents points de contrôle en supposant l'ensemble des poids $\omega_{i_1,...,i_N}$ fixés à 1 (i.e. cas des B-Splines). Les poids sont ensuite optimisés en conservant les valeurs des coordonnées des points de contrôle comme c'est le cas dans [13, 225]. Nous développerons ce principe dans la section suivante et étendrons ce concept au cas de dimension N quelconque.

4.3.2 Algorithme de calcul des coordonnées des points de contrôle d'une surface généralisé aux cas des hypersurfaces

Comme nous l'avons vu précédemment, même si les coordonnées des points de contrôle sont obtenues par la résolution d'un "simple" système d'équations linéaires, la mémoire nécessaire au stockage des matrices mises en jeu peut rendre la résolution impossible. Pour surmonter cette limitation, dans le cas des surfaces B-Splines, une technique classique [2] est de ramener le problème de calcul des coordonnées des points de contrôle de la surface à une succession de calcul des coordonnées de points de contrôle d'une courbe. Cette technique permet de ne pas avoir de matrices de grandes tailles à stocker.

Néanmoins, il est nécessaire, pour utiliser cette méthode, d'avoir un nombre de points cibles $r_k + 1$ défini et constants dans chaque direction $k = 1, 2$ de l'espace. Elle peut cependant être adaptée au cas où un seul nombre de points cibles $r_k + 1$ est défini. Les hypothèses sont les suivantes :

- On approxime $n_{TP} = (r_1 + 1) \times (r_2 + 1)$ points cibles $\mathbf{Q}_{s_1,s_2} = \left(Q^{(1)}_{s_1,s_2}, \ldots, Q^{(M)}_{s_1,s_2}\right) \in \mathbb{R}^M$, $s_k = 0, ..., r_k$,
- aux coordonnées paramétriques $\mathbf{u}_{s_1,s_2} = \left(u^{(1)}_{s_1}, u^{(2)}_{s_2}\right)$,
- par une surface NURBS $\mathbf{S}(\mathbf{u}) : \mathbb{R}^2 \to \mathbb{R}^M$, $\mathbf{u} = \left(u^{(1)}, u^{(2)}\right)$ de dimension M quelconque,
- composée de $n_k + 1$ points de contrôle dans la direction $k = 1, 2$,
- dont les fonctions de base sont de degrés p_k dans la direction $k = 1, 2$,
- dont les *knot vectors* $\mathbf{U}^{(k)}$ sont connus,
- et dont les poids ω_{i_1,i_2} affectés aux points de contrôle $\mathbf{P}_{i_1,i_2} = \left(P^{(1)}_{i_1,i_2}, ..., P^{(M)}_{i_1,i_2}\right)$ sont fixés à 1.

La fig. 4.7 présente l'algorithme de calcul utilisé. La verison originale de cet algorithme peut être retrouvé dans la référence [2] (algorithme A9.7) pour le cas $M = 3$ mais il reste valable dans le cas d'une dimension M quelconque (strictement parlant, si $M > 3$ il s'agit d'une hypersurface NURBS). Le problème d'approximation de la surface (ou hypersurface si $M > 3$) étant ramené à une succession de problème d'approximation de courbes dans chaque direction de l'espace $k = 1, 2$, il est nécessaire de calculer une matrice $\mathbf{N}_k$ dans chaque direction de l'espace. Ces matrices sont composées de $r_k + 1$ lignes et $n_k + 1$ colonnes ($k = 1, 2$) et sont données par :

$$\mathbf{N}_k = \begin{pmatrix} N_{0,p_k}(u^{(k)}_0) & \cdots & N_{n_k,p_k}(u^{(k)}_0) \\ \vdots & \ddots & \vdots \\ N_{0,p_k}(u^{(k)}_{r_k}) & \cdots & N_{n_k,p_k}(u^{(k)}_{r_k}) \end{pmatrix}, \quad k = 1, 2. \tag{4.67}$$

Afin de n'avoir à calculer qu'une seule fois ces matrices pour tous les problèmes d'approximation de courbes, il est nécessaire que le nombre de points cibles r_k+1 soit défini dans chaque direction. Il est envisageable de modifier l'algorithme proposé afin qu'il puisse traiter le cas où un seul des r_k est fixé. Pour cela, la matrice $\mathbf{N}_l$ ($l \neq k$) doit être recalculée à chaque fois, augmentant ainsi le temps de calcul. D'autres solutions peuvent être envisagées comme, par exemple, une étape préliminaire d'approximation afin d'obtenir les valeurs en des points désirés. C'est ce que fait

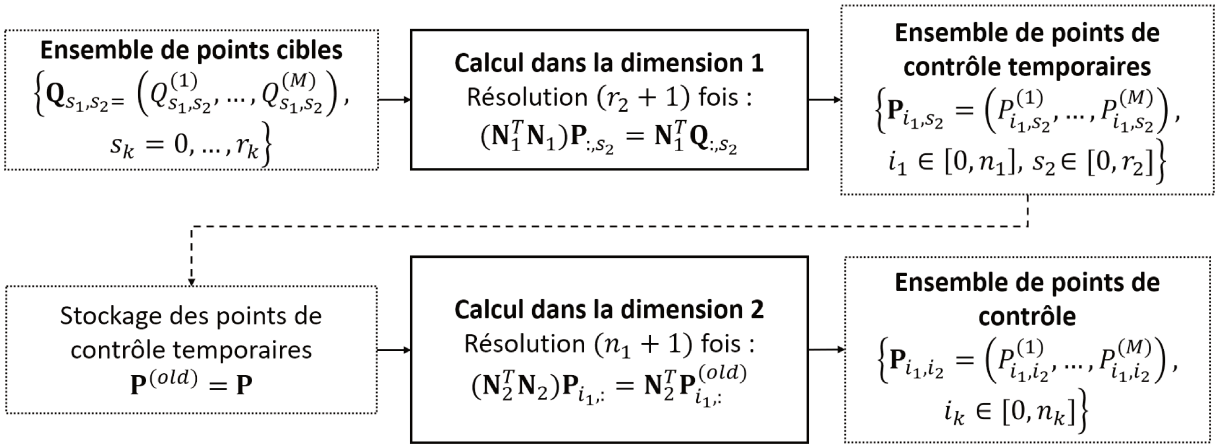


FIGURE 4.7 – Algorithme de calcul, dimension par dimension, des coordonnées des points de contrôle d'une hypersurface B-Spline ($N=2$, M quelconque)

Turner [1] par exemple en utilisant la méthode du plus proche voisin (*nearest neighbor*) pour fixer les coordonnées des points cibles à approximer. Dans la suite, on considère les r_k+1 définis et constant dans chaque direction.

Les matrices $\mathbf{N}_k$ ayant été déterminées, les matrices $\mathbf{N}_k^T\mathbf{N}_k$ sont ensuite calculées et inversées afin d'être également stockées. Le but étant de résoudre un système d'équations linéaires, il s'agit en réalité de stocker la décomposition LU de ces matrices. Ces deux étapes préliminaires étant réalisées, l'algorithme de calcul des coordonnées des points de contrôle commence à approximer des courbes à travers les points cibles dans la direction 1. Cela conduit à résoudre (r_2+1) fois le système d'équations linéaires suivant :

$$\left(\mathbf{N}_1^T\mathbf{N}_1\right)\mathbf{P}_{:,s_2}=\mathbf{N}_1^T\mathbf{Q}_{:,s_2}, \tag{4.68}$$

où $\mathbf{P}_{:,s_2}$, est la $s_2{}^{\text{e}}$ matrice des coordonnées temporaires des points de contrôle donnée par,

$$\mathbf{P}_{:,s_2}=\begin{pmatrix} P^{(1)}_{0,s_2} & \cdots & P^{(M)}_{0,s_2}\\ \vdots & \ddots & \vdots\\ P^{(1)}_{n_1,s_2} & \cdots & P^{(M)}_{n_1,s_2}\end{pmatrix},\quad s_2=0,...,r_2, \tag{4.69}$$

et $\mathbf{Q}_{:,s_2}$ est la $s_2{}^{\text{e}}$ matrice des coordonnées des points cibles donnée par,

$$\mathbf{Q}_{:,s_2}=\begin{pmatrix} Q^{(1)}_{0,s_2} & \cdots & Q^{(M)}_{0,s_2}\\ \vdots & \ddots & \vdots\\ Q^{(1)}_{r_1,s_2} & \cdots & Q^{(M)}_{r_1,s_2}\end{pmatrix},\quad s_2=0,...,r_2. \tag{4.70}$$

L'ensemble des coordonnées des points de contrôle obtenues est stocké dans une variable temporaire,

$$\mathbf{P}^{(old)}=\left\{\mathbf{P}_{i_1,s_2}=\left(P^{(1)}_{i_1,s_2},...,P^{(M)}_{i_1,s_2}\right),\ i_1=0,...,n_1,\ s_2=0,...,r_2\right\}. \tag{4.71}$$

Cet ensemble devient le nouvel ensemble de points cibles pour l'approximation par courbes des données dans la direction 2. Cette fois-ci, il est nécessaire de résoudre $(n_1 + 1)$ fois le système d'équations linéaires suivant :

$$\left(\mathbf{N}_2^T \mathbf{N}_2\right) \mathbf{P}_{i_1,:} = \mathbf{N}_2^T \mathbf{P}_{i_1,:}^{(old)}, \tag{4.72}$$

où $\mathbf{P}_{i_1,:}$, est la $i_1{}^{\text{e}}$ matrice des coordonnées des points de contrôle donnée par,

$$\mathbf{P}_{i_1,:} = \begin{pmatrix} P_{i_1,0}^{(1)} & \cdots & P_{i_1,0}^{(M)} \\ \vdots & \ddots & \vdots \\ P_{i_1,n_2}^{(1)} & \cdots & P_{i_1,n_2}^{(M)} \end{pmatrix}, \quad i_1 = 0, ..., n_1, \tag{4.73}$$

et $\mathbf{P}_{i_1,:}^{(old)}$, est la $i_1{}^{\text{e}}$ matrice des coordonnées temporaires des points de contrôle (calculée à l'étape précédente) donnée par,

$$\mathbf{P}_{i_1,:}^{(old)} = \begin{pmatrix} P_{i_1,0}^{(old,1)} & \cdots & P_{i_1,0}^{(old,M)} \\ \vdots & \ddots & \vdots \\ P_{i_1,r_2}^{(old,1)} & \cdots & P_{i_1,r_2}^{(old,M)} \end{pmatrix}, \quad i_1 = 0, ..., n_1. \tag{4.74}$$

L'avantage principal de cette méthode est qu'elle ne nécessite le stockage que des matrices $\mathbf{N}_k$ et $\left(\mathbf{N}_k^T \mathbf{N}_k\right)^{-1}$ qui sont, généralement, de tailles bien inférieures à celle de $\mathbf{N}$ dans l'éq. (4.52) dès la dimension $N = 2$. En effet, la matrice $\mathbf{N}$ est constituée, en dimension $N = 2$, de $n_{TP} = (r_1 + 1) \times (r_2 + 1)$ lignes et $n_{CP} = (n_1 + 1) \times (n_2 + 1)$ colonnes, tandis que les matrices $\mathbf{N}_k$ sont constituées de $r_k + 1$ lignes et de $n_k + 1$ colonnes. Il est donc préférable d'utiliser la formulation directe (éq. (4.52)) pour de très faibles valeurs de n_k et r_k, ce qui est rarement le cas. Il y a également un intérêt majeur à calculer les matrices $\mathbf{N}_k$ et $\left(\mathbf{N}_k^T \mathbf{N}_k\right)^{-1}$(ou leur décomposition LU) en amont de la phase de calcul des coordonnées des points de contrôle. En effet, il est nécessaire de résoudre $(r_2 + 1)$ fois le système (4.68) et $(n_1 + 1)$ fois le système (4.72). Sans le calcul préalable des matrices $\left(\mathbf{N}_k^T \mathbf{N}_k\right)^{-1}$, il serait nécessaire de la recalculer à chaque nouvelle formulation. Notons que cette possibilité est à envisager pour ne plus être contraint d'utiliser un maillage ordonné de points cibles dans une des deux directions au moins. Cette méthode présente une limitation, outre l'utilisation d'un maillage ordonné de points cibles, il n'existe pas de moyen de connaître *a priori* la direction k dans laquelle il est préférable de débuter l'approximation (i.e. direction 1 puis 2 ou direction 2 puis 1) [2]. Dans le cas où $N = 2$, il est aisé d'essayer les deux approches et de conserver celle qui donne l'erreur d'approximation minimale. Notons cependant que, dans le cas d'une dimension N quelconque, le nombre de possibilité est de $N!$. Il n'est donc pas envisageable d'essayer toutes les approches lorsque N augmente.

Cette technique de calcul des coordonnées des points de contrôle d'une hypersurface B-Spline de dimension $N = 2$ et de dimension M quelconque a été généralisée au cas des hypersurfaces B-Splines de dimension N et M quelconques. Comme dans le cas $N = 2$, il est nécessaire d'avoir un ensemble ordonné de points cibles. Les hypothèses sont les suivantes :

- On approxime $n_{TP} = (r_1 + 1) \times \cdots \times (r_N + 1)$ points cibles $\mathbf{Q}_{s_1,...,s_N} = \left(Q_{s_1,...,s_N}^{(1)}, \ldots, Q_{s_1,...,s_N}^{(M)}\right) \in \mathbb{R}^M$, $s_k = 0, ..., r_k$,

- aux coordonnées paramétriques $\mathbf{u}_{s_1,\ldots,s_N} = \left(u_{s_1}^{(1)}, \ldots, u_{s_N}^{(N)}\right)$,
- par une hypersurface NURBS $\mathbf{H}(\mathbf{u}) : \mathbb{R}^N \to \mathbb{R}^M$, $\mathbf{u} = \left(u^{(1)}, ..., u^{(N)}\right)$
- composée de $n_k + 1$ points de contrôle dans la direction $k = 1, ..., N$,
- dont les fonctions de base sont de degrés p_k dans la direction $k = 1, ..., N$,
- dont les *knot vectors* $\mathbf{U}^{(k)}$ dans la direction $k = 1, ..., N$ sont connus,
- et dont les poids $\omega_{i_1,..,i_N}$ affectés aux points de contrôle $\mathbf{P}_{i_1,\ldots,i_N} = \left(P_{i_1,\ldots,i_N}^{(1)}, ..., P_{i_1,\ldots,i_N}^{(M)}\right)$ sont fixés à 1.

La fig. 4.8 présente l'algorithme de calcul des coordonnées des points de contrôle d'une hypersurface NURBS, généralisé à partir de l'algorithme de la fig. 4.7. Le problème d'approximation

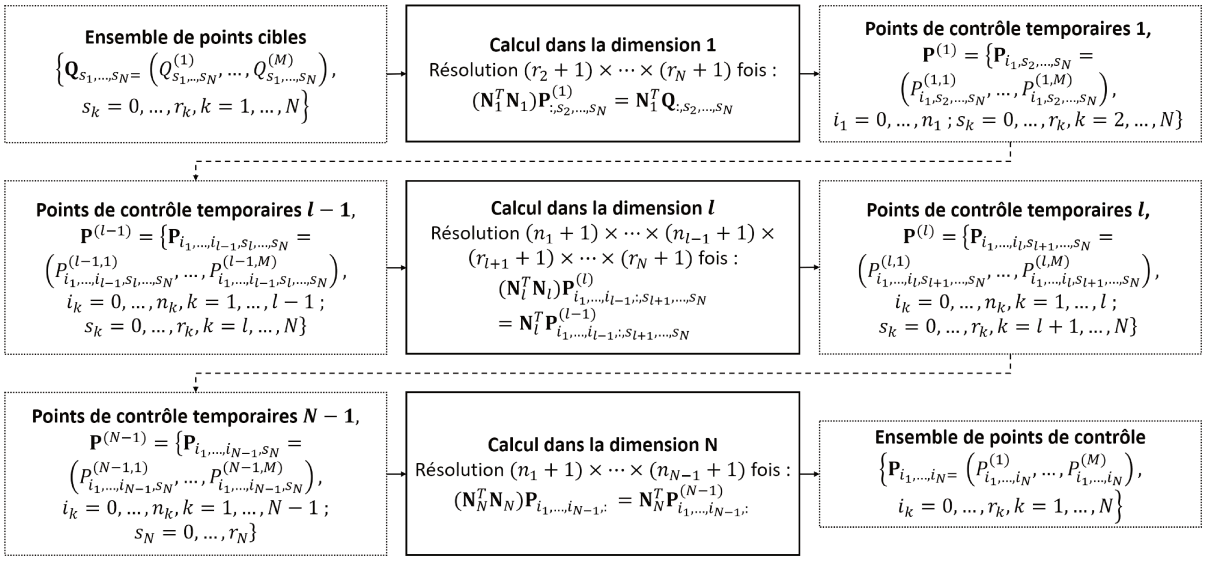


FIGURE 4.8 – Algorithme de calcul, dimension par dimension, des coordonnées des points de contrôle d'une hypersurface B-Spline

de l'hypersurface est ramené à une succession de problème d'approximation de courbes dans chaque direction de l'espace $k = 1, ..., N$. De même que dans le cas $N = 2$, il est nécessaire de calculer, préalablement, une matrice $\mathbf{N}_k$ dans chacune des directions de l'espace. Elles sont données par l'éq. (4.67) avec $k = 1, ..., N$. Les matrices $\mathbf{N}_k^T\mathbf{N}_k$ sont également calculées et inversées initialement. L'algorithme de calcul des coordonnées des points de contrôle débute l'approximation par courbes à travers les points cibles dans la direction 1. Cela conduit à résoudre $(r_2 + 1) \times \cdots \times (r_N + 1)$ fois le système d'équations linéaires suivant :

$$\left(\mathbf{N}_1^T\mathbf{N}_1\right)\mathbf{P}_{:,s_2,\ldots,s_N}^{(1)} = \mathbf{N}_1^T\mathbf{Q}_{:,s_2,\ldots,s_N}, \tag{4.75}$$

où $\mathbf{P}^{(1)}_{:,s_2,...,s_N}$, est la matrice des 1re coordonnées temporaires des points de contrôle à s_k fixés donnée par

$$\mathbf{P}^{(1)}_{:,s_2,...,s_N} = \begin{pmatrix} P^{(1)}_{0,s_2,...,s_N} & \cdots & P^{(M)}_{0,s_2,...,s_N} \\ \vdots & \ddots & \vdots \\ P^{(1)}_{n_1,s_2,...,s_N} & \cdots & P^{(M)}_{n_1,s_2,...,s_N} \end{pmatrix}, \quad s_k = 0,...,r_k, \quad k = 2,...,N, \tag{4.76}$$

et $\mathbf{Q}_{:,s_2,...,s_N}$, est la matrice des coordonnées des points cibles correspondante,

$$\mathbf{Q}_{:,s_2,...,s_N} = \begin{pmatrix} Q^{(1)}_{0,s_2,...,s_N} & \cdots & Q^{(M)}_{0,s_2,...,s_N} \\ \vdots & \ddots & \vdots \\ Q^{(1)}_{r_1,s_2,...,s_N} & \cdots & Q^{(M)}_{r_1,s_2,...,s_N} \end{pmatrix}, \quad s_k = 0,...,r_k, \quad k = 2,...,N. \tag{4.77}$$

L'ensemble des coordonnées des points de contrôle obtenues est stocké dans une variable temporaire,

$$\mathbf{P}^{(1)} = \left\{ \mathbf{P}_{i_1,s_2,...,s_N} = \left(P^{(1)}_{i_1,s_2,...,s_N}, ..., P^{(M)}_{i_1,s_2,...,s_N} \right),\ i_1 = 0,...,n_1,\ s_k = 0,...,r_k,\ k = 2,...,N \right\}. \tag{4.78}$$

Cet ensemble devient le nouvel ensemble de points cibles pour l'approximation par courbes des données dans la direction 2. Ce processus se répète pour chaque direction l de l'espace. En particulier, l'approximation dans la direction l nécessite de résoudre $(n_1+1)\times\cdots\times(n_{l-1}+1)\times(r_{l+1}+1)\times\cdots\times(r_N+1)$ fois le système d'équations linéaires suivant :

$$\left(\mathbf{N}_k^T\mathbf{N}_k\right)\mathbf{P}^{(l)}_{i_1,...,i_{l-1},:,s_{l+1},...,s_N} = \mathbf{N}_k^T\mathbf{P}^{(l-1)}_{i_1,...,i_{l-1},:,s_{l+1},...,s_N}, \tag{4.79}$$

où $\mathbf{P}^{(l)}_{i_1,...,i_{l-1},:,s_{l+1},...,s_N}$ est la l^{e} matrice des coordonnées temporaires des points de contrôle à i_k et s_k fixés donnée par

$$\mathbf{P}^{(l)}_{i_1,...,i_{l-1},:,s_{l+1},...,s_N} = \begin{pmatrix} P^{(l,1)}_{i_1,...,i_{l-1},0,s_{l+1},...,s_N} & \cdots & P^{(l,M)}_{i_1,...,i_{l-1},0,s_{l+1},...,s_N} \\ \vdots & \ddots & \vdots \\ P^{(l,1)}_{i_1,...,i_{l-1},n_l,s_{l+1},...,s_N} & \cdots & P^{(l,M)}_{i_1,...,i_{l-1},n_l,s_{l+1},...,s_N} \end{pmatrix}, \tag{4.80}$$
$$i_k = 0,...,n_k, \quad k = 1,...,l-1\ ;\ s_k = 0,...,r_k, \quad k = l+1,...,N.$$

et $\mathbf{P}^{(l-1)}_{i_1,...,i_{l-1},:,s_{l+1},...,s_N}$, est la $(l-1)^{\text{e}}$ matrice des coordonnées temporaires des points de contrôle à i_k et s_k fixés donnée par,

$$\mathbf{P}^{(l-1)}_{i_1,...,i_{l-1},:,s_{l+1},...,s_N} = \begin{pmatrix} P^{(l-1,1)}_{i_1,...,i_{l-1},0,s_{l+1},...,s_N} & \cdots & P^{(l-1,M)}_{i_1,...,i_{l-1},0,s_{l+1},...,s_N} \\ \vdots & \ddots & \vdots \\ P^{(l-1,1)}_{i_1,...,i_{l-1},r_l,s_{l+1},...,s_N} & \cdots & P^{(l-1,M)}_{i_1,...,i_{l-1},r_l,s_{l+1},...,s_N} \end{pmatrix}, \tag{4.81}$$
$$i_k = 0,...,n_k, \quad k = 1,...,l-1\ ;\ s_k = 0,...,r_k, \quad k = l+1,...,N.$$

Il est important de noter que seules les coordonnées temporaires des points de contrôle de l'étape $l-1$ et l ont besoin d'être stockées en même temps pour l'approximation dans la direction l.

Les coordonnées des points de contrôle sont ainsi calculées de manière itérative, dimension par dimension, jusqu'à la dimension N où le système d'équations linéaires suivant est résolu $(n_1+1)\times\cdots\times(n_{N-1}+1)$ fois :

$$\left(\mathbf{N}_N^T\mathbf{N}_N\right)\mathbf{P}_{i_1,\ldots,i_{N-1},:}=\mathbf{N}_N^T\mathbf{P}^{(N-1)}_{i_1,\ldots,i_{N-1},:}, \tag{4.82}$$

où $\mathbf{P}_{i_1,\ldots,i_{N-1},:}$, est la matrice des coordonnées des points de contrôle à i_k fixés, donnée par

$$\mathbf{P}_{i_1,\ldots,i_{N-1},:}=\begin{pmatrix} P^{(1)}_{i_1,\ldots,i_{N-1},0} & \cdots & P^{(M)}_{i_1,\ldots,i_{N-1},0}\\ \vdots & \ddots & \vdots \\ P^{(1)}_{i_1,\ldots,i_{N-1},n_N} & \cdots & P^{(M)}_{i_1,\ldots,i_{N-1},n_N}\end{pmatrix},\quad i_k=0,\ldots,n_k,\quad k=1,\ldots,N-1, \tag{4.83}$$

et $\mathbf{P}^{(N-1)}_{i_1,\ldots,i_{N-1},:}$ est la $(N-1)^{\text{e}}$ matrice des coordonnées temporaires des points de contrôle à i_k fixés, donnée par

$$\mathbf{P}^{(N-1)}_{i_1,\ldots,i_{N-1},:}=\begin{pmatrix} P^{(N-1,1)}_{i_1,\ldots,i_{N-1},0} & \cdots & P^{(N-1,M)}_{i_1,\ldots,i_{N-1},0}\\ \vdots & \ddots & \vdots \\ P^{(N-1,1)}_{i_1,\ldots,i_{N-1},r_N} & \cdots & P^{(N-1,M)}_{i_1,\ldots,i_{N-1},r_N}\end{pmatrix},\quad i_k=0,\ldots,n_k,\quad k=1,\ldots,N-1. \tag{4.84}$$

Les matrices $\mathbf{N}_k$, contrairement à la matrice $\mathbf{N}$ de la formulation générale, sont de tailles raisonnables. Reprenons l'exemple du paragraphe précédent où la matrice $\mathbf{N}$ à stocker nécessitait au minimum $165GB$ de mémoire dans le cas de fonctions de base de degrés $p_k=2$. Dans cet exemple, où la dimension $N=4$, un calcul élément fini permet d'obtenir $(r_1+1)\times(r_2+1)=10\,201$ points cibles ($r_1+1=101$ et $r_2+1=101$). Ce calcul est effectué pour 5 valeurs différentes de 2 paramètres soit $r_3+1=5$ et $r_4+1=5$. Pour des nombres de points de contrôle tels que $n_1=50, n_2=50, n_3=2$ et $n_4=2$ (soit $n_{CP}=23\,409$), les matrices $\mathbf{N}_1$ et $\mathbf{N}_2$ nécessitent $41ko$ de mémoire tandis que les matrices $\mathbf{N}_3$ et $\mathbf{N}_4$ ne requiert que $120o$ sans un stockage particulier (i.e. le stockage de matrice creuse n'est pas employé). Le nombre total de points de contrôle a presque été doublé par rapport l'exemple donné pour la formulation générale qui, malgré l'utilisation d'un stockage approprié, peut nécessiter des téraoctets de mémoire alors que cette méthode nécessite seulement quelques kilooctets sans tirer parti du stockage des matrices creuses. On peut noter que la dimension M n'influe pas sur la taille des matrices $\mathbf{N}$ ou $\mathbf{N}_k$ à stocker. En revanche, elle intervient sur la taille des matrices des points de contrôle et des points cibles.

L'avantage d'une telle formulation pour le calcul des coordonnées des points de contrôle est clairement indéniable, tant sur le plan du stockage des matrices mises en jeu que sur le temps de calcul. En effet, les matrices à multiplier et à inverser sont de petites tailles, et comme nous l'avons vu, l'inversion peut être faite une seule fois pour chaque direction de l'espace, la résolution des différents systèmes se ramenant à un produit matricielle. Il faut cependant noter que cette méthode nécessite une répartition ordonnée des points cibles. Cet ordre se traduit par un maillage ordonné des points cibles avec un nombre fixe r_k+1 de points dans chaque direction $k=1,\ldots,N$ de l'espace. Cet répartition n'a cependant pas l'obligation d'être homogène

(i.e. le pas de discrétisation peut être variable). Malgré l'avantage indéniable de cette méthode, la limitation concernant l'obtention des points cibles peut être une réelle contrainte. Dans le cas d'une base de données obtenue par des calculs éléments finis, le maillage peut ne pas être ordonné et varier d'un jeu de paramètres à un autre. Cela implique que la base de données ne peut pas toujours être générée en tenant compte de la contrainte imposée par le calcul des coordonnées des points de contrôle. Il existe plusieurs solutions dans un tel cas. Premièrement, il est possible de "réordonner" les points cibles, en utilisant, dans le cas des éléments finis par exemple, les fonctions de forme pour obtenir les valeurs en des points spécifiquement choisis et ainsi générer une nouvelle base de donnée ordonnée. Il est aussi possible, comme Turner [1] par exemple, d'utiliser une première méthode d'approximation avant le calcul des coordonnées de l'hypersurface. Dans le cas de Turner, la méthode du plus proche voisin est utilisée mais il est également possible d'envisager d'autres techniques. Nous avons opté pour une stratégie "hybride" mêlant la formulation générale du problème de calcul des coordonnées des points de contrôle à l'approche direction par direction précédemment décrite. Cette méthode hybride sera l'objet de la prochaine section.

4.3.3 Algorithme hybride de calcul des coordonnées des points de contrôle de l'hypersurface

Comme nous l'avons vu précédemment, la formulation générale du système d'équations linéaires permettant d'obtenir les coordonnées des points de contrôle est difficilement utilisable lorsque le nombre de points cibles et/ou la dimension N augmentent. A contrario, la technique de résolution, dimension par dimension, n'est pas limitée par la taille de la base de données ou la dimension N mais elle impose une contrainte sur l'organisation de la base de données qui doit admettre un ordre parmi les points cibles. Cela se traduit par un nombre de points $r_k + 1$ dans chaque direction $k = 1, ..., N$ de l'espace. Cette contrainte n'est pas toujours réalisable, comme dans le cas de résultats de calculs par éléments finis dont le maillage est complexe (maillage libre ou maillage structuré de cercle par exemple).

Pour améliorer les limitations engendrées par les deux méthodes, nous avons développé un concept hybride, liant ces deux approches. Pour cela, on définit les hypothèses suivantes :

- On approxime $n_{TP} = (r_L + 1) \times (r_{L+1} + 1) \times \cdots \times (r_N + 1)$ points cibles $\mathbf{Q}_{s_L, s_{L+1}, ..., s_N} = \left(Q^{(1)}_{s_L, s_{L+1}, ..., s_N}, \ldots, Q^{(M)}_{s_L, s_{L+1}, ..., s_N}\right) \in \mathbb{R}^M$, $s_k = 0, ..., r_k$, $k = L, ..., N$,
- aux coordonnées paramétriques $\mathbf{u}_{s_L, s_{L+1}, ..., s_N} = \left(u^{(1)}_{s_L}, \ldots, u^{(L)}_{s_L}, u^{(L+1)}_{s_{L+1}}, \ldots, u^{(N)}_{s_N}\right)$.
- Soit $L < N$ le nombre de dimensions dont les points cibles sont désordonnés et $s_L = 0, \ldots, r_L$ l'indice associé à cet ensemble de points cibles.
- L'approximation est réalisée par une hypersurface NURBS $\mathbf{H}(\mathbf{u}) : \mathbb{R}^N \rightarrow \mathbb{R}^M$, $\mathbf{u} = \left(u^{(1)}, ..., u^{(N)}\right)$
- composée de $n_k + 1$ points de contrôle dans la direction $k = 1, ..., N$,
- dont les fonctions de base sont de degrés p_k dans la direction $k = 1, ..., N$,
- dont les *knot vectors* $\mathbf{U}^{(k)}$ dans la direction $k = 1, ..., N$ sont connus,

- et dont les poids $\omega_{i_1,..,i_N}$ affectés aux points de contrôle $\mathbf{P}_{i_1,...,i_N} = \left(P^{(1)}_{i_1,...,i_N}, ..., P^{(M)}_{i_1,...,i_N}\right)$ sont fixés à 1.

L'idée à la base de ce concept hybride est d'utiliser la formulation générale de résolution, donnée dans l'éq. (4.52), sur les dimensions $1, ..., L$ n'ayant pas pu être ordonnées. Dans le cas de résultats de calculs par éléments finis, $L = 2$ pour une modélisation $2D$ (modèle surfacique de plaque par exemple) et $L = 3$ pour une modélisation $3D$ (modèle surfacique de coque ou modèle volumique par exemple). Les autres paramètres pouvant être ordonnés, le calcul, dimension par dimension, est utilisé après cette première étape. La fig. 4.9 présente l'algorithme hybride utilisé.

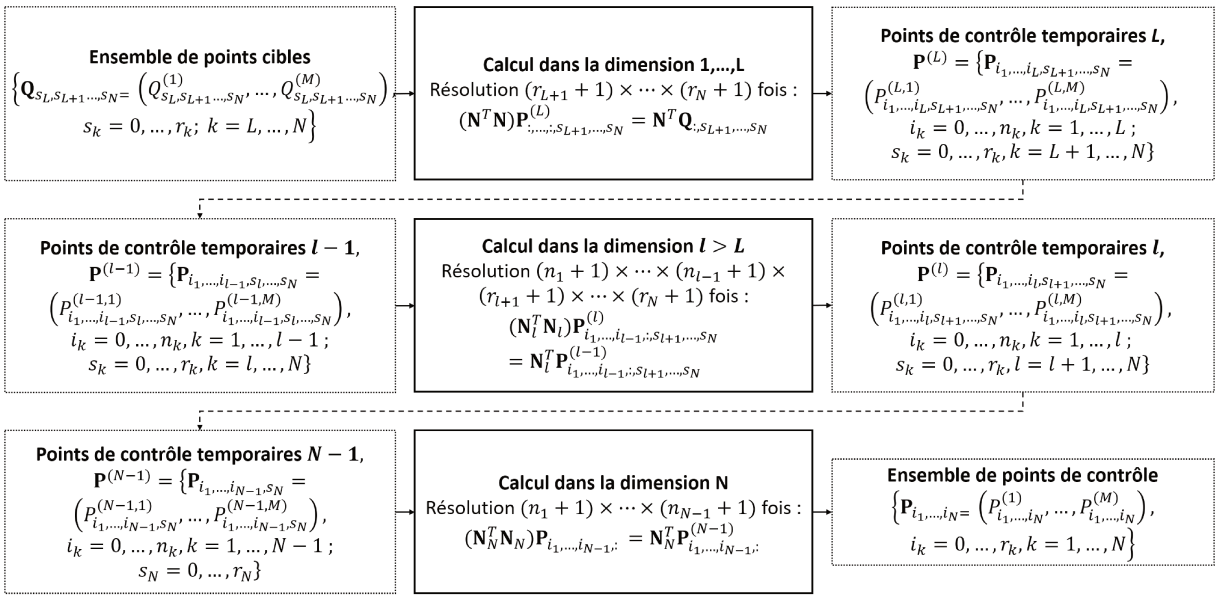


FIGURE 4.9 – Algorithme hybride de calcul des coordonnées des points de contrôle d'une hypersurface B-Spline

Dans un premier temps, il faut résoudre $(r_{L+1}+1) \times \cdots \times (r_N+1)$ fois le système d'équations linéaires suivant :

$$\left(\mathbf{N}^T\mathbf{N}\right)\mathbf{P}^{(L)}_{:,...,:,s_{L+1},...,s_N} = \mathbf{N}^T\mathbf{Q}_{:,s_{L+1},...,s_N}, \tag{4.85}$$

avec $\mathbf{P}^{(L)}_{:,...,:,s_{L+1},...,s_N}$ la matrice contenant les $(n_1+1) \times \cdots \times (n_L+1)$ coordonnées temporaires des points de contrôle à s_k fixés,

$$\mathbf{P}^{(L)}_{:,...,:,s_{L+1},...,s_N} = \begin{pmatrix} P^{(L,1)}_{0,...,0,s_{L+1},...,s_N} & \cdots & P^{(L,M)}_{0,...,0,s_{L+1},...,s_N} \\ \vdots & \ddots & \vdots \\ P^{(L,1)}_{n_1,...,n_L,s_{L+1},...,s_N} & \cdots & P^{(L,M)}_{n_1,...,n_L,s_{L+1},...,s_N} \end{pmatrix}, \tag{4.86}$$
$$s_k = 0, ..., r_k, \quad k = L+1, ..., N,$$

$\mathbf{Q}_{:,s_{L+1},...,s_N}$, la matrice des coordonnées des r_L+1 points cibles à s_k fixés,

$$\mathbf{Q}_{:,s_{L+1},...,s_N} = \begin{pmatrix} Q^{(1)}_{0,s_{L+1},...,s_N} & \cdots & Q^{(M)}_{0,s_{L+1},...,s_N} \\ \vdots & \ddots & \vdots \\ Q^{(1)}_{r,s_{L+1},...,s_N} & \cdots & Q^{(M)}_{r,s_{L+1},...,s_N} \end{pmatrix}, \quad s_k = 0,...,r_k, \quad k = L+1,...,N, \quad (4.87)$$

et $\mathbf{N}$ la matrice telle que :

$$\mathbf{N} = \begin{pmatrix} N_{0,p_1}(u_0^{(1)}) \times \cdots \times N_{0,p_L}(u_0^{(L)}) & \cdots & N_{n_1,p_1}(u_0^{(1)}) \times \cdots \times N_{n_L,p_L}(u_0^{(L)}) \\ \vdots & \ddots & \vdots \\ N_{0,p_1}(u_{r_L}^{(1)}) \times \cdots \times N_{0,p_L}(u_{r_L}^{(L)}) & \cdots & N_{n_1,p_1}(u_{r_L}^{(1)}) \times \cdots \times N_{n_L,p_L}(u_{r_L}^{(L)}) \end{pmatrix}. \quad (4.88)$$

Notons que dans le cas où les valeurs de $u_{s_L}^{(k)}$, $k = 1,...,L$, sont identiques $\forall s_k = 0,...,r_k$, $k = L+1,...,N$ (le maillage ne varie pas, d'un calcul à l'autre, dans le cas de résultats obtenus par une méthode de type éléments finis par exemple), la matrice $\mathbf{N}$ sera identique pour toutes les $(r_{L+1}+1) \times \cdots \times (r_N+1)$ résolutions à faire. En particulier, il est alors possible de calculer $\mathbf{N}$ et $\left(\mathbf{N}^T\mathbf{N}\right)^{-1}$ une seule fois pour tous les calculs. À l'inverse, si les $u_{s_L}^{(k)}$ varient, il faut recalculer la matrice $\mathbf{N}$ $(r_{L+1}+1) \times \cdots \times (r_N+1)$ fois. La suite de l'algorithme est la même que dans le cas du calcul dimension par dimension, l'étape préliminaire nous ayant permis d'obtenir directement $\mathbf{P}^{(L)}$. En particulier, l'éq. (4.79) s'applique pour le calcul dans la dimension $l > L$. Le calcul est ainsi effectué dans les dimensions restantes $(L+1,...,N)$ jusqu'à obtenir l'ensemble des coordonnées des points de contrôle.

Cette technique, bien que tirant parti des avantages des deux méthodes précédemment introduites, n'annule pas la limitation de la formulation générale, elle permet seulement de réduire son impact. En effet, si le nombre de points cibles désordonnés r_L+1 et/ou que le nombre de dimension non ordonnées L augmentent, la taille des matrices à stocker peut rendre impossible le calcul des coordonnées des points de contrôle. Notons cependant qu'elle permet de ne pas traiter les N dimensions et l'ensemble des points cibles n_{TP} en une fois, ce qui permet à cette approche d'être beaucoup moins limitée que la formulation générale. Notons également que, moyennant le calcul de $(r_{L+1}+1) \times \cdots \times (r_N+1)$ matrices $\mathbf{N}$ différentes, cette méthode s'adapte à des points cibles dont les L premières coordonnées $u_{s_L}^{(k)}$, $k = 1,...,L$, varient. Il apparaît cependant pertinent de lier la génération des points cibles à la méthode de calcul des coordonnées des points de contrôle employée.

4.4 L'algorithme HERO : Hybrid EvolutionaRy Optimization

4.4.1 Stratégie de résolution de CNLPP

Comme nous l'avons vu dans le chapitre précédent, l'utilisation de méthodes déterministes (i.e. basées sur l'utilisation du gradient des fonctions objectif et contraintes) ne permet pas d'atteindre un optimum global dans le domaine faisable sans un point de départ approprié pour l'algorithme. Les métaheuristiques sont quant à elles capables, sans le garantir, de fournir (théoriquement) un optimum global. Cependant, un nombre de variables important peut conduire à

une exploration du domaine trop coûteuse pour parvenir à un optimum global. Dans ce contexte, nous avons développé l'algorithme HERO (*Hybrid EvolutionaRy Optimisation*), tirant parti des deux différentes approches de résolution. En effet, dans un premier temps, une métaheuristique permet de trouver un point de départ convenable afin de permettre à une méthode déterministe d'atteindre, dans le pire des cas, un minimum local proche du minimum global et dans le meilleur, un minimum global.

Une connaissance profonde des phénomènes régissant le comportement que l'on cherche à modéliser, pourrait permettre à un utilisateur-expert de choisir lui même le point de départ permettant à la méthode déterministe d'obtenir un minimum global. Or, dans notre cas, cela signifierait une expertise du phénomène à modéliser par le métamodèle ainsi qu'une expertise dans les hypersurfaces NURBS. Dans la pratique, il est très peu probable de posséder cette double expertise pour un utilisateur, et la méthode de réduction de modèle que nous développons a pour vocation d'être très versatile sans nécessiter aucun savoir-faire particulier de la part de l'utilisateur. Le choix du point de départ de la méthode déterministe n'a donc pas besoin d'être fixé par l'utilisateur. Ce point ne peut pas non plus être choisi de manière aléatoire. C'est pourquoi une métaheuristique est employée pour fournir ce point de départ.

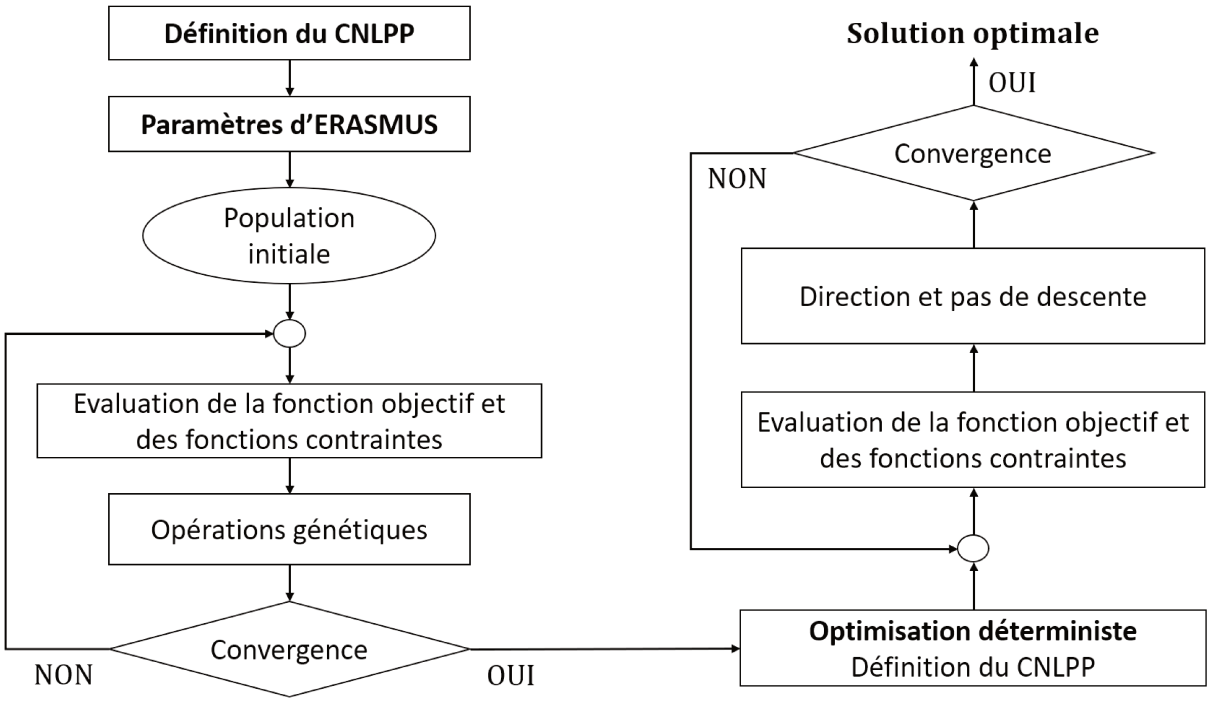


FIGURE 4.10 – Algorithme HERO (*Hybrid EvolutionaRy Optimisation*)

L'algorithme HERO est présenté dans la fig. 4.10. L'un de ces avantages réside dans l'utilisation de l'algorithme génétique ERASMUS capable de traiter des CNLPP dont le nombre de variables d'optimisation n'est pas constant. Il est notamment capable de traiter des problèmes d'optimisation de systèmes modulaires composés de différents types de modularité *indépendants*.

Dans un premier temps, les paramètres de l'algorithme ERASMUS sont fixés par l'utilisateur (structure des individus, nombre de populations, nombre d'individus, ...). Dans le cas des systèmes modulaires, outre l'obtention d'un "bon" point de départ pour la méthode déterministe, la résolution du CNLPP par ERASMUS permet d'obtenir le nombre de modules optimal pour chaque type de module. Cela permet ensuite de traiter le CNLPP par une méthode déterministe à nombre de variables d'optimisation constant (i.e. les différentes valeurs des modules de chaque composant). Dans le cas où le nombre de variables d'optimisation est constant, ERASMUS fourni seulement de "bonnes" valeurs pour ces variables en entrée de la méthode déterministe.

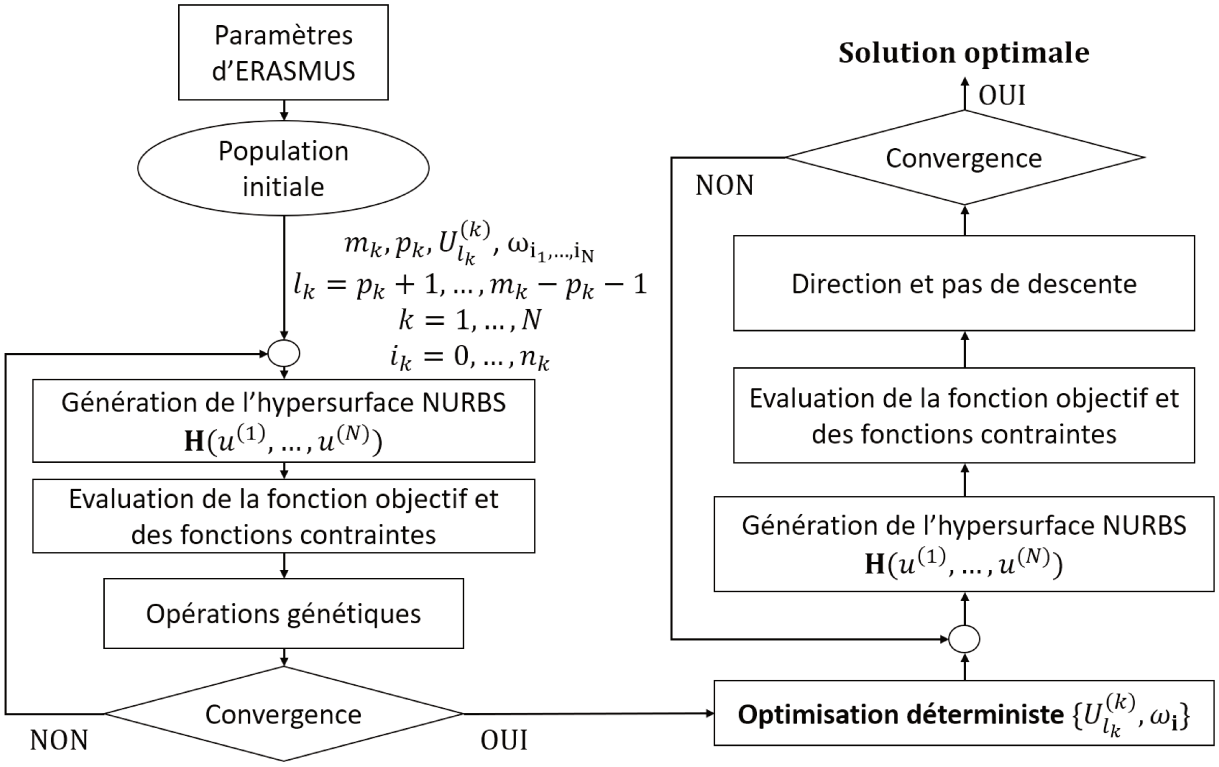


FIGURE 4.11 – Algorithme HERO (*Hybrid EvolutionaRy Optimisation*) pour l'optimisation d'une hypersurface NURBS

Lorsqu'un critère de convergence est atteint par ERASMUS, la solution qu'il fournit sert de point de départ pour la méthode déterministe. Dans le cas de systèmes modulaires, la solution fournie fixe le nombre de variables d'optimisation du CNLPP. Dans certain cas, le CNLPP résolu par l'algorithme déterministe peut différer de celui résolu par ERASMUS. En effet, dans notre cas, par exemple, la fonction objectif de l'éq. (4.42) ne fait intervenir que le nombre de points de contrôle et les degrés des fonctions de base de l'hypersurface NURBS. Or, la modification de ces paramètres entraine automatiquement un changement du nombre de valeurs internes des *knot vectors*. Il n'est donc pas possible pour la méthode déterministe de modifier ces valeurs et le CNLPP qui lui est fourni diffère de celui traité par ERASMUS. La fig. 4.11 présente l'algorithme

HERO, adapté au cas de l'optimisation d'une hypersurface NURBS. On voit, notamment, que la méthode déterministe n'admet que les valeurs internes des *knot vectors* et les poids en entrée. Il est toutefois possible, dans d'autres situations, de traiter le même CNLPP pour les deux approches (si le nombre de variables d'optimisation est constant par exemple). La convergence de la méthode déterministe permet d'obtenir une solution optimale.

La stratégie à la base de l'algorithme HERO est bien évidemment généralisable à un grand nombre de CNLPPs. Différentes métaheuristiques pourraient être utilisées pour la première phase exploratoire mais peu de méthodes permettent la résolution de CNLPPs de dimension variable, ce qui fait d'ERASMUS un outil très versatile. Le choix de la méthode déterministe utilisée a, lui, été motivé par le langage de programmation dans lequel a été implémenté ERASMUS. N'importe quelle autre méthode déterministe pourrait être utilisée en remplacement.

4.4.2 Problème d'optimisation traité par ERASMUS

Bien que généralisable à un grand nombre de CNLPPs, nous détaillons ici la version de HERO adaptée à l'optimisation de modèles réduits basés sur les hypersurfaces NURBS (fig. 4.11). Tout d'abord, il est important de préciser que l'intégration des poids $\omega_{i_1,...,i_N}$ parmi les variables d'optimisation rend l'exploration du domaine par ERAMSUS très complexe. Cette difficulté se répercute également sur le calcul des coordonnées des points de contrôle lorsque la dimension N ou que le nombre de points cibles sont très grands. C'est pourquoi l'optimisation est réalisée en deux temps. Une première exploration du domaine est réalisée en intégrant seulement les degrés p_k et les valeurs internes des *knot vectors* parmi les variables d'optimisation. Cette première étape consiste donc à optimiser les paramètres d'une hypersurface B-Spline (i.e. $\omega_{i_1,...,i_N} = 1$, $\forall i_k = 0, ..., n_k$) afin d'approximer la base de données de points cibles. Les algorithmes décrits dans la section 4.3 sont alors utilisés pour calculer les coordonnées des points de contrôle. A noter que les hypersurfaces B-Spline donnent, en général, une bonne approximation et cela permet notamment de fixer la taille des *knot vectors* $m_k + 1$ dans chaque direction (nombre de variables d'optimisation constant pour les étapes suivantes). Contrairement au cas des courbes et surfaces NURBS où tous les poids sont optimisés [13, 225, 226], nous avons choisi d'utiliser le caractère local intrinsèque des entités NURBS. Ainsi, ne sont introduits parmi les variables d'optimisation que les poids $\omega_{i_1,...,i_N}$ où l'erreur d'approximation de l'hypersurface B-Spline dépasse le seuil prévu par l'utilisateur. Cette approche en deux temps permet de réduire considérablement le temps d'exploration du domaine en réduisant le nombre de variables d'optimisation au nombre minimal nécessaire pour le seuil d'erreur d'approximation souhaité. De plus, le choix d'utiliser une hypersurface B-Spline ou NURBS n'incombe pas à l'utilisateur. En effet, si à la fin de la première phase le seuil d'erreur est respecté, alors le metamodèle généré utilisera simplement une hypersurface B-Spline. Dans le cas contraire, les poids seront introduits et le metamodèle généré utilisera une hypersurface NURBS.

L'objectif principal de la première phase d'exploration d'ERASMUS est de trouver la taille des *knot vectors* $m_k + 1$ et les degrés p_k permettant de garantir un certain seuil d'erreur sur les sorties du metamodèle. Le CNLPP traité par ERASMUS dans cette étape est donc de la forme

suivante :

$$\min_{\mathbf{x}_\mathrm{I}} \phi(\mathbf{x}_\mathrm{I}) = a \times \frac{1}{N} \sum_{k=1}^{N} \frac{n_k}{n_k^{max}} + (1-a) \times \frac{1}{N} \sum_{k=1}^{N} \frac{p_k}{p_k^{max}}, \quad 0 < a < 1,$$

$$\text{sujet à :}$$

$$\begin{cases} \epsilon_j^{(max)}(\mathbf{x}_\mathrm{I}) \leq \epsilon_{j,max}^{(max)}, \quad j = 1, ..., M, \\ \epsilon_j^{(moy)}(\mathbf{x}_\mathrm{I}) \leq \epsilon_{j,max}^{(moy)}, \quad j = 1, ..., M, \\ 0 < U_{l_k}^{(k)} < 1, \quad l_k = p_k + 1, ..., m_k - p_k + 1, \quad k = 1, ..., N, \\ U_{l_k}^{(k)} \leq U_{l_k+1}^{(k)}, \quad l_k = p_k + 1, ..., m_k - p_k, \quad k = 1, ..., N, \\ n_k - p_k \geq 0, \quad k = 1, ..., N, \\ n_{CP} \leq n_{TP}, \end{cases} \tag{4.89}$$

où $\mathbf{x}_\mathrm{I}$ est le vecteur contenant les variables d'optimisation, i.e. les N degrés p_k, les N tailles des *knot vectors* m_k+1 et les m_k-2p_k-1 valeurs internes des *knot vectors* (les $(n_1+1)\times\cdots\times(n_N+1)$ poids $\omega_{i_1,\ldots,i_N}$ sont fixés à 1). Le vecteur $\mathbf{x}_\mathrm{I}$ regroupe les variables dans l'ordre suivant :

$$\mathbf{x}_\mathrm{I} = \left(p_1, \ldots, p_N, m_1, \ldots, m_N, U_{p_1+1}^{(1)}, \ldots, U_{m_1-p_1-1}^{(1)}, \ldots, U_{p_N+1}^{(N)}, \ldots, U_{m_N-p_N-1}^{(N)}\right). \tag{4.90}$$

Comme nous l'avons vu précédemment, l'utilisateur peut pondérer la fonction objectif à l'aide du paramètre a afin de minimiser en priorité, soit le nombre de données nécessaire à l'évaluation du métamodèle, soit le temps d'évaluation. L'utilisateur doit également fixer le nombre maximum de points de contrôle $n_k^{max}+1$ ainsi que le degré maximum p_k^{max} dans chaque direction k. En l'absence de connaissances sur le, ou les, phénomènes que l'on cherche à modéliser, le choix naturel pour les valeurs de ces paramètres est de fixer $n_k^{max} = r_k$, $k = 1, ..., N$. Cependant, il n'est pas toujours possible d'obtenir une base de données de points cibles pour laquelle le nombre de points $r_k + 1$ est défini dans toutes les directions k. Dans un tel cas, en considérant que $r_L + 1$ est le nombre de points suivant les L premières dimensions et que $r_k + 1$ est celui dans les directions $k = L + 1, ..., N$, la fonction objectif (4.42) peut être remplacée par la fonction suivante :

$$\phi^{(L)}(\mathbf{x}_\mathrm{I}) = a \times \frac{\prod_{k=1}^{N}(n_k+1)}{n_{TP}} + (1-a) \times \frac{1}{N} \sum_{k=1}^{N} \frac{p_k}{p_k^{max}}, \quad 0 < a < 1. \tag{4.91}$$

avec le nombre de points cibles n_{TP} donné par :

$$n_{TP} = (r_L + 1) \times \prod_{k=L+1}^{N} (r_k + 1) \tag{4.92}$$

Bien qu'il existe des valeurs naturelles pour fixer les valeurs des n_k^{max}, le choix des degrés maximums p_k^{max} est moins évident. Les entités NURBS utilisent des fonctions de base polynomiales définies entre nœuds successifs de chaque direction de l'espace. Dans le cas où la dimension $N = 1$, la courbe NURBS obtenue est en fait une succession de courbes polynomiales de degré

p qui se relient aux différents nœuds. De ce fait, augmenter le degré peut résulter en un effet *vaguelette*, bien connu dans le cas de l'approximation polynomiale. Cet effet est aussi connu dans le cas des courbes et surfaces NURBS et, souvent, l'utilisation des poids permet de diminuer cet effet. Il est cependant très rare, en pratique, d'utiliser des fonctions de base de degrés $p_k > 10$, $k = 1, 2$. Cette règle peut être étendue au cas des hypersurfaces NURBS, ainsi, nous conseillons des valeurs $p_k^{max} \leq 10$, $k = 1, ..., N$. Les degrés minimums p_k^{min} sont, quant à eux, facilement paramétrables en fonction de la dérivabilité minimale souhaitée pour les différentes sorties du métamodèle ainsi que la continuité souhaitée aux nœuds de l'hypersurface NURBS. Pour rappel (cf. chapitre 3) la dérivabilité dans une direction k en un nœud $U_{l_k}^{(k)}$ est $p_k - \lambda_{l_k}$, λ_{l_k} étant la multiplicité du nœud $U_{l_k}^{(k)}$.

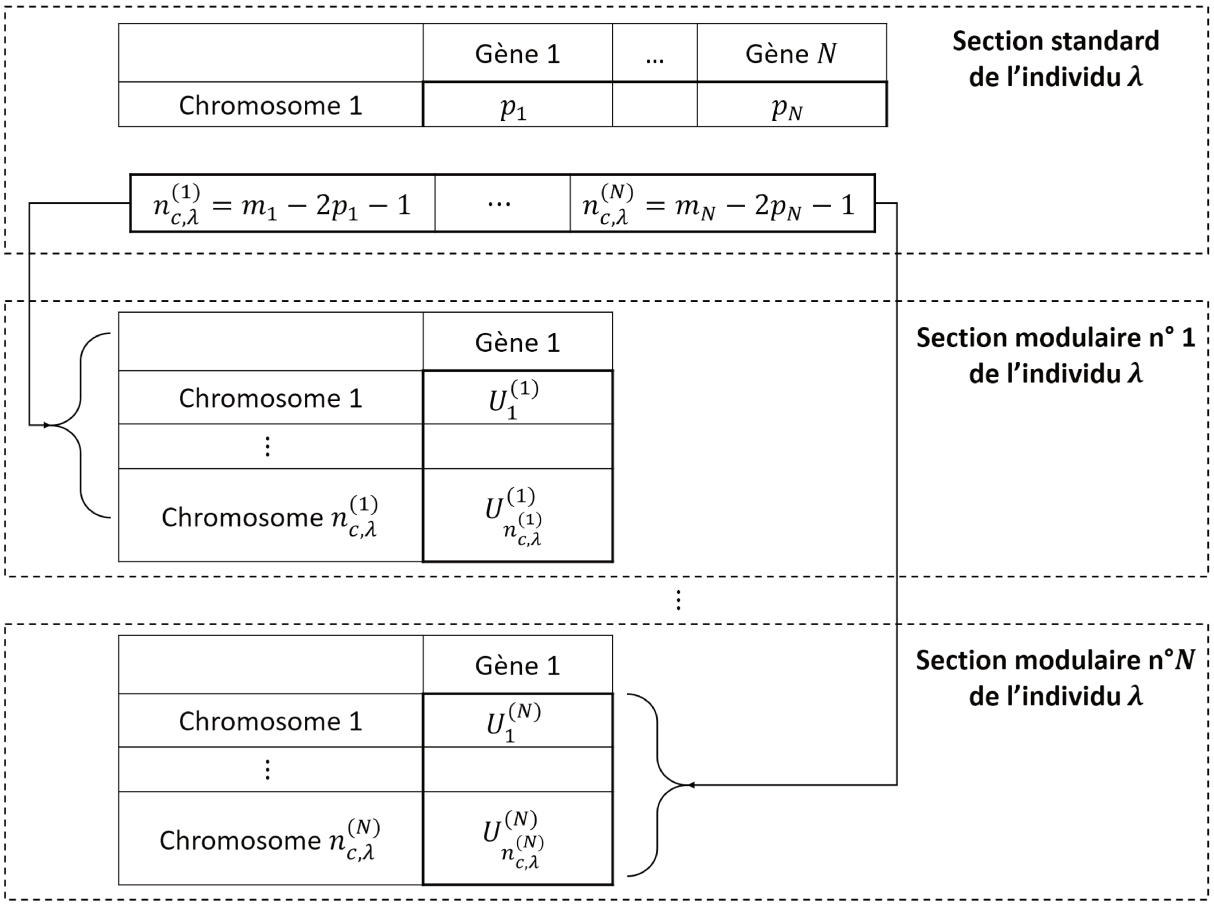


FIGURE 4.12 – Structure d'un individu d'ERASMUS pour l'optimisation d'hypersurfaces NURBS dans le cas de l'approximation

Le CNLPP (4.89) étant complètement défini, il est nécessaire de fixer certains paramètres de l'algorithme ERAMSUS. Ces paramètres sont définis dans [14, 212, 227]. Ici, seule la nouvelle structure du génotype de l'individu sera présentée en mettant l'accent sur l'analogie entre le

problème de l'*hypersurface fitting* et le problème d'optimisation d'un système modulaire avec différents types de modules. En particulier, dans le cadre de la formulation générale du problème de l'approximation d'une hypersurface, la structure d'un individu est donnée par la fig. 4.12. Il est important de noter que les valeurs $U_{l_k}^{(k)}$, $k = 1, ..., N$, des N composants modulaires de l'individu ne sont pas ordonnées, i.e. rien ne garantit que la propriété suivante soit vérifiée :

$$\forall l_k < l'_k, \quad U_{l_k}^{(k)} \leq U_{l'_k}^{(k)}. \tag{4.93}$$

Or, les *knot vectors* d'une hypersurface NURBS doivent obligatoirement être définis comme des suites de nombres non-décroissants. Une opération de mise en ordre des valeurs des *knot vectors* est donc nécessaire avant le calcul des coordonnées des points de contrôle. Dans notre cas, cela se traduit par l'utilisation de la fonction *sort* de Matlab [228]. Cela implique que la position des chromosomes, au sein d'un composant modulaire, n'a pas d'incidence.

Nous avions listé, préalablement, les variables d'optimisation comme étant les degrés p_k, les tailles $m_k + 1$ et les valeurs internes $U_{l_k}^{(k)}$ des *knot vectors*. Or, la fig. 4.12 permet de remarquer que les individus fournissent bel et bien les degrés p_k (section standard de l'individu), les valeurs internes des *knot vectors* $U_{l_k}^{(k)}$ (sections modulaires de l'individu) mais les m_k ne sont pas stockées directement. Ce sont, en fait, les nombres de chromosomes $n_{c,\lambda}^{(k)} = m_k - 2p_k - 1$ qui sont stockés dans la section standard de l'individu λ. Ces informations permettent, néanmoins, d'obtenir tous les paramètres d'entrée de l'hypersurface NURBS. Les nombres de points de contrôle $n_k + 1$ sont notamment donnés par :

$$n_k = m_k - p_k - 1, \quad k = 1, ..., N. \tag{4.94}$$

Il est possible que les valeurs de n_k obtenues par l'éq. (4.94) conduisent à une hypersurface NURBS dont les coordonnées des points de contrôle ne peuvent pas être calculées ($n_{CP} > n_{TP}$). Dans un tel cas, la valeur de la fonction objectif, pour cet individu, est fixé à 1 (valeur maximale de la fonction objectif).

La résolution du CNLPP (4.89) a pour objectif principal de fixer le nombre de variables d'optimisation pour les étapes suivantes. En particulier, cette étape fixe le nombre de valeurs internes $m_k - 2p_k - 1$ des différents *knot vectors* et les degrés p_k, ce qui a pour conséquence de fixer également le nombre de points de contrôle $n_k + 1$ dans chaque direction $k = 1, ..., N$. Comme nous l'avons dit précédemment, si le seuil d'erreur est respecté dans la première phase d'exploration, les poids $\omega_{i_1,...,i_N}$ restent fixés à 1 (i.e. hypersurface B-Spline). Si ce n'est pas le cas, nous avons choisi d'intégrer seulement les poids dans les zones où le seuil d'erreur d'approximation dépasse le seuil fixé par l'utilisateur. La raison essentielle est de ne pas complexifier le modèle inutilement. Cela est rendu possible grâce à la propriété de support local des entités NURBS. En effet, l'éq. (3.26) permet de remarquer que chaque point de contrôle, et donc chaque poids qui lui est affecté, a un domaine d'influence bien délimité. La réciproque s'applique également et les coordonnées de chaque point de l'hypersurface ne dépendent que d'un certain nombre de points de contrôle. On note,

$$\mathbb{S}^{(i_1,...,i_N)} = \left[U_{i_1}^{(1)}, U_{i_1+p_1+1}^{(1)}\right[\times ... \times \left[U_{i_N}^{(N)}, U_{i_N+p_N+1}^{(N)}\right[, \tag{4.95}$$

le support local (hyper-rectangle) du poids $\omega_{i_1,...,i_N}$ (ou du point de contrôle $\mathbf{P}_{i_1,...,i_N}$). On note,

$$\mathbb{U} = \left\{ \mathbf{u}_s, \quad s = 1, ..., n_{TP} \quad | \quad \exists j \in [1, M] : \epsilon_j(\mathbf{u}_s) > \epsilon_{j,max}^{(max)} \right\} \tag{4.96}$$

avec $\mathbf{u}_s$, $s = 1, ..., n_{TP}$, les coordonnées paramétriques des n_{TP} points cibles. L'ensemble $\mathbb{U}$ regroupe donc les points cibles pour lesquels le seuil d'erreur d'approximation n'est pas respecté pour au moins une sortie du métamodèle. Avec ces notations on définit,

$$\mathbf{\Omega} = \left\{ \omega_{i_1,...,i_N}, \quad i_k = 0, ..., n_k \quad | \quad \exists \mathbf{u}_s \in \mathbb{U} \cap \mathbb{S}^{(i_1,...,i_N)} \right\}, \tag{4.97}$$

l'ensemble des poids $\omega_{i_1,...,i_N}$ dont le support local contient au moins un point cible $\mathbf{u}_s$, $s = 1, ..., n_{TP}$, qui ne respecte pas l'erreur d'approximation pour au moins une sortie du métamodèle.

Les degrés p_k et les nombres de points de contrôle n_k+1 étant fixés, la fonction ϕ ne peut plus être utilisée comme fonction objectif du problème d'optimisation des paramètres de l'hypersurface NURBS. C'est pourquoi, le second CNLPP résolu par ERASMUS est de la forme suivante :

$$\begin{aligned} &\min_{\mathbf{x}_{\mathrm{II}}} \varphi(\mathbf{x}_{\mathrm{II}}) = \frac{1}{M} \sum_{j=1}^{M} \frac{\epsilon_j^{(max)}(\mathbf{x}_{\mathrm{II}})}{\epsilon_j^{(max)}\left(\mathbf{x}_{\mathrm{I}}^{(0)}\right)}, \\ &\text{sujet à :} \\ &\begin{cases} \dfrac{\epsilon_j^{(moy)}(\mathbf{x}_{\mathrm{II}})}{\epsilon_{j,max}^{(moy)}} \leq 1, \quad j = 1, ..., M, \\ 0 < U_{l_k}^{(k)} < 1, \quad l_k = p_k + 1, ..., m_k - p_k + 1, \quad k = 1, ..., N, \\ U_{l_k}^{(k)} \leq U_{l_k+1}^{(k)}, \quad l_k = p_k + 1, ..., m_k - p_k, \quad k = 1, ..., N, \\ \omega_{i_1,...,i_N} \geq 0, \quad i_k = 0, ..., n_k, \end{cases} \end{aligned} \tag{4.98}$$

où $\mathbf{x}_{\mathrm{II}}$ est le vecteur contenant les variables d'optimisation (i.e. les $n_k - p_k$ valeurs internes des *knot vectors* ainsi que les poids $\omega_{i_1,...,i_N} \in \mathbf{\Omega}$) et $\mathbf{x}_{\mathrm{I}}^{(0)}$ est le vecteur des variables d'optimisation fourni par ERASMUS après la résolution du CNLPP (4.89). Le vecteur $\mathbf{x}_{\mathrm{II}}$ regroupe les variables dans l'ordre suivant :

$$\mathbf{x}_{\mathrm{II}} = \left(U_{p_1+1}^{(1)}, \ldots, U_{m_1-p_1-1}^{(1)}, \ldots, U_{p_N+1}^{(N)}, \ldots, U_{m_N-p_N-1}^{(N)}, \mathbf{\Omega} \right). \tag{4.99}$$

Le vecteur $\mathbf{x}_{\mathrm{II}}$ des variables d'optimisation du CNLPP (4.98) est différent du vecteur $\mathbf{x}_{\mathrm{I}}$ des variables d'optimisation du CNLPP (4.89). En particulier, les tailles des *knot vectors* $m_k + 1$ ne font plus partie des variables d'optimisation et les poids $\omega_{i_1,...,i_N} \in \mathbf{\Omega}$ ont été introduits. De plus, les nombres de points de contrôle $n_k + 1$ et les degrés p_k sont fixés par les variables d'optimisation de $\mathbf{x}_{\mathrm{I}}^{(0)}$. Dans cette phase, les coordonnées des points de contrôle sont obtenues par les algorithmes décrits dans la section 4.3 et les poids n'interviennent que dans le calcul des points de l'hypersurface NURBS. Pour améliorer la convergence, un individu ayant les $m_k - 2p_k - 1$ valeurs internes des *knot vectors* de la solution fournie par ERASMUS suite à la résolution du CNLPP (4.89) (i.e. $\mathbf{x}_{\mathrm{I}}^{(0)}$) ainsi que tous ses poids $\omega_{i_1,...,i_N} \in \mathbf{\Omega}$ fixés à 1 est introduit dans la population initiale pour la résolution du CNLPP (4.98).

La résolution du CNLPP (4.98) par ERAMSUS permet de fournir un "bon" point de départ pour une méthode déterministe. Ces deux étapes permettent notamment de fixer les nombres

optimaux de points de contrôle n_k+1 et les degrés p_k, $k = 1, ..., N$ garantissant une certaine erreur maximale d'approximation pour chacune des sorties du métamodèle. Elle permet également de fournir à la méthode déterministe, des valeurs internes des *knot vectors* et des poids $\omega_{i_1,...,i_N} \in \mathbf{\Omega}$ proches des valeurs optimales. Il est à noter que le problème pourrait directement être traité par une méthode déterministe dès la fin de la première phase d'exploration. En effet, le nombre de variables d'optimisation peut être fixé par cette première étape (le nombre de points de contrôle et les degrés sont fixés), il est alors possible de chercher un optimum local par une méthode déterministe. Cependant, cela ne doit se faire que dans une optique de gain de temps. En effet, l'introduction des poids parmi les variables d'optimisation rend le problème fortement non-linéaire et l'utilisation d'une méthode déterministe n'est donc pas recommandée.

4.4.3 Problème d'optimisation traité par la méthode déterministe de MATLAB

Comme évoqué précédemment, l'étape d'optimisation réalisée avec le code ERASMUS a deux objectifs : fixer le nombre de variables d'optimisation ; et fournir des valeurs initiales de ces variables proches d'un optimum local (dans le meilleur des cas, proches de l'optimum global). L'objectif de la seconde étape d'optimisation est, quant à lui, d'affiner la valeur de l'optimum (local ou global). En effet, les méthodes déterministes focalisent la recherche de l'optimum sur un seul point, et non sur une population, ce qui rend la convergence plus rapide. Les objectifs des deux étapes étant différents, la formulation du CNLPP traité par la méthode déterministe peut différer de celle du CNLPP abordé par la métaheuristique.

Dans le cas de l'optimisation d'une hypersurface NURBS, le CNLPP (4.89), traité par ERASMUS, ne peut pas être traité par une méthode déterministe. En effet, la fonction objectif de ce problème vise à minimiser le nombre points de contrôle et le temps d'évaluation (i.e. les degrés des fonctions de base) du métamodèle obtenu, en garantissant un certain seuil d'erreur maximale d'approximation pour chacune des sorties. Le nombres de points de contrôle $n_k + 1$ et le degrés p_k dans chaque direction ne peuvent, cependant, pas faire partie des variables d'optimisation du problème traité par la méthode déterministe car il ne s'agit pas de variables continues et qu'elles sont liées au nombre de variables de conception. L'objectif de cette deuxième étape d'optimisation ne consiste donc pas à minimiser le nombre de paramètres et le temps d'évaluation du métamodèle, mais bel et bien à minimiser l'erreur maximale d'approximation de celui-ci. Dans ce cas, il pourrait être tentant d'utiliser la fonction φ du CNLPP (4.98) comme fonction objectif de notre problème de minimisation. Or cela n'est pas envisageable car la fonction φ est discontinue, ce qui n'est pas compatible avec une méthode d'optimisation basée sur le gradient. Ainsi, le CNLPP traité, dans notre cas, par la fonction *fmincon* de MATLAB utilise la χ-norme pour approximer l'erreur maximale d'approximation :

$$\max_s \left(H^{(j)}(\mathbf{u}_s) - Q_s^{(j)} \right) \approx \left[\sum_{s=1}^{n_{TP}} \left(H^{(j)}(\mathbf{u}_s) - Q_s^{(j)} \right)^{\chi} \right]^{(\frac{1}{\chi})}, \quad j = 1, ..., M, \quad \chi \geq 20 \qquad (4.100)$$

Le CNLPP qui résulte de l'utilisation de la χ-norme est de la forme suivante :

$$\begin{aligned}
&\min_{\mathbf{x}_{\mathrm{II}}} \psi^{(\chi)}(\mathbf{x}_{\mathrm{II}}) = \frac{1}{M}\sum_{j=1}^{M} \frac{\left[\sum_{s=1}^{n_{TP}} \left(H^{(j)}_{\mathbf{x}_{\mathrm{II}}}(\mathbf{u}_s) - Q^{(j)}_s\right)^{\chi}\right]^{(\frac{1}{\chi})}}{\left[\sum_{s=1}^{n_{TP}} \left(H^{(j)}_{0}(\mathbf{u}_s) - Q^{(j)}_s\right)^{\chi}\right]^{(\frac{1}{\chi})}}, \\
&\text{sujet à :} \\
&\begin{cases}
0 < U^{(k)}_{l_k} < 1, \quad l_k = p_k + 1, ..., m_k - p_k + 1, \quad k = 1, ..., N, \\
U^{(k)}_{l_k} \leq U^{(k)}_{l_k+1}, \quad l_k = p_k + 1, ..., m_k - p_k, \quad k = 1, ..., N, \\
\omega_{i_1,...,i_N} \geq 0, \quad i_k = 0, ..., n_k,
\end{cases}
\end{aligned} \tag{4.101}$$

où $\mathbf{H}_{\mathbf{x}_{\mathrm{II}}} = \left(H^{(1)}_{\mathbf{x}_{\mathrm{II}}}, ..., H^{(M)}_{\mathbf{x}_{\mathrm{II}}}\right)$ est l'hypersurface NURBS obtenue avec le vecteur des variables d'optimisation $\mathbf{x}_{\mathrm{II}}$ tandis que $\mathbf{H}_0 = \left(H^{(1)}_0, ..., H^{(M)}_0\right)$ est l'hypersurface NURBS obtenue avec le vecteur des variables d'optimisation $\mathbf{x}^{(0)}_{\mathrm{II}}$, point de départ de la méthode déterministe (i.e. les valeurs fournies par ERASMUS).

L'utilisation de la χ-norme permet deux choses. D'une part, la fonction $\psi^{(\chi)}$ est continue, ce qui permet l'utilisation d'une méthode déterministe, et d'autre part elle permet de conserver une qualité d'approximation "globale" sans contraintes supplémentaires (i.e. pas de contraintes sur l'erreur moyenne d'approximation). La fonction objectif est donnée sans dimension afin d'éliminer d'éventuels effets de variation d'échelle brutale. Comme nous l'avons dit précédemment, nous utilisons la fonction *fmincon* de MATLAB, disponible dans l'*optimization toolboox* [214]. Afin de traiter le CNLPP (4.101), un certain nombre de paramètres doivent être renseignés :

- Le point de départ $\mathbf{x}_0 = \left(U^{0\,(1)}_{p_1+1}, \cdots, U^{0\,(1)}_{m_1-p_1+1}, \cdots, U^{0\,(N)}_{p_N+1}, \cdots, U^{0\,(N)}_{m_N-p_N+1}, \omega^0_{i_1,...,i_N} \in \mathbf{\Omega}\right)$ contenant les $n_k - p_k$ valeurs internes des N *knot vectors* $\mathbf{U}^{0\,(k)}$ obtenues par ERASMUS ainsi que les poids $\omega^0_{i_1,...,i_N}, \omega^0_{i_1,...,i_N} \in \mathbf{\Omega}$.
- Le type d'algorithme déterministe utilisé (IP, AS ou SQP). Par défaut, MATLAB utilise l'algorithme IP. Nous lui préférons cependant la méthode AS pour sa robustesse et son traitement des contraintes.
- Les critères de convergence. Nous avons vu précédemment que MATLAB gère plusieurs critères simultanément :
  - *Nombre maximal d'évaluations de la fonction objectif.* La valeur par défaut donnée, dans le cas de l'algorithme AS, est de 100× le nombre de variables. Cette valeur peut être utilisée dans la plupart des cas.
  - *Nombre maximal d'itérations* : K_{max}. La valeur par défaut de MATLAB est $K_{max} = 400$. Dans la pratique, nous n'avons jamais atteint ce nombre d'itérations et il est difficile de connaître à l'avance le nombre d'itérations nécessaire. De ce fait, la valeur par défaut est une valeur acceptable, permettant, en cas de divergence, de stopper l'algorithme.

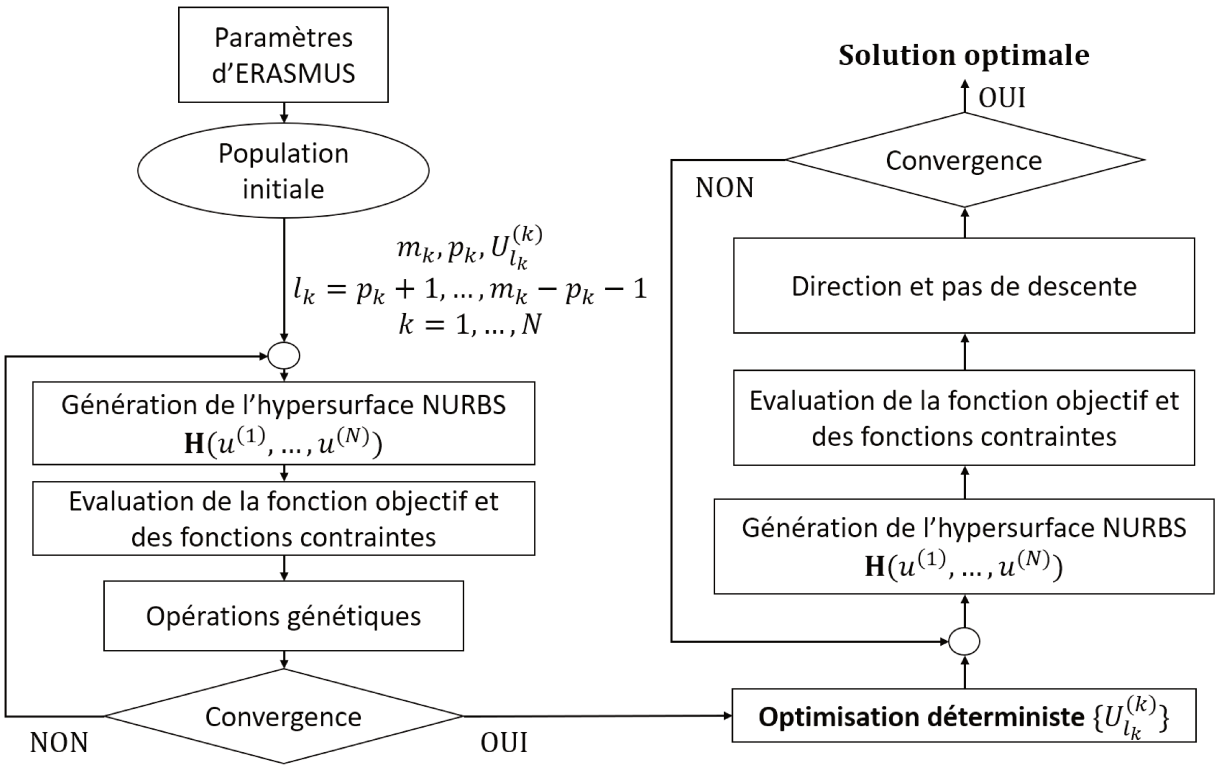


FIGURE 4.13 – Algorithme HERO (*Hybrid EvolutionaRy Optimisation*) dans le cas de l'optimisation d'une hypersurface B-Spline

  - — *Faible amélioration de la fonction objectif* : σ_f. MATLAB attribue une valeur par défaut de $\sigma_f = 10^{-6}$. Les valeurs de la fonction objectif étant sans dimension, cette valeur peut être conservée. Cependant, dans le cas où l'erreur ne serait pas sans dimension, la valeur de σ_f devrait être paramétrée en fonction de l'ordre de grandeur de la fonction objectif.
  - — *Variation négligeable des valeurs des variables d'optimisation* : σ_x. MATLAB attribut par défaut la valeur $\sigma_x = 10^{-6}$. Les valeurs internes des *knot vectors* étant sans dimension, la valeur par défaut peut être utilisée.
  - — *Norme du gradient de la fonction de Lagrange proche de 0* : σ_∇. La valeur $\sigma_\nabla = 10^{-6}$ est attribuée par défaut par MATLAB. Dans le cas où les valeurs de la fonction objectif et des fonctions contraintes sont sans dimension, cette valeur est acceptable.
- Le seuil de tolérance de dépassement des contraintes. Cette valeur est fixée par défaut à 10^{-6} et permet l'exploration des bornes du domaine. Pour des valeurs de fonctions contraintes sans dimension, la valeur par défaut est utilisable.

La solution fournie par la fonction *fmincon* permet d'obtenir la précision optimale du métamodèle pour les nombres de points de contrôle $n_k + 1$ et les degrés p_k, $k = 1, ..., N$, obtenus par ERASMUS.

Même si la stratégie présentée jusqu'à maintenant permet de déterminer automatiquement s'il sera nécessaire d'utiliser des hypersurfaces NURBS ou B-Spline, dans beaucoup de cas, les hypersurfaces B-Spline permettent d'obtenir des précisions acceptables. C'est pourquoi, dans un soucis de gain de temps, un utilisateur peut être intéressé par l'utilisation d'hypersurface B-Spline uniquement. Une version spéciale du code a été développée dans ce cas et est présentée sur la fig. 4.13. La structure de l'algorithme est très similaire à celle de la fig. 4.11, à la différence que les poids ne sont pas introduits parmi les variables d'optimisation. De ce fait, seul le CNLPP (4.89) est traité par ERASMUS et la solution fournie sert de point de départ pour la méthode déterministe où seules les valeurs internes des *knot vectors* font partie des variables d'optimisation.

4.5 Conclusions

Une nouvelle stratégie de résolution de CNLPPs, l'algorithme HERO, a été présentée dans ce chapitre. L'utilisation combinée d'une métaheuristique et d'une méthode déterministe permet de traiter des CNLPPs, pour lesquels, les phénomènes physiques en jeu sont très complexes et difficilement compréhensibles sans expertise. De plus, l'utilisation de la métaheuristique ERASMUS permet de traiter des CNLPPs dont le nombre de variables d'optimisation n'est pas constant. Dans ce contexte, le CNLPP traité par ERAMSUS est différent de celui traité par la méthode déterministe. En effet, l'objectif principal d'ERASMUS, en plus de fournir les valeurs des variables d'optimisation formant un "bon" point de départ, est de déterminer le *nombre optimal* de variables d'optimisation permettant de respecter les contraintes fixées par l'utilisateur. L'objectif de la méthode déterministe est alors la recherche d'un minimum (local ou global) minimisant l'erreur d'approximation, tout en respectant une contrainte sur l'erreur moyenne, avec le nombre de variables obtenu par ERAMSUS.

Cette stratégie, bien qu'applicable à un grand nombre de problèmes, a été illustrée dans le cadre de l'optimisation d'hypersurfaces NURBS pour la réduction de modèle. Bien que notre méthode de réduction de modèle ait été développée dans le cadre de l'approximation (i.e. $n_{CP} < n_{TP}$), elle est tout à fait applicable au cas de l'interpolation (i.e. $n_{CP} = n_{TP}$). Dans les deux cas, le code HERO est utilisé afin d'obtenir les valeurs optimales de l'hypersurface NURBS. Lors de la première phase d'optimisation, l'utilisateur peut choisir d'accentuer soit la minimisation du nombre total de paramètres nécessaires à l'évaluation du métamodèle (i.e. le nombre total de points de contrôle n_{CP}), soit le temps d'évaluation (i.e. les degrés $p_k, k = 1, ..., N$) de celui-ci. L'utilisateur fixe également les erreurs maximales d'approximation du métamodèle. ERASMUS fournit alors, en deux temps, les paramètres de l'hypersurface NURBS permettant de respecter au mieux les contraintes fixées par l'utilisateur. Une méthode déterministe permet ensuite d'atteindre un minimum (local ou global) minimisant l'erreur d'approximation en formulant un second CNLPP. Dans de nombreux cas, l'utilisation d'hypersurfaces B-Spline permet d'obtenir la précision souhaitée. Seules les valeurs internes des *knot vectors* font alors partie des variables d'optimisation dans cette seconde phase d'optimisation (i.e. le nombre total de points de contrôle n_{CP} et les degrés p_k sont fixés). La méthode proposée peut, cependant, choisir d'utiliser des hypersurfaces NURBS et intégrer les poids $\omega_{i_1,...,i_N}$ aux variables d'optimisation. Le nombre total de poids peut cependant s'avérer trop important, et la stratégie que nous avons

développée s'appuie sur le caractère local intrinsèque des entités NURBS. Ainsi, seuls les poids affectés à des points de contrôle dans des zones où l'erreur d'approximation dépasse le seuil fixé par l'utilisateur sont introduits parmi les variables d'optimisation.

La méthode d'optimisation HERO permet, dans le cadre de la réduction de modèle basée sur les hypersurfaces NURBS, de s'affranchir des règles empiriques jusqu'alors utilisées et ne bénéficiant d'aucune justification physique. De plus, aucune expertise de l'utilisateur n'est requise pour appliquer cette méthode. En effet, des connaissances approfondies des entités NURBS ne sont pas nécessaires puisque c'est l'algorithme HERO qui en fixe les paramètres. Le besoin d'expertise n'est pas non plus transféré sur la métaheuristique ERASMUS puisque la plupart de ces paramètres peuvent être fixés à des valeurs par défaut. Seuls restent les critères de convergences de celui-ci mais le nombre maximal de générations peut être fixé de telle sorte que le nombre total d'évaluations de la fonction objectif soit égal à 80000 et le seuil de la fonction objectif peut être déterminé par l'utilisateur en fonction d'un gain attendu, par rapport à une solution de référence, ou par rapport au nombre de points cibles de la base de données. Enfin, il n'est pas non plus nécessaire d'avoir des connaissances approfondies des méthodes déterministes, et en particulier de la fonction *fmincon* de MATLAB utilisée, puisque les critères d'arrêts par défaut peuvent être utilisés dans le cas où la fonction objectif et les fonctions contraintes sont données sans dimension.

L'efficacité de l'approche proposée sera prouvée sur des benchmarks tirés de la littérature ainsi que sur un problème pratique de complexité industrielle. Les benchmarks envisagés concernent : un problème thermique défini sur un domaine convexe dont la sortie dépend de quatre paramètres d'entrée ($N = 4$ et $M = 1$) ; un problème de mécanique en élasticité linéaire sur un domaine convexe dont les deux sorties dépendent de quatre paramètres ($N = 4$ et $M = 2$) ; et un problème de mécanique en élasticité linéaire sur un domaine non-convexe dont les deux sorties dépendent de quatre paramètres ($N = 4$ et $M = 2$). L'application pratique traite, quant à elle, l'unité répétitive d'un panneau raidi avec prise en compte d'un problème aux valeurs propres : la détermination de la charge critique de flambage de la structure.

Chapitre 5

Cas d'application de la méthode

5.1 Introduction

La méthode de réduction de modèle, développée au chapitre 4, a été testée sur différents cas d'application "simples" en apparence mais ayant été choisis pour exhiber certains comportements particuliers. Le premier problème traité, par exemple, vise à modéliser, par une hypersurface NURBS, le champ de température d'une plaque mince en fonction de l'épaisseur de celle-ci et du coefficient de convection avec le milieu environnant. Dans ce cas l'hypersurface admet quatre paramètres en entrée, à savoir, les deux coordonnées de la plaque x et y, l'épaisseur t et le coefficient de convection h. La sortie, quant à elle, est la température dépendant de ces quatre paramètres. Les conditions limites appliquées à la plaque rendent la variation de température indépendante de la variable y, l'objectif étant, pour notre méthode, d'obtenir un métamodèle exhibant cette propriété.

Dans le second cas d'application, le métamodèle concerne le champ de déplacement d'une plaque mince en fonction de l'épaisseur de celle-ci et de la norme d'un effort qui lui est appliqué. Dans cet exemple, il y a quatre paramètres en entrée, à savoir, les coordonnées de la plaque x et y, l'épaisseur t et la norme de l'effort appliqué F. Le nombre de sorties est ici de trois, u_x, u_y et θ_z, correspondant au champ de déplacement de la plaque en fonction des quatre paramètres précédents. L'objectif, pour notre méthode, est de montrer sa capacité à traiter des problèmes ayant plusieurs sorties, sans nécessiter un nouveau métamodèle pour chaque sortie ainsi que sa capacité à intégrer des conditions limites parmi les variables d'entrée.

Enfin, le dernier cas d'application présenté dans ce chapitre traite le champ de déplacement d'une plaque mince trouée en fonction de l'épaisseur et du rayon du trou de celle-ci. Il y a également quatre variables d'entrée dans cet exemple, à savoir, les coordonnées de la plaque x et y, l'épaisseur de la plaque t et le rayon du trou R. Il y a également trois sorties, u_x, u_y et θ_z, correspondantes au champ de déplacement de la plaque trouée en fonction des paramètres d'entrée. Dans cet exemple, la géométrie de la plaque varie en fonction du rayon R de celle-ci. L'objectif de notre technique de réduction de modèle dans cet exemple est de montrer sa capacité à s'adapter à des géométries variables en intégrant des paramètres de conception dans ces variables d'entrée.

Afin de mettre en lumière l'intérêt de notre méthode, les résultats obtenus ont été comparés

à une méthode itérative utilisant les règles empiriques classiques [2] pour fixer, notamment, les degrés p_k et les valeurs internes des *knot vectors*. En particulier, les degrés $p_k = 2, k = 1, ..., N$ sont utilisés (comme dans le cas de Turner [1]) et les valeurs internes des *knot vectors* $U^{(k)}_{i_k}$ sont fixées en fonctions des valeurs des coordonnées sans dimensions des points cibles $u^{(k)}_{s_k}$ par la relation suivante [2] :

$$\begin{aligned} d_k &= \frac{r_k + 1}{n_k - p_k + 1}, \qquad l_k = \lfloor j_k d_k \rfloor, \qquad \alpha_k = j_k d_k - l_k, \\ U^{(k)}_{p_k + j_k} &= (1 - \alpha_k)\, u^{(k)}_{l_k - 1} + \alpha_i u^{(k)}_{l_k}, \quad j_k = 1, ..., n_k - p_k,\ k = 1, ..., N. \end{aligned} \tag{5.1}$$

Dans l'éq. (5.1), $\lfloor \cdot \rfloor$ désigne la fonction partie entière par défaut. Cette règle empirique assure que la matrice $\left(\mathbf{N}^T\mathbf{N}\right)$, à inverser pour le calcul des coordonnées des points de contrôle, soit définie positive et bien conditionnée [2]. Ceci est dû au fait que l'éq. (5.1) assure qu'il y ait, au moins, un $u^{(k)}_{s_k}$ entre chaque nœud du *knot vector*.

Comme dans le cas de Turner [1], les degrés sont fixés à $p_k = 2$, $k = 1, ..., N$. Initialement, le nombre de points de contrôle dans chaque direction $n_k + 1$, $k = 1, ..., N$ sont fixés à $n_k = 2$. Ensuite, à chaque itération, le nombre de points de contrôle $n_k + 1$ est augmenté dans la direction la direction k dont le ratio suivant est le plus faible :

$$q_k = \frac{n_k}{r_k}, \quad k = 1, ..., N. \tag{5.2}$$

Le quotient q_k représente une mesure du "chargement" de points de contrôle dans une direction. Des points sont donc ajoutés dans la direction étant la moins "chargée".

5.2 Problème thermique : Champ de température d'une plaque mince avec épaisseur et coefficient de convection variables $\mathbb{R}^4 \to \mathbb{R}$

5.2.1 Modélisation du problème et génération de la base de données

Dans cette section, notre méthode de réduction de modèle est appliquée à un problème thermique. La géométrie de la plaque, ainsi que les conditions limites qui lui sont appliquées sont illustrées sur la fig. 5.1. La plaque est de longueur $L_x = 0.1$m selon l'axe x, de longueur $L_y = 0.1$m selon l'axe y et d'épaisseur $t \in [t_{min}, t_{max}]$, avec $t_{min} = 0.001$m et $t_{max} = 0.01$m. Un flux $\Phi_0 = 100\text{W/m}^2$ est appliqué en $x = 0$ et en $x = L_x$ tandis que le reste de la plaque est soumis à un phénomène convectif avec l'air ambiant, de température infinie $T_\infty = 298.15$K et de coefficient $h \in [h_{min}, h_{max}]$, avec $h_{min} = 5\text{W/m}^2$ et $h_{max} = 100\text{W/m}^2$. L'objectif du métamodèle est de donner une représentation du champ de température pour différentes valeurs des paramètres t et h, soit :

$$T = f(x, y, t, h). \tag{5.3}$$

Dans ce contexte, l'épaisseur t et le coefficient h représentent des paramètres de conception tandis que les variables x et y représentent un point de la plaque. La base de données de points

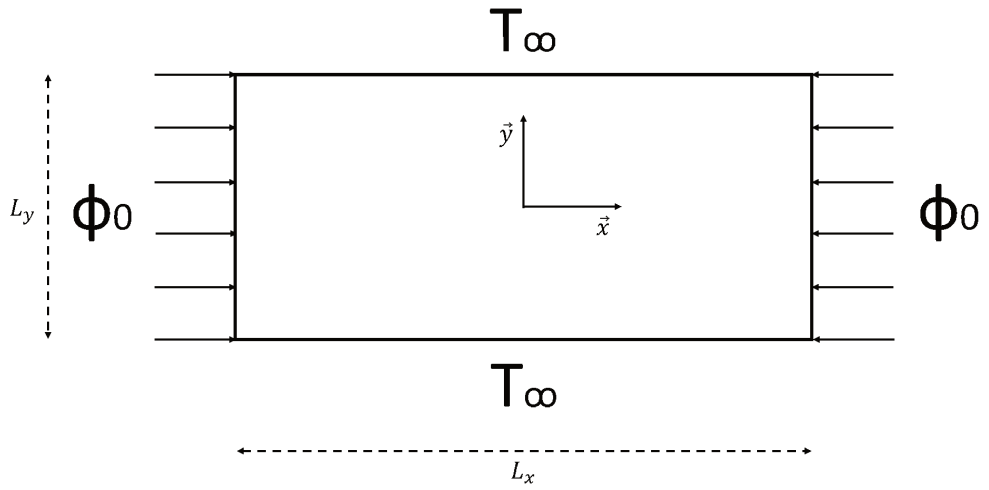


FIGURE 5.1 – Géométrie et conditions limites appliquées à la plaque mince

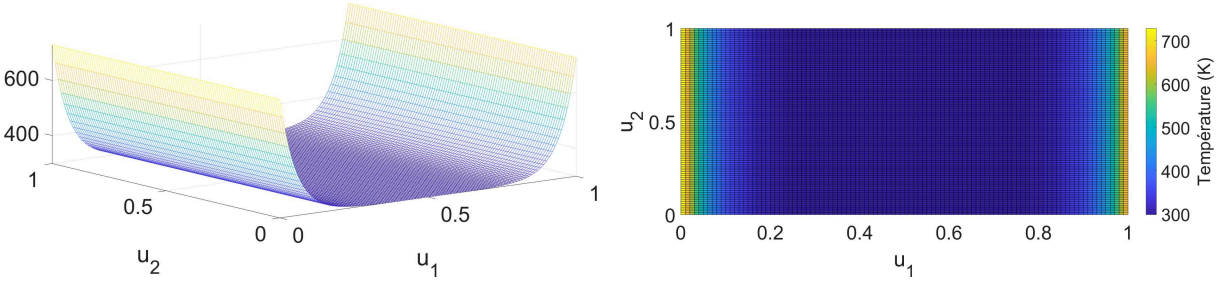
cibles est obtenue à l'aide d'une modélisation par éléments finis de la géométrie et des conditions limites de la fig. 5.1. Les simulations numériques ont été effectuées, pour différentes valeurs d'épaisseur t et de coefficient h, à l'aide du logiciel ANSYS et le maillage utilisé est un maillage structuré de 101×101 éléments *Shell131* avec un seul degré de liberté par nœud. La fig. 5.2 montre l'évolution du champ de température obtenu pour une valeur $t = 0.001\text{m}$ de l'épaisseur de la plaque et pour une valeur $h = 5\text{W/m}^2$ du coefficient de convection avec l'air. Comme nous l'avions remarqué dans l'introduction, les conditions aux limites imposées à la plaque, rendent l'évolution du champ de température indépendant de la position y sur la plaque (i.e. à t et h fixés, l'évolution de la température ne dépend que de x). On s'attendrait donc à ce que le nombre de points de contrôle dans la direction correspondante à y soit bien inférieur au nombre de points de contrôle dans la direction correspondante à x. Or, la plupart des méthodes existantes ne sont pas capables d'affecter moins de points de contrôle dans une direction dont l'influence sur la réponse est négligeable. L'algorithme HERO le permet grâce à l'exploration métaheuristique d'ERASMUS.

La base de données de points cibles est composée des valeurs de la température pour différentes valeurs des variables x, y, t et h, et est stockée sous la forme d'une matrice à quatre dimensions $\mathbf{Q}$, avec

$$Q_{s_1,s_2,s_3,s_4} = T(x_{s_1}, y_{s_2}, t_{s_3}, h_{s_4}), \quad s_k = 0, ..., r_k. \tag{5.4}$$

Les différents paramètres $u_{s_k}^{(k)}$ sont donnés par les relations suivantes :

$$u_{s_1}^{(1)} = \frac{x_{s_1} - \dfrac{-L_x}{2}}{\dfrac{L_x}{2} - \dfrac{-L_x}{2}}, \quad x_{s_1} \in \left[\frac{-L_x}{2}, \frac{L_x}{2}\right] \tag{5.5}$$



(a) Forme du champ de température T

(b) Champ de température T

FIGURE 5.2 – Champ de température de la plaque pour une épaisseur $t = 0.001\text{m}$ et un coefficient de convection $h = 5\text{W/m}^2$

$$u_{s_2}^{(2)} = \frac{y_{s_2} - \dfrac{-L_y}{2}}{\dfrac{L_y}{2} - \dfrac{-L_y}{2}}, \quad y_{s_2} \in \left[\frac{-L_y}{2}, \frac{L_y}{2}\right] \tag{5.6}$$

$$u_{s_3}^{(3)} = \frac{t_{s_3} - t_{min}}{t_{max} - t_{min}}, \quad t_{s_3} \in [t_{min}, t_{max}], \tag{5.7}$$

$$u_{s_4}^{(4)} = \frac{h_{s_4} - h_{min}}{h_{max} - h_{min}}, \quad h_{s_4} \in [h_{min}, h_{max}]. \tag{5.8}$$

Le maillage de la plaque étant structuré et ordonné, il est possible de définir les nombres de points cibles $r_k + 1$ dans chaque direction $k = 1, ..., 4$ et donc d'utiliser la stratégie de calcul, *dimension par dimension*, des coordonnées des points de contrôle (i.e. le problème d'obtention des coordonnées des points de contrôle de l'hypersurface est ramené à une succession de problèmes de calcul des coordonnées des points de contrôle de courbes NURBS). L'hypersurface NURBS permettant la représentation du champ de température est une application $H(\mathbf{u}) : \mathbb{R}^4 \to \mathbb{R}$. Les différents paramètres sans dimension $u^{(k)}$ sont donnés par les relations suivantes :

$$u^{(1)} = \frac{x - \dfrac{-L_x}{2}}{\dfrac{L_x}{2} - \dfrac{-L_x}{2}}, \quad x \in \left[\frac{-L_x}{2}, \frac{L_x}{2}\right], \tag{5.9}$$

$$u^{(2)} = \frac{y - \dfrac{-L_y}{2}}{\dfrac{L_y}{2} - \dfrac{-L_y}{2}}, \quad y \in \left[\frac{-L_y}{2}, \frac{L_y}{2}\right], \tag{5.10}$$

$$u^{(3)} = \frac{t - t_{min}}{t_{max} - t_{min}}, \quad t \in [t_{min}, t_{max}], \tag{5.11}$$

$$u^{(4)} = \frac{h - h_{min}}{h_{max} - h_{min}}, \quad h \in [h_{min}, h_{max}]. \tag{5.12}$$

5.2.2 Métamodèle généré par HERO

L'optimisation de l'hypersurface NURBS est réalisée par l'algorithme HERO. Comme nous l'avons expliqué au chapitre précédent, deux CNLPPs différents sont traités pendant l'optimisation. Dans un premier temps, le CNLPP traité par ERASMUS est de la forme suivante :

$$\min_{\mathbf{x}_\mathrm{I}} \phi(\mathbf{x}_\mathrm{I}) = a \times \frac{1}{4}\sum_{k=1}^{4}\frac{n_k}{r_k} + (1-a)\times\frac{1}{4}\sum_{k=1}^{4}\frac{p_k}{p_k^{max}}, \quad a = 0.5,$$

sujet à :

$$\begin{cases} \epsilon_T^{(max)}(\mathbf{x}_\mathrm{I}) \leq \epsilon_{max}^{(max)}, \\ \epsilon_T^{(moy)}(\mathbf{x}_\mathrm{I}) \leq \epsilon_{max}^{(moy)}, \\ 0 < U_{l_k}^{(k)} < 1, \quad l_k = p_k+1,...,m_k-p_k+1, \quad k=1,...,4, \\ U_{l_k}^{(k)} \leq U_{l_k+1}^{(k)}, \quad l_k = p_k+1,...,m_k-p_k, \quad k=1,...,4, \\ n_k - p_k \geq 0, \quad k=1,...,4, \\ n_{CP} \leq n_{TP}, \end{cases} \tag{5.13}$$

où $\mathbf{x}_\mathrm{I}$ est le vecteur contenant les variables d'optimisation, i.e. les 4 degrés p_k, les 4 tailles des *knot vectors* m_k+1 et les m_k-2p_k-1 valeurs non prédéfinies des *knot vectors*. L'erreur maximale d'approximation du champ de température $\epsilon_T^{(max)}$ est donnée par :

$$\epsilon_T^{(max)} = \max_{\mathbf{u}_{s_1,s_2,s_3,s_4}} \left(\frac{|H(\mathbf{u}_{s_1,s_2,s_3,s_4}) - T(x_{s_1},y_{s_2},t_{s_3},h_{s_4})|}{T_\mathrm{max}-T_\mathrm{min}} \right), \tag{5.14}$$

où $\mathbf{u}_{s_1,s_2,s_3,s_4} = \left(u_{s_1}^{(1)}, u_{s_2}^{(2)}, u_{s_3}^{(3)}, u_{s_4}^{(4)}\right)$ représente les coordonnées sans dimensions du point cible $T(x_{s_1},y_{s_2},t_{s_3},h_{s_4})$, $s_k = 0,...,r_k$, $k=1,...,4$, T_max est la température maximale sur l'ensemble des points cibles, tandis que T_{min} est la température minimale sur cet ensemble. L'erreur moyenne d'approximation, quant à elle, est donnée par :

$$\epsilon_T^{(moy)} = \frac{1}{n_{TP}}\sum_{s_1=0}^{r_1}\sum_{s_2=0}^{r_2}\sum_{s_3=0}^{r_3}\sum_{s_4=0}^{r_4}\frac{|H(\mathbf{u}_{s_1,s_2,s_3,s_4}) - T(x_{s_1},y_{s_2},t_{s_3},h_{s_4})|}{T_\mathrm{max}-T_\mathrm{min}}, \tag{5.15}$$

avec n_{TP} le nombre de points cibles. Les erreurs maximale et moyenne étant formulées par l'éq. (5.14) et l'éq. (5.15) respectivement, les seuils d'erreurs maximale et moyenne ont été fixés à $\epsilon_{max}^{(max)} = 2e^{-3}$ et $\epsilon_{max}^{(moy)} = 1e^{-4}$ respectivement. La structure d'un individu d'ERASMUS pour traiter ce problème, dans le cadre de l'approximation, est donnée à la fig. 5.3. Elle est composée d'une section standard, contenant les degrés p_1, p_2, p_3, p_4 et les nombres de chromosomes des différentes sections modulaires, ainsi que de quatre sections modulaires indépendantes, contenant les valeurs internes des *knot vectors* $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$, $\mathbf{U}^{(3)}$ et $\mathbf{U}^{(4)}$.

Les différents paramètres génétiques d'ERASMUS sont définis comme suit pour ce problème :

1. Plusieurs calculs ont été réalisés avec différents couples de nombre de populations et nombre d'individus $(n_{pop}, n_{ind}) = (2,120);(10,100)$.

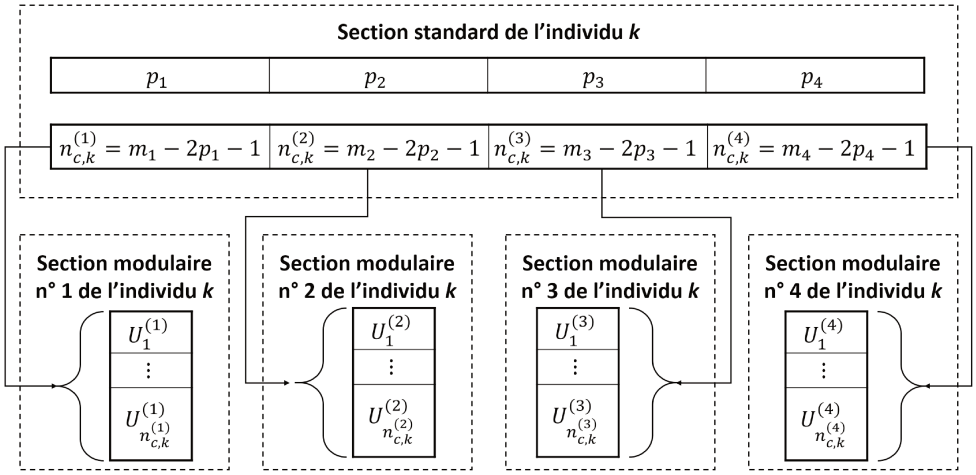


FIGURE 5.3 – Structure d'un individu d'ERASMUS pour le CNLPP (5.13)

2. Le critère d'arrêt sélectionné est le nombre d'itérations, fixé à $n_{gen} = 100$ pour ce cas.
3. La probabilité de croisement est fixée à la valeur par défaut $p_{cross} = 0.85$.
4. La probabilité de mutation des gènes est fixée à une valeur de $p_{mut} = 0.005$, l'exploration du domaine étant déjà grandement favorisée par le fait que les valeurs internes des *knot vectors* ne soient pas ordonnées dans les individus.
5. La probabilité de décalage est fixée à $p_{shift} = 0.5$. Ce paramètre a peu d'influence pour les mêmes raisons que la probabilité de mutation.
6. Le type de sélection retenu est la méthode de la *roulette*.
7. Le paramètre de *fitness pressure* est fixé à $f_{pres} = 1$, afin de ne pas brider l'exploration du domaine.
8. L'opérateur d'*élitisme* est actif pour nos études. Il n'existe que très peu de cas où cet opérateur n'est pas utile.
9. Le nombre de générations pendant lesquelles les populations évoluent de manière isolée les unes des autres est fixé à $I_{time} = 5$.
10. Le nombre de contraintes est, dans le cas du CNLPP (5.13), $n_{constr} = 2$.
11. Le nombre de chromosomes est variable.
12. Le nombre de composants modulaires est égal au nombre de *knot vectors*, soit $n_{modular} = 4$.
13. Les nombres minimums et maximums de chromosomes sont fixés à $n_{chrom,min}^{mod_i} = 0$ et $n_{chrom,max}^{mod_k} = r_k$ pour chaque type de module (i.e. $k = 1, ..., 4$).
14. Dans le cas de l'approximation, le nombre de chromosomes de la partie standard des individus est toujours $n_{chrom}^{std} = 1$.
15. Le nombre de gènes de la partie standard est égal au nombre de degrés à fixer, soit $n_{gene}^{std} = 4$.

16. Les nombres de gènes des chromosomes des différents composants modulaires sont, quelle que soit l'utilisation (interpolation ou approximation), $n_{gene}^{mod_k} = 1$, $k = 1, ..., 4$.
17. Le nombre de variables différentes est $n_{var} = 2$, dans le cas de l'approximation. Il s'agit des degrés p_k et des valeurs internes des *knot vectors* :
    - *Degré* :
        - *Nom de la variable* : degré
        - *Nature de la variable* : régulièrement discrétisée.
        - *La limite inférieure de la variable* : $p_k^{min} = 1$.
        - *La limite supérieure de la variable* : $p_k^{max} = 7$. Cette valeur est volontairement choisie relativement élevée afin de valider le comportement de la méthode vis à vis de la minimisation des degrés.
        - *Le pas de discrétisation* : 1
    - *Valeur de nœud* :
        - *Nom de la variable* : valeur de nœud
        - *Nature de la variable* : continue (*extended*), dans ce cas, une discrétisation est réalisée automatiquement par ERASMUS en fonction de la précision machine (i.e. pas de pas de discrétisation à fixer).
        - *La limite inférieure de la variable* : $0 + \epsilon$ avec $\epsilon = 0.0001$.
        - *La limite supérieure de la variable* : $1 - \epsilon$.

Après qu'ERASMUS ait atteint son critère d'arrêt, la solution qu'il renvoie sert de point de départ pour le CNLPP suivant :

$$\min_{\mathbf{x}_{\text{II}}} \psi^{(30)}(\mathbf{x}_{\text{II}}) = \frac{\left[\sum_{s_1=0}^{r_1} \cdots \sum_{s_4=0}^{r_4} \left(H_{\mathbf{x}_{\text{II}}}(\mathbf{u}_{s_1,s_2,s_3,s_4}) - T(x_{s_1}, y_{s_2}, t_{s_3}, h_{s_4})\right)^{30}\right]^{(\frac{1}{30})}}{\left[\sum_{s_1=0}^{r_1} \cdots \sum_{s_4=0}^{r_4} \left(H_0(\mathbf{u}_{s_1,s_2,s_3,s_4}) - T(x_{s_1}, y_{s_2}, t_{s_3}, h_{s_4})\right)^{30}\right]^{(\frac{1}{30})}},$$
$$\text{sujet à :} \tag{5.16}$$
$$\begin{cases} 0 < U_{l_k}^{(k)} < 1, \quad l_k = p_k + 1, ..., m_k - p_k + 1, \quad k = 1, ..., 4, \\ U_{l_k}^{(k)} \leq U_{l_k+1}^{(k)}, \quad l_k = p_k + 1, ..., m_k - p_k, \quad k = 1, ..., 4, \\ \omega_{i_1,...,i_4} \geq 0, \quad i_k = 0, ..., n_k, \end{cases}$$

où $H_{\mathbf{x}_{\text{II}}}$ est l'hypersurface NURBS obtenue avec le vecteur des variables d'optimisation $\mathbf{x}_{\text{II}}$ tandis que H_0 est l'hypersurface NURBS obtenue avec le vecteur des variables d'optimisation $\mathbf{x}_{\text{II}}^{(0)}$ fourni par ERASMUS. Le vecteur $\mathbf{x}_{\text{II}}$ regroupe les variables d'optimisation de la manière suivante :

$$\mathbf{x}_{\text{II}} = \left(U_{p_1+1}^{(1)}, \cdots, U_{m_1-p_1+1}^{(1)}, \cdots, U_{p_4+1}^{(4)}, \cdots, U_{m_4-p_4+1}^{(4)}, \boldsymbol{\Omega}\right). \tag{5.17}$$

Pour toutes les solutions renvoyées par ERASMUS, il s'est avéré que l'utilisation des poids n'était pas nécessaire pour ce cas d'application (i.e. $\boldsymbol{\Omega} = \emptyset$). C'est pourquoi le CNLPP (5.16) est directement traité par une méthode déterministe.

Comme pour ERASMUS, certains critères doivent être fixés dans la fonction *fmincon* de MATLAB. Ces paramètres sont, pour ce cas, fixés de la manière suivante :

- Le type d'algorithme déterministe utilisé est la méthode AS (*Active Set*) pour sa robustesse et son traitement des contraintes.
- *Les critères de convergence* :
  - *Nombre maximal d'évaluations de la fonction objectif* : Nous utilisons la valeur par défaut qui en pratique n'est souvent atteinte qu'en cas de divergence.
  - *Nombre maximal d'itérations* : K_{max}. De même que pour le nombre maximale d'évaluations de la fonction objectif, la valeur par défaut est rarement atteinte lorsque l'algorithme converge. La valeur par défaut de $K_{max} = 400$ est donc conservée.
  - *Faible amélioration de la fonction objectif* : σ_f. Les valeurs de la fonction objectif étant sans dimension, la valeur par défaut attribuée par MATLAB de $\sigma_f = 10^{-6}$ peut être utilisée.
  - *Variation négligeable des valeurs des variables d'optimisation* : σ_x. Les valeurs internes des *knot vectors* étant sans dimension, la valeur $\sigma_x = 10^{-6}$ attribuée par défaut de MATBLAB est utilisée.
  - *Norme du gradient de la fonction de Lagrange proche de 0* : σ_∇. Les valeurs de la fonction objectif et des fonctions contraintes étant sans dimension, la valeur par défaut de $\sigma_\nabla = 10^{-6}$ est utilisée.
- *Le seuil de tolérance de dépassement des contraintes*. Les valeurs des fonctions contraintes étant sans dimensions la valeur fixée par défaut de 10^{-6} est utilisée.

Le tableau 5.1 fournit les résultats obtenus de l'optimisation de l'hypersurface B-Spline par l'algorithme HERO avec les paramètres précédemment définis. Deux différents couples de nombre de populations et nombre d'individus ont été utilisés par HERO. On peut remarquer qu'augmenter le nombre de populations ne permet pas nécessairement d'obtenir une meilleur solution. Cela s'est avéré récurrent dans tous nos essais et nous conseillons, en pratique, d'utiliser seulement deux ou trois populations (dans le cas spécifique des hypersurfaces NURBS). Le tableau 5.1 fournit les erreurs maximale et moyenne d'approximation à chaque étape de HERO (i.e. les erreurs en sortie d'ERASMUS et en sortie de *fmincon*). Pour les deux couples (n_{pop}, n_{ind}), on peut remarquer que les seuils fixés ont été respectés. On peut également noter que le points de départ de *fmincon* pour le couple $(n_{pop}, n_{ind}) = (10, 100)$ était très proche du minimum local puisque l'erreur maximale d'approximation n'a quasiment pas évoluée (i.e. deux chiffres significatifs ne permettent pas de mettre en évidence les différences). L'erreur moyenne d'approximation a, cependant, était améliorée. On remarque également que la solution proposée par le couple $(n_{pop}, n_{ind}) = (2, 120)$ reflète la propriété d'invariance du champ de température par rapport à la coordonnées de la plaque y, le nombre de points de contrôle affectés dans cette direction étant très faible et le degré obtenu étant le degré minimal (i.e. $p_2 = 1$).

Les résultats d'une méthode itérative, utilisant les règles empiriques classiques, sont également présentés dans le tableau 5.1 afin d'être comparés à ceux de HERO. La méthode itérative nécessite un critère d'arrêt, l'erreur maximale d'approximation $\epsilon_{max}^{(max)}$. Dans ce cas, ce seuil a été fixé à $\epsilon_{max}^{(max)} = 1e^{-3}$, correspondant à l'arrondi supérieur de la meilleur solution obtenue par HERO

Variable		x	y	t	h	Nombre total de points	Erreur maximale (TPs)	Erreur moyenne (TPs)
$r_k + 1$ (TPs)		101	101	21	21	4 498 641	ϵ_T^{max}	ϵ_T^{moy}
Méthode itérative	p_k	2	2	2	2	1 025 100	$9.26e^{-4}$	$6.03e^{-6}$
	n_k	67	66	14	14		$1.87e^{-5}$ (*fmincon*)	$1.41e^{-6}$ (*fmincon*)
HERO ($n_{pop} = 2$, $n_{ind} = 120$, $n_{gen} = 100$)	p_k	4	1	3	4	12 480	$1.30e^{-3}$ (ERASMUS)	$6.06e^{-5}$ (ERASMUS)
	n_k	51	4	5	7		$8.25e^{-4}$ (*fmincon*)	$1.24e^{-5}$ (*fmincon*)
HERO ($n_{pop} = 10$, $n_{ind} = 100$, $n_{gen} = 100$)	p_k	4	1	4	3	27 840	$1.50e^{-3}$ (ERASMUS)	$7.34e^{-5}$ (ERASMUS)
	n_k	28	19	5	7		$1.50e^{-3}$ (*fmincon*)	$2.04e^{-5}$ (*fmincon*)

Tableau 5.1 – Comparaison des erreurs d'approximation des hypersurfaces B-Splines obtenues par HERO et une méthode itérative utilisant les règles empiriques pour les CNLPPs (5.13) et (5.16)

(i.e. $(n_{pop,ind}) = (2, 120)$). Pour cette valeur de seuil, la solution itérative a permis de réduire la taille de la base de données d'un facteur supérieur à 4, passant de 4 498 641 points cibles à 1 010 025 points de contrôle. Cependant, comme on peut le constater pour la meilleur solution de HERO (i.e. couple $(n_{pop}, n_{ind}) = (2, 120)$, non seulement la taille de la base de données a été réduite d'un facteur supérieur à 360, pour un nombre total de points de contrôle de 12 480, mais l'erreur maximale d'approximation obtenue est plus faible que celle de la méthode itérative. Il est à noter, cependant, que l'erreur moyenne de la méthode itérative est plus faible d'un ordre de grandeur par rapport à celle de HERO. Ceci est dû au grand nombre de points de contrôle utilisé. L'erreur moyenne d'approximation de la solution de HERO est, tout de même, inférieure d'un ordre de grandeur à l'erreur maximale d'approximation de celle-ci, et respecte le seuil $\epsilon_{max}^{(moy)} = 1e^{-4}$ fixé initialement. Afin de mettre en évidence le fait que la solution obtenue par la méthode itérative est loin d'être optimale, nous avons traité le CNLPP (5.16) avec les mêmes paramètres que ceux définis pour HERO, mais en utilisant la solution de la méthode itérative comme points de départ. On peut constater que l'optimisation des valeurs internes des *knot vectors* permet de gagner plus d'un ordre de grandeur sur l'erreur maximale d'approximation (l'erreur maximale d'approximation a été multipliée par un facteur de 0.02). Cela est dû au fait que les règles empiriques utilisées sont destinées à simplifier le calcul des coordonnées des points de contrôle et non à en réduire le nombre. En effet, ayant été élaborées dans le cadre des courbes et surfaces, le nombre total de points de contrôle représente rarement, dans ce cas, une limitation. Dans le cadre de la réduction de modèle en revanche, il y a un intérêt, pour un grand nombre d'applications, à être capable de réduire le nombre de données nécessaires.

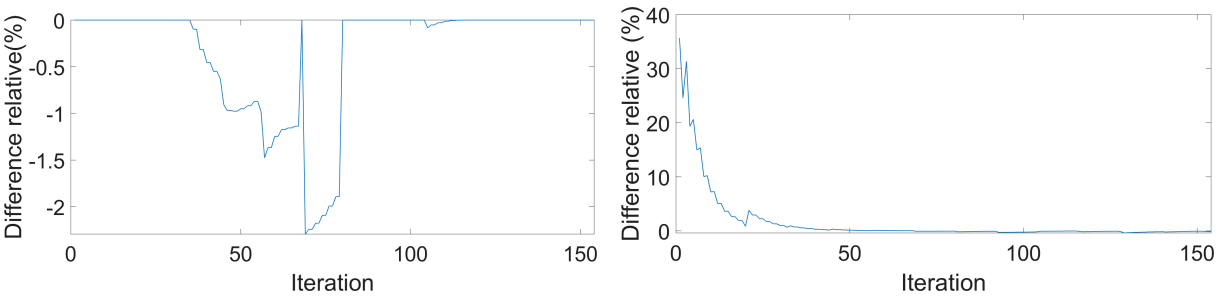
Les erreurs maximale et moyenne d'approximation respectant les seuils fixés initialement, les

Variable		x	y	t	h	Nombre total de points	Erreur maximale (validation)	Erreur moyenne (validation)
$r_k + 1$ (validation)		101	101	18	18	3 305 124	$\epsilon_{T,val}^{max}$	$\epsilon_{T,val}^{moy}$
Méthode itérative	p_k	2	2	2	2	1 025 100	$1.50e^{-3}$	$3.05e^{-5}$
	n_k	67	66	14	14		$1.25e^{-3}$ (*fmincon*)	$2.72e^{-5}$ (*fmincon*)
HERO ($n_{pop} = 2$, $n_{ind} = 120$, $n_{gen} = 100$)	p_k	4	1	3	4	12 480	$3.80e^{-3}$ (ERASMUS)	$1.83e^{-4}$ (ERASMUS)
	n_k	51	4	5	7		$1.70e^{-3}$ (*fmincon*)	$3.45e^{-5}$ (*fmincon*)
HERO ($n_{pop} = 10$, $n_{ind} = 100$, $n_{gen} = 100$)	p_k	4	1	4	3	27 840	$3.66e^{-3}$ (ERASMUS)	$2.12e^{-4}$ (ERASMUS)
	n_k	28	19	5	7		$2.89e^{-3}$ (*fmincon*)	$6.00e^{-5}$ (*fmincon*)

Tableau 5.2 – Comparaison des erreurs de validation des hypersurfaces B-Splines obtenues par HERO et une méthode itérative sur un ensemble de points de validation n'ayant pas été utilisé pour le calcul des coordonnées des points de contrôle

solutions de HERO doivent maintenant être validées. Le tableau 5.2 présente les erreurs maximale et moyenne de validation obtenues pour les différentes solutions proposées dans le tableau 5.1. L'ensemble des points de validation a été obtenu par la même simulation numérique que les points cibles, pour des valeurs de t et h différentes. Comme on peut le constater, toutes les solutions, obtenues par HERO ou par la méthode itérative, respectent les seuils d'erreurs maximale et moyenne fixés initialement sur l'ensemble des points de validation. Les erreurs de validation des solutions fournies par ERASMUS sont également présentées dans le tableau 5.2, afin de mettre en évidence, l'intérêt de l'utilisation de *fmincon*. En effet, pour les deux solutions proposées, on remarque que l'utilisation d'une méthode déterministe d'optimisation a permis, non seulement d'améliorer l'erreur maximale de validation, mais également l'erreur moyenne de validation et donc la qualité "globale" d'approximation de l'hypersurface $T(x, y, t, h)$. Concernant les résultats de la méthode itérative, on constate que les erreurs maximales et moyennes sur l'ensemble des points de validation sont très proches avec ou sans utilisation de la fonction *fmincon*. On remarque également que les erreurs de validation de la méthode itérative sont du même ordre de grandeur que celles de HERO alors que le nombre total de points de contrôle est 82 fois plus élevé pour le couple $(n_{pop}, n_{ind}) = (2, 120)$ et 36 fois pour le couple $(n_{pop}, n_{ind}) = (10, 100)$.

Le maillage de la plaque étant structuré, nous avons utilisé la méthode de calcul *dimension par dimension* dans l'algorithme HERO. Nous avons toutefois comparé les résultats obtenus entre la méthode *dimension par dimension* et la méthode hybride de calcul des coordonnées des points de contrôle. Nous avons, ici, une formulation générale pour les dimensions 1 et 2 (i.e. $L = 2$), liées à x et y respectivement, et une formulation *dimension par dimension* pour les dimensions 3 et 4, liées à t et h respectivement. Cela correspond à traiter le maillage obtenu par la simulation



(a) Écart relatif d'erreur maximale d'approximation (b) Écart relatif d'erreur moyenne d'approximation

FIGURE 5.4 – Différences relatives sur les erreurs d'approximation entre la méthode *dimension par dimension* généralisée à partir de [2] et la méthode hybride (cf. chapitre 3) avec $L = 2$

numérique indépendamment des autres directions de l'espace. Ainsi, même s'il est structuré dans notre cas, le maillage de la plaque pourrait être quelconque. Les valeurs d'erreurs ont été obtenues à l'aide de la méthode itérative, dans laquelle nous avons calculé les points de contrôle avec les deux méthodes. Le critère d'arrêt n'a, cependant, fait intervenir que les erreurs calculées avec la méthode *dimension par dimension* afin de comparer les résultats à ceux obtenus par HERO. Les erreurs sont données en pourcentage relatif l'une par rapport à l'autre et sont définies comme suit :

$$\begin{aligned} \Delta_{nb/hyb}^{(max)} &= \frac{\epsilon_{T,nb}^{max} - \epsilon_{T,hyb}^{max}}{\epsilon_{T,nb}^{max} + \epsilon_{T,hyb}^{max}}, \\ \Delta_{nb/hyb}^{(moy)} &= \frac{\epsilon_{T,nb}^{moy} - \epsilon_{T,hyb}^{moy}}{\epsilon_{T,nb}^{moy} + \epsilon_{T,hyb}^{moy}}, \end{aligned} \tag{5.18}$$

avec $\epsilon_{T,nb}^{max}$ et $\epsilon_{T,nb}^{moy}$ les erreurs maximale et moyenne obtenues par la méthode généralisée du *NURBS BOOK* [2], respectivement, $\epsilon_{T,hyb}^{max}$ et $\epsilon_{T,hyb}^{moy}$ les erreurs maximale et moyenne obtenues par la méthode hybride ($L = 2$), respectivement, $\Delta_{nb/hyb}^{(max)}$ l'écart relatif sur l'erreur maximale d'approximation et $\Delta_{nb/hyb}^{(moy)}$ l'écart relatif sur l'erreur moyenne entre les deux méthodes. La fig. 5.4 montre ces écarts. Il est à noter que l'erreur maximale d'approximation de la méthode *dimension par dimension* est toujours inférieur à celle de la méthode hybride. Cet écart n'excède cependant pas 3%. Concernant l'erreur moyenne, c'est la méthode hybride qui permet d'obtenir de meilleurs résultats. Cette différence est très notable pour de faibles nombre de points de contrôle (écart supérieur à 30%). Cependant, après 20 itérations, l'écart est inférieur à 5% et après 50 itérations, cet écart n'excède pas 0.2%. Les écarts étant très faibles (pour un "nombre convenable" de points de contrôle), nous conseillons, lorsque c'est possible, d'utiliser la méthode de calcul *dimension par dimension*. En effet, à nombre total de points de contrôle élevé, l'écart sur les erreurs est faible tandis que le temps de calcul des deux méthodes est très différent. Ces temps de calcul sont présentés à la fig. 5.5. Comme on peut le voir, le temps de calcul évolue très peu dans le cas de la méthode *dimension par dimension* et reste faible, alors que le temps de calcul de la méthode hybride augmente grandement avec le nombre total de points de contrôle.

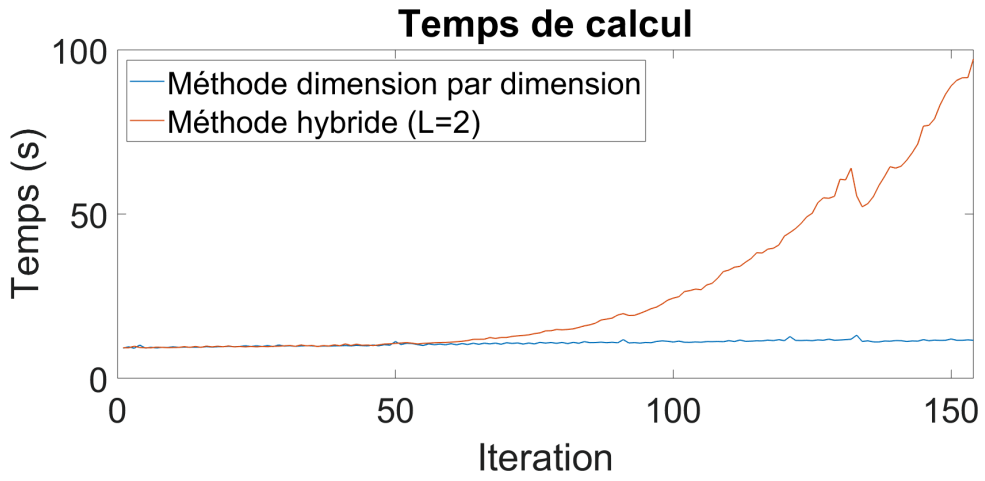


FIGURE 5.5 – Temps de calcul des coordonnées des points de contrôle pour l'algorithme généralisé de [2] (*dimension par dimension*) et la formulation hybride avec $L = 2$

Cela est dû au fait que la taille de la matrice $(\mathbf{N}^T\mathbf{N})$, dans le cas de la méthode hybride, est liée au produit $(n_1 + 1) \times (n_2 + 1)$, tandis que les matrices $\mathbf{N}_1$ et $\mathbf{N}_2$ sont liées à $n_1 + 1$ et $n_2 + 1$, respectivement, dans le cas de la méthode *dimension par dimension*.

5.3 Problème mécanique : champ de déplacement d'une plaque mince avec épaisseur et norme de l'effort appliqué variables $\mathbb{R}^4 \to \mathbb{R}^2$

5.3.1 Modélisation du problème

Notre méthode de réduction de modèle est appliquée, ici, à un problème mécanique. On souhaite approximer le champ de déplacement d'une plaque dont la géométrie et les conditions limites qui lui sont appliquées sont illustrées sur la fig. 5.6. La plaque est rectangulaire, de longueur $L_x = 0.2$m selon l'axe x, de longueur $L_y = 0.1$m selon l'axe y et d'épaisseur $t \in [t_{min}, t_{max}]$, avec $t_{min} = 0.001$m et $t_{max} = 0.01$m. Un effort dirigé selon la direction négative de l'axe y de norme $F \in [F_{min}, F_{max}]$, avec $F_{min} = 50$N et $F_{max} = 200$N, est appliqué au point $(x, y) = (\frac{L_x}{2}, 0)$ tandis que la plaque est encastrée en $x = \frac{-L_x}{2}$. L'objectif du métamodèle est de donner une représentation du champ de déplacement pour différentes valeurs des paramètres t et F, soit :

$$\begin{aligned} u_x &= f_x(x, y, t, F), \\ u_y &= f_y(x, y, t, F), \\ \theta_z &= f_z(x, y, t, F), \end{aligned} \tag{5.19}$$

où u_x et u_y sont respectivement les déplacements d'un point de la plaque selon les axes x et y, tandis que θ_z est la rotation d'un point de la plaque autour de l'axe z.

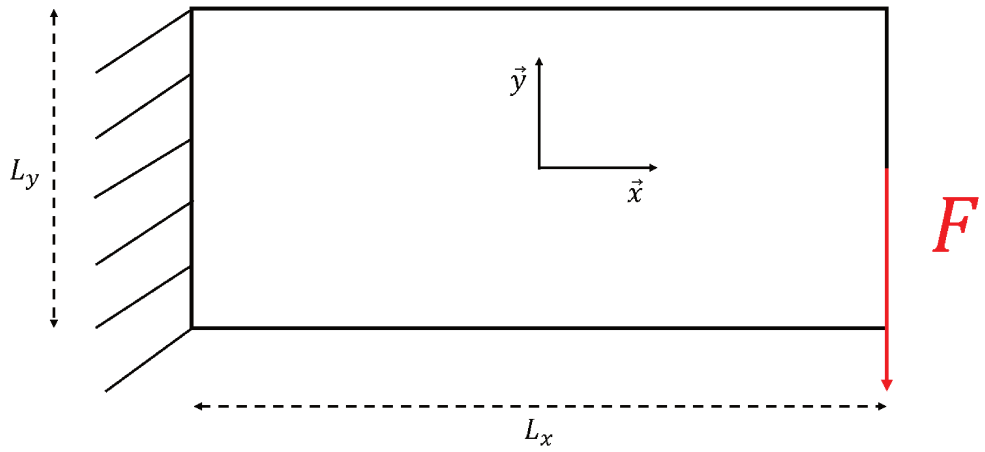


FIGURE 5.6 – Géométrie et conditions limites appliquées à la plaque mince

Dans ce contexte, l'épaisseur t et la norme de l'effort F représentent des paramètres de conception tandis que les variables x et y représentent un point de la plaque. La base de données de points cibles est obtenue à l'aide d'une modélisation par éléments finis de la géométrie et des conditions limites de la fig. 5.6. Les simulations numériques ont été effectuées, pour différentes valeurs d'épaisseur t et de norme de l'effort F, à l'aide du logiciel ANSYS et le maillage utilisé est un maillage structuré de 126×30 éléments *Shell181*. La fig. 5.7 montre l'évolution du champ de déplacement obtenu pour une valeur $t = 0.001$m d'épaisseur de la plaque et une norme d'effort $F = 50$N. Comme énoncé dans l'introduction de ce chapitre, le problème traité ici admet plusieurs sorties et, contrairement à d'autres techniques de réduction de modèle, la formulation NURBS permet d'obtenir toutes les sorties dans un seul modèle réduit, chaque sortie correspondant à une coordonnée supplémentaire des points de contrôle. L'hypersurface NURBS devrait posséder trois sorties, toutefois, pour le type d'élément choisi dans ANSYS, la rotation autour de l'axe z en un point est calculée à partir des déplacements $u_x(x, y)$ et $u_y(x, y)$, il n'est donc pas nécessaire d'inclure cette quantité dans notre métamodèle et le nombre de sorties est donc réduit à deux. C'est également pour cela que nous ne nous intéressons pas aux champs de contrainte ou de déformation, ces deux champs étant liés aux dérivées du champ de déplacement. Il n'est pas pertinent, quelle que soit la méthode de réduction de modèle employée, d'approximer par un même métamodèle, une quantité et sa dérivée. En revanche, la formulation NURBS permet d'obtenir très simplement les dérivées de l'hypersurface, et donc d'obtenir les champs de déformation et de contrainte à partir du champ de déplacement. De plus, même si le sens physique de la modélisation de l'effort appliqué à la plaque est discutable, le fort gradient de déplacement qu'il génère autour de son point d'application est une caractéristique intéressante. En effet, cela permet d'exhiber le comportement de l'hypersurface NURBS dans le cas où la fonction à modéliser présente de forts gradients localement.

La base de données de points cibles est composée des valeurs des champs de déplacement u_x et u_y pour différentes valeurs des variables x, y, t et F, et est stockée sous la forme d'une matrice

à quatre dimensions $\mathbf{Q}$, avec

$$\mathbf{Q}_{s_1,s_2,s_3,s_4} = \left(u_x(x_{s_1}, y_{s_2}, t_{s_3}, F_{s_4}), u_y(x_{s_1}, y_{s_2}, t_{s_3}, F_{s_4})\right), \quad s_k = 0, ..., r_k. \tag{5.20}$$

Les différents paramètres $u_{s_k}^{(k)}$ sont donnés par les relations suivantes :

$$u_{s_1}^{(1)} = \frac{x_{s_1} - \dfrac{-L_x}{2}}{\dfrac{L_x}{2} - \dfrac{-L_x}{2}}, \quad x_{s_1} \in \left[\frac{-L_x}{2}, \frac{L_x}{2}\right], \tag{5.21}$$

$$u_{s_2}^{(2)} = \frac{y_{s_2} - \dfrac{-L_y}{2}}{\dfrac{L_y}{2} - \dfrac{-L_y}{2}}, \quad y_{s_2} \in \left[\frac{-L_y}{2}, \frac{L_y}{2}\right], \tag{5.22}$$

$$u_{s_3}^{(3)} = \frac{t_{s_3} - t_{min}}{t_{max} - t_{min}}, \quad t_{s_3} \in [t_{min}, t_{max}], \tag{5.23}$$

$$u_{s_4}^{(4)} = \frac{F_{s_4} - F_{min}}{F_{max} - F_{min}}, \quad F_{s_4} \in [F_{min}, F_{max}]. \tag{5.24}$$

Le maillage de la plaque étant structuré et ordonné dans ce cas également, il est possible de définir les nombres de points cibles $r_k + 1$ dans chaque direction $k = 1, ..., 4$ et donc d'utiliser la stratégie de calcul, *dimension par dimension*, pour la calcul des coordonnées des points de contrôle. La fig. 5.7, qui présente l'évolution du champ de déplacement, à t et F fixés, permet de remarquer que le maillage utilisé dans le calcul numérique n'est pas réparti uniformément. En effet, les parties externes sont maillées plus "finement" que la bande centrale de la plaque. Cela permet d'insister sur le fait que l'utilisation de l'algorithme de calcul des coordonnées des points de contrôle, *dimension par dimension*, nécessite un ensemble de points cibles ordonnés mais qu'il n'est pas nécessaire qu'ils soient uniformément répartis. Il est donc envisageable de générer des points cibles avec des densités différentes suivant la région de l'espace considérée.

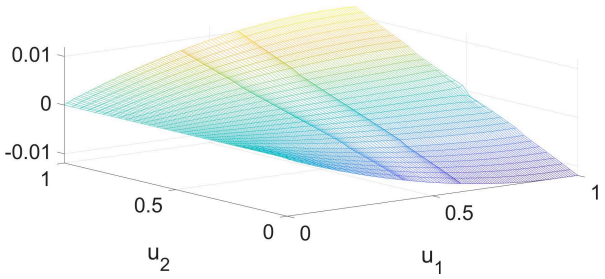
L'hypersurface NURBS permettant la représentation du champ de déplacement est une application $\mathbf{H}(\mathbf{u}) : \mathbb{R}^4 \to \mathbb{R}^2$. Les différents paramètres sans dimension $u^{(k)}$ sont donnés par les relations suivantes :

$$u^{(1)} = \frac{x - \dfrac{-L_x}{2}}{\dfrac{L_x}{2} - \dfrac{-L_x}{2}}, \quad x \in \left[\frac{-L_x}{2}, \frac{L_x}{2}\right], \tag{5.25}$$

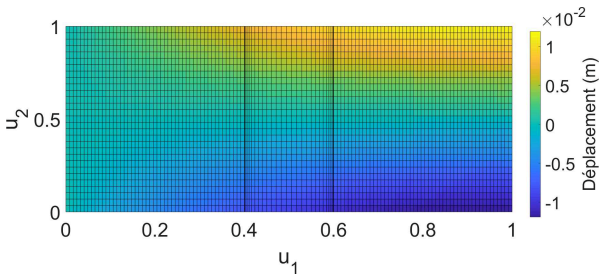
$$u^{(2)} = \frac{y - \dfrac{-L_y}{2}}{\dfrac{L_y}{2} - \dfrac{-L_y}{2}}, \quad y \in \left[\frac{-L_y}{2}, \frac{L_y}{2}\right], \tag{5.26}$$

$$u^{(3)} = \frac{t - t_{min}}{t_{max} - t_{min}}, \quad t \in [t_{min}, t_{max}], \tag{5.27}$$

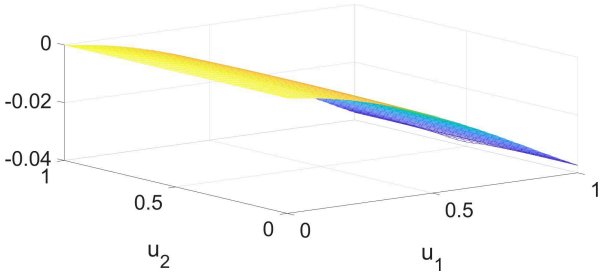
$$u^{(4)} = \frac{F - F_{min}}{F_{max} - F_{min}}, \quad F \in [F_{min}, F_{max}]. \tag{5.28}$$



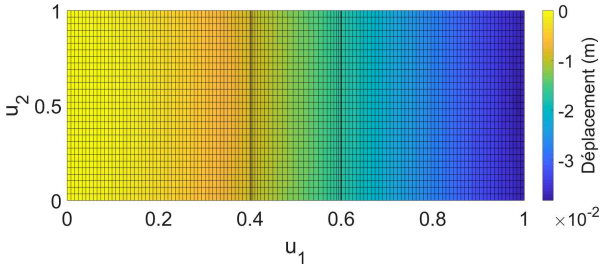
(a) Forme du champ de dépalcement u_x



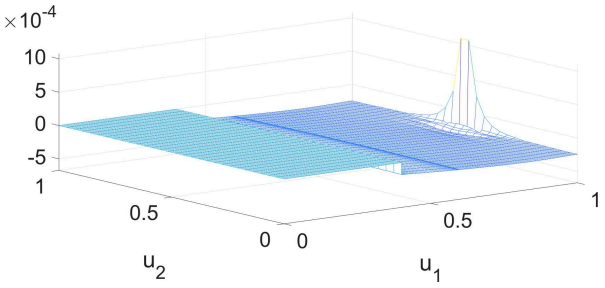
(b) Champ de dépalcement u_x



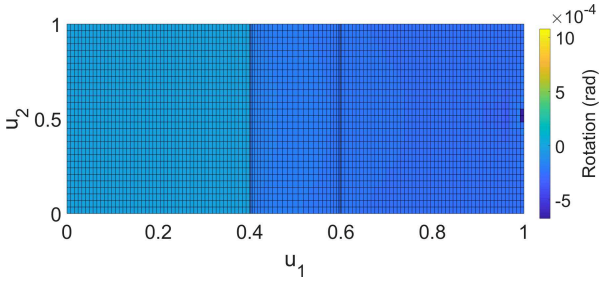
(c) Forme du champ de dépalcement u_y



(d) Champ déplacement u_y



(e) Forme du champ de rotation θ_z



(f) Champ de rotation θ_z

FIGURE 5.7 – Champ de déplacement de la plaque pour une épaisseur $t = 0.001$m et une norme de l'effort appliqué $F = 10$N

5.3.2 Métamodèle généré par HERO

Pour ce problème mécanique, le premier CNLPP, traité par ERASMUS, est de la forme suivante :

$$\min_{\mathbf{x}_\mathrm{I}} \phi(\mathbf{x}_\mathrm{I}) = a \times \frac{1}{4}\sum_{k=1}^{4}\frac{n_k}{r_k} + (1-a)\times\frac{1}{4}\sum_{k=1}^{4}\frac{p_k}{p_k^{max}}, \quad a = 0.5,$$

sujet à :

$$\begin{cases} \epsilon_{u_x}^{(max)}(\mathbf{x}_\mathrm{I}) \leq \epsilon_{u_x,max}^{(max)}, \\ \epsilon_{u_y}^{(max)}(\mathbf{x}_\mathrm{I}) \leq \epsilon_{u_y,max}^{(max)}, \\ \epsilon_{u_x}^{(moy)}(\mathbf{x}_\mathrm{I}) \leq \epsilon_{u_x,max}^{(moy)}, \\ \epsilon_{u_y}^{(moy)}(\mathbf{x}_\mathrm{I}) \leq \epsilon_{u_y,max}^{(moy)}, \\ 0 < U_{l_k}^{(k)} < 1, \quad l_k = p_k+1,...,m_k-p_k+1, \quad k=1,...,4, \\ U_{l_k}^{(k)} \leq U_{l_k+1}^{(k)}, \quad l_k = p_k+1,...,m_k-p_k, \quad k=1,...,4, \\ n_k - p_k \geq 0, \quad k=1,...,4, \\ n_{CP} \leq n_{TP}, \end{cases} \tag{5.29}$$

où $\mathbf{x}_\mathrm{I}$ est le vecteur contenant les variables d'optimisation, i.e. les 4 degrés p_k, les 4 tailles des *knot vectors* m_k+1 et les $m_k - 2p_k - 1$ valeurs internes des *knot vectors*. Les erreurs maximales d'approximation du champ de déplacement $\epsilon_{u_x}^{(max)}$ et $\epsilon_{u_y}^{(max)}$ sont données par :

$$\begin{aligned} \epsilon_{u_x}^{(max)} &= \max_{\mathbf{u}_{s_1,s_2,s_3,s_4}} \left(\frac{|H^{(1)}(\mathbf{u}_{s_1,s_2,s_3,s_4}) - u_x(x_{s_1},y_{s_2},t_{s_3},F_{s_4})|}{u_x^{\max}-u_x^{\min}} \right), \\ \epsilon_{u_y}^{(max)} &= \max_{\mathbf{u}_{s_1,s_2,s_3,s_4}} \left(\frac{|H^{(2)}(\mathbf{u}_{s_1,s_2,s_3,s_4}) - u_y(x_{s_1},y_{s_2},t_{s_3},F_{s_4})|}{u_y^{\max}-u_y^{\min}} \right), \end{aligned} \tag{5.30}$$

où $\mathbf{u}_{s_1,s_2,s_3,s_4} = \left(u_{s_1}^{(1)}, u_{s_2}^{(2)}, u_{s_3}^{(3)}, u_{s_4}^{(4)}\right)$ représente les coordonnées sans dimensions du point cible $\mathbf{Q}_{s_1,s_2,s_3,s_4}$, $s_k = 0,...,r_k$, $k=1,...,4$, $u_x^{\max}$ et $u_y^{\max}$ sont les déplacements maximums de u_x et u_y, respectivement, sur l'ensemble des points cibles, tandis que $u_x^{\min}$ et $u_y^{\min}$ sont respectivement les déplacements minimums de u_x et u_y sur cet ensemble. Les erreurs moyennes d'approximation, quant à elles, sont données par :

$$\begin{aligned} \epsilon_{u_x}^{(moy)} &= \frac{1}{n_{TP}}\sum_{s_1=0}^{r_1}\cdots\sum_{s_4=0}^{r_4}\frac{|H^{(1)}(\mathbf{u}_{s_1,s_2,s_3,s_4}) - u_x(x_{s_1},y_{s_2},t_{s_3},F_{s_4})|}{u_x^{\max}-u_x^{\min}}, \\ \epsilon_{u_y}^{(moy)} &= \frac{1}{n_{TP}}\sum_{s_1=0}^{r_1}\cdots\sum_{s_4=0}^{r_4}\frac{|H^{(2)}(\mathbf{u}_{s_1,s_2,s_3,s_4}) - u_y(x_{s_1},y_{s_2},t_{s_3},F_{s_4})|}{u_y^{\max}-u_y^{\min}}, \end{aligned} \tag{5.31}$$

avec n_{TP} le nombre de points cibles. Les erreurs maximales et moyennes étant formulées par l'éq. (5.30) et l'éq. (5.31) respectivement, et donc sans dimensions, les seuils d'erreurs maximales et moyennes ont été fixés à $\epsilon_{u_x,max}^{(max)} = \epsilon_{u_y,max}^{(max)} = 5e^{-3}$ et $\epsilon_{u_x,max}^{(moy)} = \epsilon_{u_y,max}^{(moy)} = 5e^{-4}$ respectivement.

La structure d'un individu d'ERASMUS pour traiter ce problème, dans le cadre de l'approximation, est identique à celle du cas précédent et est donnée par la fig. 5.3. Il est composé d'une section standard, contenant les degrés p_1, p_2, p_3, p_4, et les nombres de chromosomes des différentes sections modulaires, et de quatre sections modulaires indépendantes, contenant les valeurs internes des *knot vectors* $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$, $\mathbf{U}^{(3)}$ et $\mathbf{U}^{(4)}$.

Pour ce problème, les différents paramètres génétiques d'ERASMUS sont définis comme suit :

1. Le nombre de populations a été fixé à $n_{pop} = 3$
2. Le nombre d'individus a été fixé à $n_{ind} = 100$
3. Le critère d'arrêt est le nombre d'itérations, fixé à $n_{gen} = 100$ pour ce cas.
4. La probabilité de croisement est fixée à la valeur par défaut $p_{cross} = 0.85$.
5. La probabilité de mutation des gènes est fixé à une valeur de $p_{mut} = 0.005$, l'exploration du domaine étant déjà grandement favorisée par le fait que les valeurs internes des *knot vectors* ne soient pas ordonnées dans les individus.
6. La probabilité de décalage est fixée à $p_{shift} = 0.5$. Ce paramètre a peu d'influence pour les mêmes raisons que la probabilité de mutation.
7. Le type de sélection retenu est la méthode de la *roulette*.
8. Le paramètre de *fitness pressure* est fixé à $f_{pres} = 1$, afin de ne pas brider l'exploration du domaine.
9. L'opérateur d'*élitisme* est actif.
10. Le nombre de générations pendant lesquelles les populations évoluent de manière isolée les unes des autres est fixé à $I_{time} = 5$.
11. Le nombre de contraintes est, dans le cas du CNLPP (5.29), $n_{constr} = 4$.
12. Le nombre de chromosomes est variable.
13. Le nombre de composants modulaires est égal au nombre de *knot vectors*, soit $n_{modular} = 4$.
14. Les nombres minimums et maximums de chromosomes sont fixés à $n^{mod_k}_{chrom,min} = 0$ et $n^{mod_k}_{chrom,max} = r_k$ pour chaque composant modulaire (i.e. $k = 1, ..., 4$).
15. Dans le cas de l'approximation, le nombre de chromosomes de la partie standard des individus est toujours $n^{std}_{chrom} = 1$.
16. Le nombre de gènes de la partie standard est égal au nombre de degrés à fixer, soit $n^{std}_{gene} = 4$.
17. Les nombres de gènes des chromosomes des différents composants modulaires sont, quelle que soit l'utilisation (interpolation ou approximation), $n^{mod_k}_{gene} = 1$, $k = 1, ..., 4$.
18. Le nombre de variables différentes est $n_{var} = 2$, dans le cas de l'approximation. Il s'agit des degrés p_k et des valeurs internes des *knot vectors* :
    - *Degré* :
        - *Nom de la variable* : degré
        - *Nature de la variable* : régulièrement discrétisée.

- *La limite inférieure de la variable* : Le champ de déplacement devant être dérivé par rapport à x et y, liés à u_1 et u_2, afin d'obtenir les champs de déformation et de contrainte, les degrés minimums sont fixés à $p_1^{min} = 2$ et $p_2^{min} = 2$ dans ces directions. Les degrés minimums dans les autres directions sont fixés à $p_k^{min} = 1$, $k = 3, 4$.
- *La limite supérieure de la variable* : $p_k^{max} = 7$.
- *Le pas de discrétisation* : 1

— *Valeur de nœud* :

- *Nom de la variable* : valeur de nœud
- *Nature de la variable* : continue (*extended*).
- *La limite inférieure de la variable* : $0 + \epsilon$ avec $\epsilon = 0.0001$.
- *La limite supérieure de la variable* : $1 - \epsilon$.

Après qu'ERASMUS ait atteint son critère d'arrêt, la solution qu'il renvoie sert de point de départ pour le CNLPP suivant :

$$\begin{aligned}\min_{\mathbf{x}_{\text{II}}} \psi^{(30)}(\mathbf{x}_{\text{II}}) \quad = \quad & \frac{1}{2}\left[\frac{\left[\sum_{s_1=0}^{r_1}\cdots\sum_{s_4=0}^{r_4}\left(H^{(1)}_{\mathbf{x}_{\text{II}}}(\mathbf{u}_{s_1,s_2,s_3,s_4}) - u_x(x_{s_1}, y_{s_2}, t_{s_3}, F_{s_4})\right)^{30}\right]^{(\frac{1}{30})}}{\left[\sum_{s_1=0}^{r_1}\cdots\sum_{s_4=0}^{r_4}\left(H^{(1)}_{0}(\mathbf{u}_{s_1,s_2,s_3,s_4}) - u_x(x_{s_1}, y_{s_2}, t_{s_3}, F_{s_4})\right)^{30}\right]^{(\frac{1}{30})}}\right. \\ & \left. + \frac{\left[\sum_{s_1=0}^{r_1}\cdots\sum_{s_4=0}^{r_4}\left(H^{(2)}_{\mathbf{x}_{\text{II}}}(\mathbf{u}_{s_1,s_2,s_3,s_4}) - u_y(x_{s_1}, y_{s_2}, t_{s_3}, F_{s_4})\right)^{30}\right]^{(\frac{1}{30})}}{\left[\sum_{s_1=0}^{r_1}\cdots\sum_{s_4=0}^{r_4}\left(H^{(2)}_{0}(\mathbf{u}_{s_1,s_2,s_3,s_4}) - u_y(x_{s_1}, y_{s_2}, t_{s_3}, F_{s_4})\right)^{30}\right]^{(\frac{1}{30})}}\right],\end{aligned}$$

sujet à :

$$\begin{cases} 0 < U^{(k)}_{l_k} < 1, \quad l_k = p_k + 1, ..., m_k - p_k + 1, \quad k = 1, ..., 4, \\ U^{(k)}_{l_k} \leq U^{(k)}_{l_k+1}, \quad l_k = p_k + 1, ..., m_k - p_k, \quad k = 1, ..., 4, \\ \omega_{i_1,...,i_4} \geq 0, \quad i_k = 0, ..., n_k, \end{cases} \tag{5.32}$$

où $\mathbf{H}_{\mathbf{x}_{\text{II}}} = \left(H^{(1)}_{\mathbf{x}_{\text{II}}}, H^{(2)}_{\mathbf{x}_{\text{II}}}\right)$ est l'hypersurface NURBS obtenue avec le vecteur des variables d'optimisation $\mathbf{x}_{\text{II}}$ tandis que $\mathbf{H}_0 = \left(H^{(1)}_0, H^{(2)}_0\right)$ est l'hypersurface NURBS obtenue avec le vecteur des variables d'optimisation $\mathbf{x}^{(0)}_{\text{II}}$ fourni par ERASMUS. Le vecteur $\mathbf{x}_{\text{II}}$ regroupe les variables d'optimisation de la manière suivante :

$$\mathbf{x}_{\text{II}} = \left(U^{(1)}_{p_1+1}, \cdots, U^{(1)}_{m_1-p_1+1}, \cdots, U^{(4)}_{p_4+1}, \cdots, U^{(4)}_{m_4-p_4+1}, \boldsymbol{\Omega}\right). \tag{5.33}$$

De même que dans le cas précédent, toutes les solutions renvoyées par ERASMUS respectaient les contraintes d'erreurs d'approximation et l'introduction des poids n'était pas nécessaire pour ce cas d'application (i.e. $\mathbf{\Omega} = \emptyset$). C'est pourquoi le CNLPP (5.32) est directement traité par une méthode déterministe. Les paramètres de *fmincon* pour traité le CNLPP (5.32) sont les mêmes que pour l'exemple précédent.

Variable		x	y	t	F	Nombre total de points	Erreurs maximales (TPs)		Erreurs moyennes (TPs)	
$r_k + 1$ (TPs)		126	30	21	21	1 666 980	$\epsilon_{u_x}^{max}$	$\epsilon_{u_y}^{max}$	$\epsilon_{u_x}^{moy}$	$\epsilon_{u_y}^{moy}$
Méthode itérative	p_k	2	2	2	2	1 062 423	$2.36e^{-3}$	$2.05e^{-3}$	$4.04e^{-5}$	$6.91e^{-5}$
	n_k	108	26	18	18		$5.24e^{-4}$ (*fmincon*)	$8.46e^{-4}$ (*fmincon*)	$1.95e^{-5}$ (*fmincon*)	$4.25e^{-5}$ (*fmincon*)
HERO	p_k	4	4	1	1	43 500	$4.11e^{-3}$ (ERASMUS)	$3.20e^{-3}$ (ERASMUS)	$1.08e^{-4}$ (ERASMUS)	$2.09e^{-4}$ (ERASMUS)
	n_k	28	24	14	3		$2.33e^{-3}$ (*fmincon*)	$3.02e^{-3}$ (*fmincon*)	$9.42e^{-5}$ (*fmincon*)	$2.09e^{-4}$ (*fmincon*)

Tableau 5.3 – Comparaison des erreurs d'approximation des hypersurfaces B-Splines obtenues par HERO et une méthode itérative utilisant les règles empiriques pour les CNLPPs (5.29) et (5.32)

Les résultats obtenus par HERO et une méthode itérative, sont présentés dans le tableau 5.3. Pour les seuils d'erreur fixés précédemment, à savoir, $\epsilon_{u_x,max}^{(max)} = \epsilon_{u_y,max}^{(max)} = 5e^{-3}$ et $\epsilon_{u_x,max}^{(moy)} = \epsilon_{u_y,max}^{(moy)} = 5e^{-4}$, l'algorithme HERO a permis de réduire le nombre de données nécessaires d'un facteur 38, passant de 1 666 980 points cibles à 43 500 points de contrôle. En revanche, avec les mêmes critères d'arrêt pour la méthode itérative, le nombre total de points de contrôle atteint les 1 062 423. De même que pour le cas précédent, le CNLPP (5.32) a été traité en prenant la solution de la méthode itérative comme point de départ. On remarque que pour le nombre de points de contrôle obtenu, l'erreur maximale d'approximation peut, encore une fois, être grandement améliorée. Il est à noter, de plus, que l'erreur maximale d'approximation obtenue avec la solution de la méthode itérative comme point de départ n'est probablement pas la meilleure pouvant être atteinte pour ce nombre de points de contrôle et ces degrés. En effet, rien ne garantit qu'il s'agisse d'un "bon" point départ (i.e. permettant d'atteindre un minimum global pour ce nombre de points de contrôle et ces degrés). Toutefois, il ne fait aucun doute que la solution obtenue par la méthode itérative est loin d'être optimale.

5.4 Problème mécanique : champ de déplacement d'une plaque mince trouée avec épaisseur et rayon variables $\mathbb{R}^4 \to \mathbb{R}^2$

5.4.1 Modélisation du problème

Notre méthode de réduction de modèle est appliquée, ici également, à un problème mécanique. On souhaite approximer le champ de déplacement d'une plaque, trouée cette fois-ci, dont la géométrie et les conditions limites qui lui sont appliquées sont illustrées sur la fig. 5.8. La plaque est rectangulaire, de longueur $L_x = 0.2$m selon l'axe x, de longueur $L_y = 0.1$m selon l'axe y et d'épaisseur $t \in [t_{min}, t_{max}]$, avec $t_{min} = 0.001$m et $t_{max} = 0.01$m. La plaque est percée en son centre et ce trou circulaire a un rayon $R \in [R_{min}, R_{max}]$, avec $R_{min} = 0.005$m et $R_{max} = 0.025$m. Un effort dirigé selon la direction négative de l'axe y de norme $F_0 = 100$N, est appliqué au point $(x, y) = (\frac{L_x}{2}, 0)$ tandis que la plaque est encastrée en $y = \frac{-L_x}{2}$. L'objectif du métamodèle est de donner une représentation du champ de déplacement pour différentes valeurs des paramètres t et R, soit :

$$\begin{aligned} u_x &= f_x(x, y, t, R), \\ u_y &= f_y(x, y, t, R), \\ \theta_z &= f_z(x, y, t, R), \end{aligned} \tag{5.34}$$

où u_x et u_y sont respectivement les déplacements d'un point de la plaque selon l'axe x et l'axe y, tandis que θ_z est la rotation d'un point de la plaque autour de l'axe z.

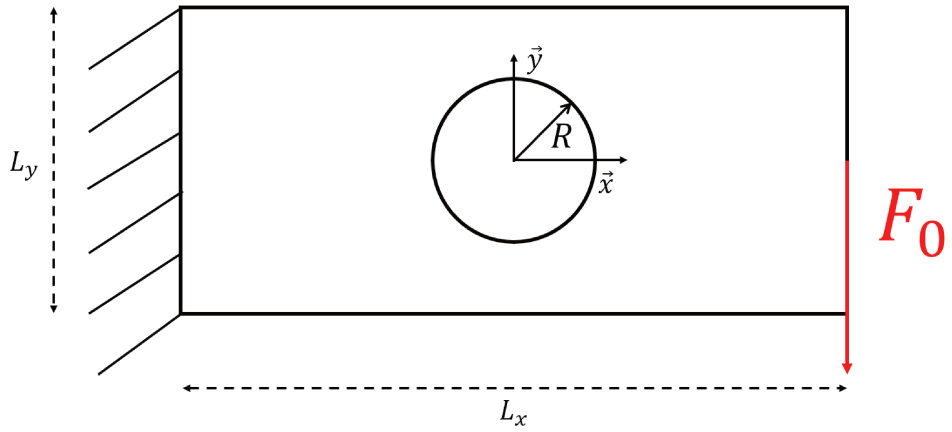


FIGURE 5.8 – Géométrie et conditions limites appliquées à la plaque mince

Dans ce contexte, l'épaisseur t et le rayon du trou R représentent des paramètres de conception tandis que les variables x et y représentent un point de la plaque. La base de données de points cibles est obtenue à l'aide d'une modélisation par éléments finis de la géométrie et des conditions limites de la fig. 5.8. Les simulations numériques ont été effectuées, pour différentes valeurs d'épaisseur t et de rayon R, à l'aide du logiciel ANSYS et le maillage utilisé est un maillage structuré d'éléments *Shell181*. Cependant, même si le maillage est structuré au sens des éléments finis, il n'est pas ordonné et ne permet pas d'utiliser l'algorithme de calcul des coordonnées

des points de contrôle *dimension par dimension*. La technique hybride (avec $L = 2$) a donc été employée ici avec une formulation générale pour les dimensions liées à x et y et une formulation dimension par dimension pour les autres. La fig. 5.9 montre l'évolution du champ de déplacement obtenu pour une valeur $t = 0.001$m d'épaisseur de la plaque et rayon $R = 0.005$m. Comme énoncé dans l'introduction de ce chapitre, le problème traité ici admet plusieurs sorties et la géométrie de la plaque varie. Les éléments utilisés dans le calcul par éléments finis étant les mêmes que dans le cas précédent, on ne s'intéressera qu'aux déplacements u_x et u_y, la rotation θ_z étant calculée à partir de ces deux déplacements. Les mêmes considérations s'appliquent aux champs de contrainte et déformation, liés aux dérivées du champ de déplacement, et donc facilement obtenus en dérivant l'hypersurface NURBS du métamodèle.

La difficulté de ce problème réside dans le fait que la géométrie de la plaque varie et que la morphologie de l'espace des points cibles varie avec les paramètres de conception. De plus, comme nous l'avons vu, les NURBS ont un comportement local intrinsèque et il est donc possible que certain points de contrôle ne possède pas de points cibles pour être paramétré (i.e. les points de contrôle situés dans le trou). Cela implique une étape préalable de filtrage, afin d'identifier les points de contrôle à exclure du calcul. Ce filtrage permet de garder les points de contrôle $P_{i_1,\dots,i_N}$ respectant la condition suivante :

$$\exists \mathbf{Q}_s \in \mathbf{Q},\ \mathbf{u}_s \in \left[U^{(1)}_{i_1}, U^{(1)}_{i_1+p_1+1}\right[\times \cdots \times \left[U^{(N)}_{i_N}, U^{(N)}_{i_N+p_N+1}\right[,\ s = 1, ..., n_{TP}, \tag{5.35}$$

où $\mathbf{Q}_s$ est le s^{e} point cible de l'ensemble des n_{TP} points cibles $\mathbf{Q}$, $\mathbf{u}_s = (u^{(1)}_s, ..., u^{(N)}_s)$ est le vecteur contenant les paramètres sans dimension relatif au point cible $\mathbf{Q}_s$. Cette formulation permet de conserver tous les points de contrôle pour lesquels il existe, au moins, un point cible appartenant à son support local. Cela permet, notamment, de traiter des problèmes à variables non-continues.

La base de données de points cibles est composée des valeurs des déplacements u_x et u_y pour différentes valeurs des variables x, y, t et R, et est stockée sous la forme d'une matrice à trois dimensions $\mathbf{Q}$, avec

$$\begin{gathered}\mathbf{Q}_{s_{1.2},s_3,s_4} = \left(u_x(x_{s_{1.2}}, y_{s_{1.2}}, t_{s_3}, R_{s_4}), u_y(x_{s_{1.2}}, y_{s_{1.2}}, t_{s_3}, R_{s_4})\right), \\ s_{1.2} = 0, ..., r_{1.2},\ s_k = 0, ..., r_k,\ k = 3, 4,\end{gathered} \tag{5.36}$$

avec $r_{1.2}+1$ le nombre de nœuds du maillage du calcul par éléments finis. Notons que ce nombre est constant dans nos simulations. S'il ne l'avait pas été, un stockage différent aurait été employé. Les différents paramètres $u^{(k)}_{s_k}$ sont donnés par les relations suivantes :

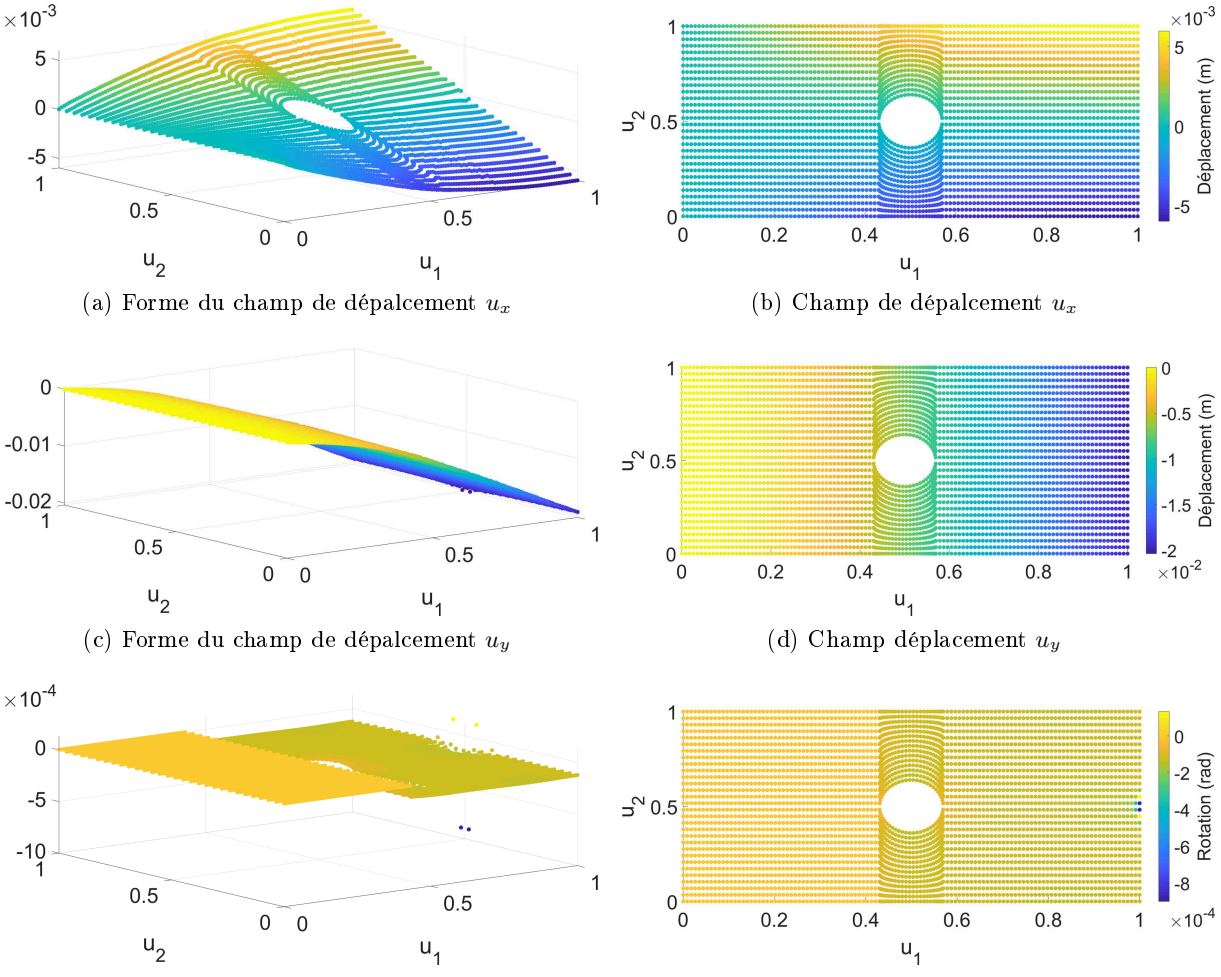
$$u^{(1)}_{s_{1.2}} = \frac{x_{s_{1.2}} - \dfrac{-L_x}{2}}{\dfrac{L_x}{2} - \dfrac{-L_x}{2}}, \quad x_{s_{1.2}} \in \left[\frac{-L_x}{2}, \frac{L_x}{2}\right], \tag{5.37}$$

$$u^{(2)}_{s_{1.2}} = \frac{y_{s_{1.2}} - \dfrac{-L_y}{2}}{\dfrac{L_y}{2} - \dfrac{-L_y}{2}}, \quad y_{s_{1.2}} \in \left[\frac{-L_y}{2}, \frac{L_y}{2}\right], \tag{5.38}$$

$$u_{s_3}^{(3)} = \frac{t_{s_3} - t_{min}}{t_{max} - t_{min}}, \quad t_{s_3} \in [t_{min}, t_{max}]\,, \tag{5.39}$$

$$u_{s_4}^{(4)} = \frac{R_{s_4} - R_{min}}{R_{max} - R_{min}}, \quad R_{s_4} \in [R_{min}, R_{max}]\,. \tag{5.40}$$

Le maillage ne permet pas, dans ce cas, d'obtenir un ensemble ordonné de points cibles dans les directions liées à x et y. La formulation générale est donc employée pour ces deux directions avec $L = 2$ (cf. eq. (4.85)).



(a) Forme du champ de dépalcement u_x

(b) Champ de dépalcement u_x

(c) Forme du champ de dépalcement u_y

(d) Champ déplacement u_y

(e) Forme du champ de rotation θ_z

(f) Champ de rotation θ_z

FIGURE 5.9 – Champ de déplacement de la plaque pour une épaisseur $t = 0.001$m et un rayon $R = 0.005$m

L'hypersurface NURBS permettant la représentation du champ de déplacement est une application $\mathbf{H}(\mathbf{u}) : \mathbb{R}^4 \to \mathbb{R}^2$. Les différents paramètres sans dimension $u^{(k)}$ sont donnés par les

relations suivantes :

$$u^{(1)} = \frac{x - \dfrac{-L_x}{2}}{\dfrac{L_x}{2} - \dfrac{-L_x}{2}}, \quad x \in \left[\frac{-L_x}{2}, \frac{L_x}{2}\right], \tag{5.41}$$

$$u^{(2)} = \frac{y - \dfrac{-L_y}{2}}{\dfrac{L_y}{2} - \dfrac{-L_y}{2}}, \quad y \in \left[\frac{-L_y}{2}, \frac{L_y}{2}\right], \tag{5.42}$$

$$u^{(3)} = \frac{t - t_{min}}{t_{max} - t_{min}}, \quad t \in [t_{min}, t_{max}], \tag{5.43}$$

$$u^{(4)} = \frac{R - R_{min}}{R_{max} - R_{min}}, \quad R \in [R_{min}, R_{max}]. \tag{5.44}$$

5.4.2 Métamodèle généré par HERO

Pour ce problème mécanique, le premier CNLPP, traité par ERASMUS, est de la forme suivante :

$$\min_{\mathbf{x}_\mathrm{I}} \phi(\mathbf{x}_\mathrm{I}) = a \times \frac{\sum_{k=1}^{4}(n_k+1)}{(r_{1.2}+1)\times\prod_{k=3}^{4}(r_k+1)} + (1-a)\times\frac{1}{4}\sum_{k=1}^{4}\frac{p_k}{p_k^{max}}, \quad a = 0.5,$$

sujet à :

$$\begin{cases}
\epsilon_{u_x}^{(max)}(\mathbf{x}_\mathrm{I}) \leq \epsilon_{u_x,max}^{(max)}, \\
\epsilon_{u_y}^{(max)}(\mathbf{x}_\mathrm{I}) \leq \epsilon_{u_y,max}^{(max)}, \\
\epsilon_{u_x}^{(moy)}(\mathbf{x}_\mathrm{I}) \leq \epsilon_{u_x,max}^{(moy)}, \\
\epsilon_{u_y}^{(moy)}(\mathbf{x}_\mathrm{I}) \leq \epsilon_{u_y,max}^{(moy)}, \\
0 < U_{l_k}^{(k)} < 1, \quad l_k = p_k+1, ..., m_k - p_k + 1, \quad k = 1, ..., 4, \\
U_{l_k}^{(k)} \leq U_{l_k+1}^{(k)}, \quad l_k = p_k+1, ..., m_k - p_k, \quad k = 1, ..., 4, \\
n_k - p_k \geq 0, \quad k = 1, ..., 4, \\
n_{CP} \leq n_{TP},
\end{cases} \tag{5.45}$$

où $\mathbf{x}_\mathrm{I}$ est le vecteur contenant les variables d'optimisation, i.e. les 4 degrés p_k, les 4 tailles des *knot vectors* $m_k + 1$ et les $m_k - 2p_k - 1$ valeurs non prédéfinies des *knot vectors*. Les erreurs maximales d'approximation du champ de déplacement $\epsilon_{u_x}^{(max)}$ et $\epsilon_{u_y}^{(max)}$ sont données par :

$$\begin{aligned}
\epsilon_{u_x}^{(max)} &= \max_{\mathbf{u}_{s_{1.2},s_3,s_4}} \left(\frac{|\mathbf{H}(\mathbf{u}_{s_{1.2},s_3,s_4}) - u_x(x_{s_{1.2}}, y_{s_{1.2}}, t_{s_3}, R_{s_4})|}{u_x^{\max} - u_x^{\min}} \right), \\
\epsilon_{u_y}^{(max)} &= \max_{\mathbf{u}_{s_{1.2},s_3,s_4}} \left(\frac{|\mathbf{H}(\mathbf{u}_{s_{1.2},s_3,s_4}) - u_y(x_{s_{1.2}}, y_{s_{1.2}}, t_{s_3}, R_{s_4})|}{u_y^{\max} - u_y^{\min}} \right),
\end{aligned} \tag{5.46}$$

où $\mathbf{u}_{s_{1.2},s_3,s_4} = \left(u^{(1)}_{s_{1.2}}, u^{(2)}_{s_{1.2}}, u^{(3)}_{s_3}, u^{(4)}_{s_4}\right)$ représente les coordonnées sans dimensions du point cible $\mathbf{Q}_{s_{1.2},s_3,s_4}$, $s_{1.2} = 0, ..., r_{1.2}$, $s_k = 0, ..., r_k$, $k = 3, 4$, $u_x^{\max}$ et $u_y^{\max}$ sont les déplacements maximums de u_x et u_y, respectivement, sur l'ensemble des points cibles, tandis que $u_x^{\min}$ et $u_y^{\min}$ sont respectivement les déplacements minimums de u_x et u_y sur cet ensemble. Les erreurs moyennes d'approximation, quant à elles, sont données par :

$$\begin{aligned} \epsilon^{(moy)}_{u_x} &= \frac{1}{n_{TP}} \sum_{s_{1.2}=0}^{r_{1.2}} \sum_{s_3=0}^{r_3} \sum_{s_4=0}^{r_4} \frac{|\mathbf{H}(\mathbf{u}_{s_{1.2},s_3,s_4}) - u_x(x_{s_{1.2}}, y_{s_{1.2}}, t_{s_3}, R_{s_4})|}{u_x^{\max} - u_x^{\min}}, \\ \epsilon^{(moy)}_{u_y} &= \frac{1}{n_{TP}} \sum_{s_{1.2}=0}^{r_{1.2}} \sum_{s_3=0}^{r_3} \sum_{s_4=0}^{r_4} \frac{|\mathbf{H}(\mathbf{u}_{s_{1.2},s_3,s_4}) - u_y(x_{s_{1.2}}, y_{s_{1.2}}, t_{s_3}, R_{s_4})|}{u_y^{\max} - u_y^{\min}}, \end{aligned} \tag{5.47}$$

avec n_{TP} le nombre total de points cibles. Les seuils d'erreurs maximales et moyennes ont été fixés à $\epsilon^{(max)}_{u_x,max} = \epsilon^{(max)}_{u_y,max} = 3e^{-2}$ et $\epsilon^{(moy)}_{u_x,max} = \epsilon^{(moy)}_{u_y,max} = 1e^{-3}$ respectivement. La structure d'un individu d'ERASMUS pour traiter ce problème, dans le cadre de l'approximation, est identique à celle des deux cas précédent et est donnée par la fig. 5.3. Il est composé d'une section standard, contenant les degrés p_1, p_2, p_3, p_4 et les nombres de chromosomes des différentes sections modulaires, ainsi que de quatre sections modulaires indépendantes, contenant les valeurs internes des *knot vectors* $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$, $\mathbf{U}^{(3)}$ et $\mathbf{U}^{(4)}$.

Pour ce problème, les différents paramètres génétiques d'ERASMUS sont définis comme suit :

1. Plusieurs calculs ont été réalisés avec différents couples de nombres de populations et d'individus $(n_{pop}, n_{ind}) = (3, 100); (3, 200)$.
2. Le critère d'arrêt est le nombre d'itérations, fixé à $n_{gen} = 100$ pour ce cas.
3. La probabilité de croisement est fixée à la valeur par défaut $p_{cross} = 0.85$.
4. La probabilité de mutation des gènes est fixée à une valeur de $p_{mut} = 0.005$, l'exploration du domaine étant déjà grandement favorisée par le fait que les valeurs internes des *knot vectors* ne soient pas ordonnées dans les individus.
5. La probabilité de décalage est fixée à $p_{shift} = 0.5$. Ce paramètre a peu d'influence pour les mêmes raisons que la probabilité de mutation.
6. Le type de sélection retenu est la méthode de la *roulette*.
7. Le paramètre de *fitness pressure* est fixé à $f_{pres} = 1$, afin de ne pas brider l'exploration du domaine.
8. L'opérateur d'*élitisme* est actif.
9. Le nombre de générations pendant lesquelles les populations évoluent de manière isolée les unes des autres est fixé à $I_{time} = 5$.
10. Le nombre de contraintes est, dans le cas du CNLPP (5.45), $n_{constr} = 4$.
11. Le nombre de chromosomes est variable.
12. Le nombre de composants modulaires est égal au nombre de *knot vectors*, soit $n_{modular} = 4$.
13. Les nombres minimums et maximums de chromosomes sont fixés à $n^{mod_i}_{chrom,min} = 0$ et $n^{mod_k}_{chrom,max} = r_k$ pour chaque composant modulaire $k = 3, 4$. Dans ce cas, r_1 et r_2 ne pouvant être défini, les nombres de chromosomes maximums sont donnés par $n^{mod_1}_{chrom,max} = n^{mod_2}_{chrom,max} = \lceil\sqrt{r_{1.2}}\rceil$, avec $\lceil\cdot\rceil$ la fonction partie entière par excès.

14. Dans le cas de l'approximation, le nombre de chromosomes de la partie standard des individus est toujours $n_{chrom}^{std} = 1$.

15. Le nombre de gènes de la partie standard est égal au nombre de degrés à fixer, soit $n_{gene}^{std} = 4$.

16. Les nombres de gènes des chromosomes des différents composants modulaires sont, quelle que soit l'utilisation (interpolation ou approximation), $n_{gene}^{mod_k} = 1$, $k = 1, ..., 4$.

17. Le nombre de variables différentes est $n_{var} = 2$, dans le cas de l'approximation. Il s'agit des degrés p_k et des valeurs internes des *knot vectors* :

    — *Degré* :

    - *Nom de la variable* : degré
    - *Nature de la variable* : régulièrement discrétisée (*regular_discr*).
    - *La limite inférieure de la variable* : Le champ de déplacement devant être dérivé par rapport à x et y, liés à u_1 et u_2, afin d'obtenir les champs de déformation et de contrainte, les degrés minimums sont fixés à $p_1^{min} = 2$ et $p_2^{min} = 2$ dans ces directions. Les degrés minimums dans les autres directions sont fixés à $p_k^{min} = 1$, $k = 3, 4$.
    - *La limite supérieure de la variable* : $p_k^{max} = 7$.
    - *Le pas de discrétisation* : 1

    — *Valeur de nœud* :

    - *Nom de la variable* : valeur de nœud
    - *Nature de la variable* : continue (*extended*).
    - *La limite inférieure de la variable* : $0 + \epsilon$ avec $\epsilon = 0.0001$.
    - *La limite supérieure de la variable* : $1 - \epsilon$.

Après qu'ERASMUS ait atteint son critère d'arrêt, la solution qu'il renvoie sert de point de

départ pour le CNLPP suivant :

$$\min_{\mathbf{x}_{\text{II}}} \quad \psi^{(30)}(\mathbf{x}_{\text{II}}) =$$

$$\frac{1}{2}\left[\frac{\left[\sum_{s_{1.2}=0}^{r_{1.2}}\sum_{s_3=0}^{r_3}\sum_{s_4=0}^{r_4}\left(H^{(1)}_{\mathbf{x}_{\text{II}}}(\mathbf{u}_{s_{1.2},s_3,s_4}) - u_x(x_{s_{1.2}}, y_{s_{1.2}}, t_{s_3}, R_{s_4})\right)^{30}\right]^{(\frac{1}{30})}}{\left[\sum_{s_{1.2}=0}^{r_{1.2}}\sum_{s_3=0}^{r_3}\sum_{s_4=0}^{r_4}\left(H^{(1)}_{0}(\mathbf{u}_{s_{1.2},s_3,s_4}) - u_x(x_{s_{1.2}}, y_{s_{1.2}}, t_{s_3}, R_{s_4})\right)^{30}\right]^{(\frac{1}{30})}}\right.$$

$$\left. + \frac{\left[\sum_{s_{1.2}=0}^{r_{1.2}}\sum_{s_3=0}^{r_3}\sum_{s_4=0}^{r_4}\left(H^{(2)}_{\mathbf{x}_{\text{II}}}(\mathbf{u}_{s_{1.2},s_3,s_4}) - u_y(x_{s_{1.2}}, y_{s_{1.2}}, t_{s_3}, R_{s_4})\right)^{30}\right]^{(\frac{1}{30})}}{\left[\sum_{s_{1.2}=0}^{r_{1.2}}\sum_{s_3=0}^{r_3}\sum_{s_4=0}^{r_4}\left(H^{(2)}_{0}(\mathbf{u}_{s_{1.2},s_3,s_4}) - u_y(x_{s_{1.2}}, y_{s_{1.2}}, t_{s_3}, R_{s_4})\right)^{30}\right]^{(\frac{1}{30})}}\right],$$

$$\text{sujet à :}$$

$$\begin{cases} 0 < U^{(k)}_{l_k} < 1, \quad l_k = p_k + 1, ..., m_k - p_k + 1, \quad k = 1, ..., 4, \\ U^{(k)}_{l_l} \leq U^{(k)}_{l_k+1}, \quad l_k = p_k + 1, ..., m_k - p_k, \quad k = 1, ..., 4, \\ \omega_{i_1,...,i_4} \geq 0, \quad i_k = 0, ..., n_k, \end{cases} \tag{5.48}$$

où $\mathbf{H}_{\mathbf{x}_{\text{II}}} = \left(H^{(1)}_{\mathbf{x}_{\text{II}}}, H^{(2)}_{\mathbf{x}_{\text{II}}}\right)$ est l'hypersurface NURBS obtenue avec le vecteur des variables d'optimisation $\mathbf{x}_{\text{II}}$ tandis que $\mathbf{H}_0 = \left(H^{(1)}_0, H^{(2)}_0\right)$ est l'hypersurface NURBS obtenue avec le vecteur des variables d'optimisation $\mathbf{x}^{(0)}_{\text{II}}$ fourni par ERASMUS. Le vecteur $\mathbf{x}_{\text{II}}$ regroupe les variables d'optimisation de la manière suivante :

$$\mathbf{x}_{\text{II}} = \left(U^{(1)}_{p_1+1}, \cdots, U^{(1)}_{m_1-p_1+1}, \cdots, U^{(4)}_{p_4+1}, \cdots, U^{(4)}_{m_4-p_4+1}, \boldsymbol{\Omega}\right). \tag{5.49}$$

De même que dans le cas précédent, toutes les solutions renvoyées par ERASMUS respectaient les contraintes d'erreurs d'approximation et l'introduction des poids n'était pas nécessaire pour ce cas d'application (i.e. $\boldsymbol{\Omega} = \emptyset$). C'est pourquoi le CNLPP (5.48) est directement traité par une méthode déterministe. Les paramètres de *fmincon* pour traité le CNLPP (5.48) sont identiques à ceux utilisés dans les exemples précédents.

Les résultats obtenus par HERO pour les deux couples (n_{pop}, n_{ind}) précédemment définis, sont présentés dans le tableau 5.4. On remarque que, dans les deux cas, la solution renvoyée par ERASMUMS était "proche" du minimum local obtenu par la fonction *fmincon*. Pour les seuils d'erreur fixés précédemment, le nombre de données a été réduit d'un facteur supérieur à 162 pour le couple $(n_{pop}, n_{ind}) = (3, 100)$ et d'un facteur supérieur à 754 pour le couple $(n_{pop}, n_{ind}) = (3, 200)$. Les résultats d'une méthode itérative sont également présentés dans le tableau 5.4. Dans le premier cas, le critère d'arrêt a été fixé par la précision de l'hypersurface du couple $(n_{pop}, n_{ind}) = (3, 100)$, tandis que la seconde a eu comme critère la précision obtenue

Variable		x	y	t	r	Nombre total de points	Erreurs maximales (TPs)		Erreurs moyennes (TPs)	
$r_k + 1$ (TPs)		4 681		21	21	2 064 321	$\epsilon_{u_x}^{max}$	$\epsilon_{u_y}^{max}$	$\epsilon_{u_x}^{moy}$	$\epsilon_{u_y}^{moy}$
Méthode itérative	p_k	2	2	2	2	1 334 256	$8.27e^{-3}$	$7.54e^{-3}$	$1.41e^{-4}$	$6.84e^{-5}$
	n_k	131	27	18	18		$7.36e^{-3}$ (*fmincon*)	$2.38e^{-3}$ (*fmincon*)	$1.31e^{-4}$ (*fmincon*)	$5.76e^{-5}$ (*fmincon*)
HERO ($n_{pop} = 3$, $n_{ind} = 100$, $n_{gen} = 100$)	p_k	1	1	4	1	12 726	$1.30e^{-2}$ (ERASMUS)	$8.13e^{-3}$ (ERASMUS)	$8.44e^{-4}$ (ERASMUS)	$5.12e^{-4}$ (ERASMUS)
	n_k	100	8	6	1		$1.10e^{-2}$ (*fmincon*)	$8.13e^{-3}$ (*fmincon*)	$7.99e^{-4}$ (*fmincon*)	$5.08e^{-4}$ (*fmincon*)
Méthode itérative	p_k	2	2	2	2	811 512	$1.21e^{-2}$	$2.12e^{-2}$	$2.24e^{-4}$	$9.10e^{-5}$
	n_k	116	23	16	16		$6.79e^{-3}$ (*fmincon*)	$2.00e^{-3}$ (*fmincon*)	$2.20e^{-4}$ (*fmincon*)	$9.10e^{-5}$ (*fmincon*)
HERO ($n_{pop} = 3$, $n_{ind} = 200$, $n_{gen} = 100$)	p_k	1	1	4	1	2 736	$2.08e^{-2}$ (ERASMUS)	$2.45e^{-2}$ (ERASMUS)	$9.59e^{-4}$ (ERASMUS)	$8.04e^{-4}$ (ERASMUS)
	n_k	11	18	5	1		$2.05e^{-2}$ (*fmincon*)	$2.14e^{-2}$ (*fmincon*)	$9.54e^{-4}$ (*fmincon*)	$7.31e^{-4}$ (*fmincon*)

Tableau 5.4 – Comparaison des erreurs d'approximation des hypersurfaces B-Splines obtenues par HERO et une méthode itérative utilisant les règles empiriques pour les CNLPPs (5.45) et (5.48)

par l'hypersurface du couple $(n_{pop}, n_{ind}) = (3, 200)$. Dans les deux cas, on remarque que le nombre total de points de contrôle nécessaire à la méthode itérative pour obtenir la précision de l'hypersurface générée par HERO est bien plus élevée. Nous avons également, comme pour le cas précédent, traiter le CNLPP (5.48) avec les solutions de la méthode itérative comme point de départ. On remarque que pour les nombres de points de contrôle obtenus, l'erreur maximale d'approximation peut, encore une fois, être améliorée. Comme nous le faisions remarquer dans le cas précédent, l'erreur maximale d'approximation obtenue avec la solution de la méthode itérative comme point de départ n'est probablement pas la meilleure pouvant être atteinte pour ce nombre de points de contrôle et ces degrés. On peut notamment remarquer que l'erreur obtenue, après traitement du CNLPP (5.48) par *fmincon*, est meilleure pour la solution ayant, pourtant, moins de points de contrôle.

5.5 Conclusions

Dans ce chapitre, la méthode d'optimisation HERO a été appliquée dans le cadre de la réduction de modèle à trois cas d'applications exhibant des spécificités courantes dans les cas industrielles. Ainsi, nous avons montré que cette technique de réduction de modèle est capable de s'adapter et de déterminer si des paramètres d'entrée ont une influence moindre par rapport aux autres. Dans le cas de l'approximation d'un champ de température, invariant par rapport à une variable d'entrée (y), HERO a généré une solution représentant cette propriété, en affectant très peu de points de contrôle dans cette direction et un degré minimal. La méthode ne remplace cependant pas une réelle étude des variables pertinentes du problème considéré. De par sa formulation, la méthode itérative employée pour la comparaison ne pouvait pas, quant à elle, refléter cette propriété.

Nous avons également montrer que la formulation NURBS permet aisément d'obtenir des métamodèles à plusieurs sorties, chaque sortie étant simplement une coordonnée des points de contrôle. Cela signifie, que le métamodèle est paramétré une seule fois, pour toutes les sorties. De plus, l'hypersurface obtenue est dérivable, ce qui permet, dans le cas d'un problème d'optimisation, d'utiliser le modèle réduit, ainsi que ses dérivées. Fournir les dérivées à une méthode déterministe d'optimisation permet de réduire le nombre d'évaluation de la fonction objectif, et donc le temps de calcul. Dans le cas d'une utilisation nécessitant des réponses rapides, comme la réalité virtuelle par exemple, le temps d'évaluation du modèle réduit est lui aussi un avantage et, sur les exemples de ce chapitre, le temps de calcul de l'ensemble des points de la plaque, pour des valeurs fixées des paramètres de conception, est inférieur à $1s$.

Une conclusion importante de ce chapitre est la capacité du métamodèle généré par HERO à intégrer des paramètres de conception telles que des conditions limites ou des variables géométriques comme variables d'entrée. Nous avons ainsi montré que l'hypersurface NURBS est capable de s'adapter à une géométrie variable, moyennant une étape de filtrage des points de contrôle "inactifs" de l'hypersurface. Ce filtrage est également possible dans le cas de variables présentant des discontinuités. Notons cependant que les bornes de la géométrie doivent être connues (i.e. les variables discontinues sont bornées).

Les résultats de notre approche ont été comparés à une méthode itérative utilisant les règles empiriques existantes dans le cadre des courbes et surfaces NURBS. Dans chaque cas, notre approche a montré de bien meilleurs résultats que la méthode itérative, nécessitant un nombre de points de contrôle bien plus faible pour une précision équivalente. Par ailleurs, l'utilisation de la fonction *fmincon* avec les résultats des solutions itératives a également montré que la précision des métamodèles obtenus peut être améliorée d'au moins un ordre de grandeur à chaque fois, accentuant le fait que ces solutions sont sous-optimales. De plus, ces solutions ne constituent pas un "bon" point de départ pour l'optimisation déterministe. En effet, dans le dernier cas présenté, la précision d'une solution de la méthode itérative, obtenue après l'optimisation par la fonction *fmincon*, était meilleure qu'une autre solution de la méthode itérative, optimisée par *fmincon* également, ayant pourtant plus de points de contrôle. Ces exemples ont permis de montrer l'intérêt qu'il y avait à ne pas utiliser les règles empiriques pour fixer les paramètres de l'hypersurface NURBS, et l'efficacité de l'algorithme HERO, développé pour trouver ces paramètres.

La limitation la plus importante de cette méthode est sa dépendance à l'organisation de la

base de données des points cibles. En effet, comme nous l'avons vu précédemment, l'efficacité de calcul des coordonnées des points de contrôle est grandement liée au fait que la base de données soit ordonnée ou non. Cela implique de coupler la génération de la base de données avec le choix de la méthode de réduction de modèle. Notons cependant qu'une première étape, visant à réorganiser les points cibles, peut être envisagée comme solution à ce problème. Turner [1], par exemple, traite ce problème avec la méthode du voisinage naturel (*natural neighborhood*) qui affecte la valeur du plus proche voisin du point de contrôle P_i pour la valeur de Q_i.

Chapitre 6

Application de la méthode de réduction de modèle au problème d'optimisation d'un panneau raidi d'aile d'avion

6.1 Introduction

Parmi les cas où l'utilisation d'un métamodèle offre un grand potentiel, nous pouvons citer les problèmes d'optimisation de structures dont la complexité est de niveau industriel. Ce chapitre traite, en particulier, du problème de conception d'une structure légère en matériaux composites. Ces matériaux anisotropes sont de plus en plus utilisés industriellement et dans de nombreux domaines grâce à leurs caractéristiques mécaniques particulières, à savoir, des rapports rigidité/-masse et résistance/masse permettant d'obtenir des structures plus légères qu'avec les alliages métalliques classiques. Cependant, les propriétés d'hétérogénéité et d'anisotropie de ces matériaux rendent la conception de ces structures difficile, d'autant plus que le comportement doit être caractérisé à différentes échelles : microscopique (i.e. échelle des composants), mésoscopique (i.e. échelle du pli) et macroscopique (i.e. échelle du stratifié).

Le problème considéré dans ce chapitre traite de l'optimisation d'une structure de type *panneau raidi* sujet à des contraintes de différentes natures. Ce type de panneau est utilisé dans de nombreuses applications car ils permettent, à rigidité équivalente, un gain de masse important. Même si de nombreux domaines, tel que l'aéronautique par exemple, porte un intérêt à la réduction de la masse structurelle, la complexité de conception de ce type de structure, couplée au manque de méthodes générales de conception, conduisent souvent à des solutions de mauvaises qualités. En effet, industriellement, ce genre de panneau est composé d'empilement "standard", ce qui permet d'assurer certaines propriétés attendues qui sont difficilement formalisables autrement (en raison des propriétés d'hétérogénéité et d'anisotropie des matériaux utilisés). Par exemple, pour assurer un découplage membrane/flexion, un empilement symétrique est utilisé. Or, cette condition est suffisante mais pas nécessaire à l'obtention d'une telle propriété. D'autre part, l'orientation angulaire des plis est souvent restreinte à un ensemble très limité de valeurs, composé des valeurs canoniques $0°$, $90°$ et $\pm 45°$ dans la majorité des cas. Il est très difficile d'obtenir la structure la plus légère possible dans ces conditions et les solutions industrielles sont

bien souvent sous-optimales, puisque limitées à un espace de conception très restreint.

Afin de s'affranchir des restrictions précédentes, une stratégie d'optimisation multi-échelle à deux niveaux (MS2L pour *Multi Scale Two Levels*) permettant la conception de structures complexes anisotropiques a été développée [229,230] et utilisée avec succès dans le cadre de l'optimisation des paramètres géométriques et mécaniques d'une structure du type panneau raidi [3]. Cette stratégie MS2L propose une formulation très générale du problème de conception sans introduire d'hypothèses simplificatrices et en considérant, comme variables de conception, à la fois les caractéristiques géométriques et les propriétés mécaniques qui définissent le comportement du panneau à chaque échelle caractéristique (mesoscopique et macroscopique). Pour ce faire, le problème d'optimisation est scindé en deux sous-problèmes distincts interdépendants. Le premier niveau intervient à l'échelle macroscopique et a pour objectif de trouver les valeurs optimales des variables de conceptions géométriques et mécaniques du panneau minimisant sa masse et respectant un ensemble de contraintes. Le second niveau intervient à l'échelle mesoscopique du pli et a pour but de trouver, au moins, une séquence d'empilement, pour chacun des stratifiés composant le panneau raidi, qui corresponde aux paramètres géométriques et mécaniques fournis par le problème du premier niveau. Cette approche utilise le formalisme polaire pour la représentation des caractéristiques mécaniques du panneau à l'échelle macroscopique [231] et l'algorithme génétique ERASMUS [14,212] présenté au chapitre 3.

Bien que cette approche ait montrée son efficacité [3], elle utilise, notamment pour le calcul des contraintes imposées sur le panneau, un code éléments finis qui ralentit l'exploration de l'algorithme dans le premier niveau d'optimisation. L'idée du travail réalisé dans ce chapitre est de générer le métamodèle permettant de s'affranchir du logiciel de calcul par éléments finis pour le calcul des contraintes dans le premier niveau de la stratégie MS2L d'optimisation. Après une brève introduction sur la théorie des stratifiés, le formalisme polaire et la stratégie MS2L de résolution du problème d'optimisation lié au panneau raidi, le métamodèle obtenu par la méthode HERO sera présenté et comparé au métamodèle obtenu par la méthode itérative décrite au chapitre 5. En fin de chapitre le métamodèle est utilisé pour résoudre le premier niveau de la stratégie MS2L et les résultats obtenus seront présentés.

6.2 Formulation du problème d'optimisation du panneau raidi

6.2.1 Description du problème d'optimisation

Le problème d'optimisation traité dans ce chapitre concerne l'unité répétitive (UR) d'un panneau raidi réalisé en composite stratifié typiquement utilisé pour la réalisation d'ailes d'avions. L'UR est composée par l'union d'une peau et d'un raidisseur de forme "omega" comme illustré par la fig. 6.1. Les dimensions extérieures de l'UR sont fixées à $a = 150$mm pour la largeur et $b = 600$mm pour la longueur. La longueur b représente la distance entre deux jonctions consécutives tandis que la largeur a représente le pas qui sépare les différents raidisseurs équidistants les uns des autres. Les raidisseurs, comme les peaux, sont réalisés à partir de plis unidirectionnels orthotropiques en carbone/epoxy dont les propriétés sont disponibles dans le tableau 6.1 (valeurs prises dans [3] et les références associées).

Les principales hypothèses utilisées pour la réponse mécanique à l'échelle macroscopique de

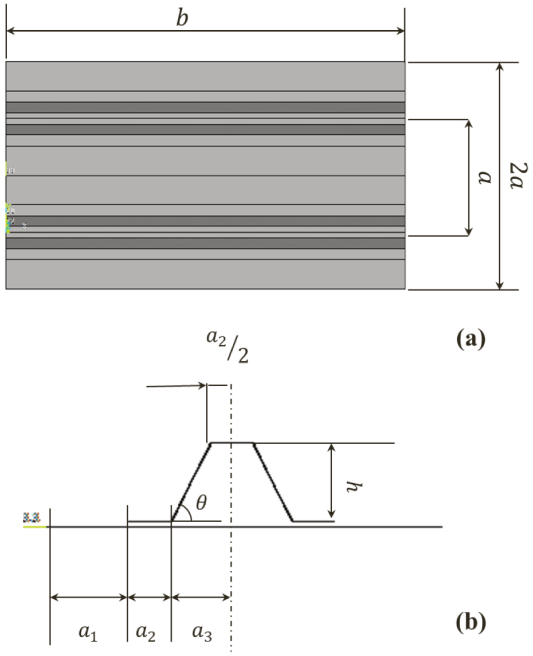


FIGURE 6.1 – (a) Géométrie et dimensions du panneau raidi (seulement deux unités répétitives sont représentées) et (b) paramètres géométriques de l'unité répétitive

l'UR concernent essentiellement le comportement et la géométrie du stratifié, de la peau comme du raidisseur :

- Les stratifiés sont composés de plis identiques (i.e. les plis ont la même épaisseur t_{ply} et sont constitués des mêmes matériaux).
- Le matériau qui compose une couche a un comportement élastique linéaire isotrope transverse.
- Chaque stratifié est quasi-homogène et complètement orthotropique [3, 232].
- A l'échelle macroscopique, la réponse élastique de chaque stratifié est décrite dans le cadre de la théorie FSDT (*First-order Shear Deformation Theory*) et les matrices de rigidité de la plaque sont exprimées en fonction des paramètres polaires du stratifié [233, 234].
- Il n'y a aucun délaminage aux interfaces entre plis, pour la peau comme pour la raidisseur.

L'intérêt de cette approche est l'abscence d'hypothèses simplificatrices sur les paramètres géométriques et mécaniques de l'UR et, en particulier, sur la nature de la séquence d'empilement des plis [3].

6.2.2 Stratégie d'optimisation multi-échelle à deux niveaux MS2L

La stratégie MS2L a pour objectif principal la conception de structures légères anisotropes. Dans le cas du panneau présenté à la fig. 6.1, le but est de minimiser la masse de celui-ci

Paramètres constants		Paramètres polaires de $\mathbf{Q}$ [a]		Paramètres polaires de $\hat{\mathbf{Q}}$ [b]	
E_1 [MPa]	161000.0	T_0 [MPa]	23793.3868	T [MPa]	5095.4545
E_2 [MPa]	9000.0	T_1 [MPa]	21917.8249	R [MPa]	1004.5454
G_{12} [MPa]	6100.0	R_0 [MPa]	17693.3868	Φ [deg]	90.0
ν_{12} [MPa]	0.26	R_1 [MPa]	19072.0711		
ν_{23} [MPa]	0.10	Φ_0 [deg]	0.0		
		Φ_1 [deg]	0.0		
Densité et épaisseur					
ρ [Kg/mm^3]	1.58×10^{-6}				
t_{pli} [mm]	0.125				

[a] Matrice réduite de rigidité du pli dans le plan.
[b] Matrice rigidité de cisaillement du pli hors plan.

Tableau 6.1 – Caractéristiques matériau des plis carbone/epoxy composant la peau et les raidisseurs [3]

en intégrant des contraintes de différentes natures, à savoir, mécaniques et géométriques mais aussi des contraintes de faisabilité et des contraintes technologiques. Pour ce faire, le processus d'optimisation est décomposé en deux phases distinctes liées entre-elles.

First-level problem. L'objectif de cette phase est de déterminer les valeurs optimales des paramètres géométriques et mécaniques des stratifiés composant l'UR du panneau minimisant sa masse et satisfaisant l'intégralité des contraintes imposées. À ce premier niveau, chaque stratifié est modélisé par une plaque anisotrope homogène équivalente dont le comportement est décrit par les paramètres polaires du stratifié [233,234]. Les variables de conception de cette étape sont donc les paramètres géométriques de l'UR ainsi que les paramètres polaires de la peau et du raidisseur. Le problème est traité à l'échelle macroscopique de la structure.

Second-level problem. Le second niveau de cette stratégie a pour but de déterminer une séquence d'empilement appropriée, pour les stratifiés de la peau et du raidisseur, correspondant aux paramètres géométriques et mécaniques fournis par le problème du premier niveau. Ce problème est traité à l'échelle mesoscopique (i.e. à l'échelle des plis constituant les stratifiés). Le but est de trouver, au moins, une séquence d'empilement pour chaque stratifié qui soit quasi-homogène, orthotrope en membrane et en flexion et dont les paramètres polaires correspondent à ceux issus de la première étape. Pour ce faire, les variables de conception de cette phase sont les orientations des différents plis constituants chaque stratifié.

6.2.3 Formulation mathématique du *first-level problem*

Variables de conception

Les variables de conception du panneau raidi sont de deux types : géométriques et mécaniques. Certains des paramètres géométriques sont illustrés sur la fig. 6.1. Tous ne sont bien sûr pas indépendants et les variables géométriques indépendantes sont les suivantes :

- Les épaisseurs des stratifiés de la peau et du raidisseur, t_S et t_B, respectivement ;
- La largeur a_2 de la *flange* inférieur du raidisseur ;
- La hauteur h du raidisseur ;
- La longueur a_3.

La longueur a_1 est liée aux variables précédentes par la relation suivante,

$$a_1 = \frac{a}{2} - a_2 - a_3, \tag{6.1}$$

tandis que l'angle d'inclinaison de la paroi du raidisseur est donné par

$$\theta = \arctan\left(\frac{h}{a_3 - \dfrac{a_2}{2}}\right). \tag{6.2}$$

Même si tous ces paramètres étaient inclus dans [3], notre étude n'utilise que les épaisseurs t_S et t_B, de la peau et du raidisseur respectivement, comme variables de conception géométriques. Ces variables sont soumises à des contraintes de nature technologique. En effet, les épaisseurs totales des stratifiés composant l'UR sont des variables discrètes et leur pas de discrétisation est égal à l'épaisseur d'un pli unitaire t_{ply} (tableau 6.1) :

$$t_\alpha = n_\alpha t_{ply}, \quad \alpha = S, B, \tag{6.3}$$

avec n_S et n_B les nombres de couches constituant les stratifiés de la peau et du raidisseur respectivement. Il est possible de noter que les valeurs optimales des épaisseurs des stratifiés, t_S et t_B, trouvées lors du problème de premier niveau fixent également les nombres de plis, n_S et n_B, utilisés dans le problème de second niveau. Les autres paramètres géométriques du panneau sont ceux de la solution de référence donnés au tableau 6.2. Le vecteur ξ_g^T regroupe les variables géométriques d'optimisation :

$$\boldsymbol{\xi}_g^T = \{n_S, n_B\}. \tag{6.4}$$

Selon la théorie FSDT [235], les lois de comportement qui régissent le stratifié (dans le repère global $R = \{0; x, y, z\}$) peuvent être formulée de la façon suivante :

$$\begin{Bmatrix} \mathbf{N} \\ \mathbf{M} \end{Bmatrix} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B} & \mathbf{D} \end{pmatrix} \begin{Bmatrix} \boldsymbol{\epsilon}_0 \\ \boldsymbol{\chi}_0 \end{Bmatrix}, \tag{6.5}$$

$$\mathbf{F} = \mathbf{H}\boldsymbol{\gamma}_0 \tag{6.6}$$

où $\mathbf{A}$, $\mathbf{B}$ et $\mathbf{D}$ sont les matrices de rigidité du stratifié en membrane, en couplage membrane/-flexion et en flexion respectivement, tandis que $\mathbf{H}$ est la matrice de rigidité en cisaillement hors plan. $\mathbf{N}$, $\mathbf{M}$ et $\mathbf{F}$ sont les vecteurs des efforts de membrane, de moments de flexion et des efforts de cisaillement par unité de longueur, respectivement, alors que $\boldsymbol{\epsilon}_0$, $\boldsymbol{\chi}_0$ et $\boldsymbol{\gamma}_0$ sont les vecteurs de contraintes dans le plan, de courbures et de contraintes de cisaillement hors plan, respectivement, du plan médian du stratifié (ces équations utilisent la notation de Voigt [235]).

a [mm]	150.00	
b [mm]	600.00	
a_2 [mm]	15.00	
a_3 [mm]	21.50	
h [mm]	30.00	
M_{ref} [Kg]	0.92	
λ_{ref} [N]	445074	
Séquence d'empilement	Pièce	Nombre de plis
$[(45/-45/90_2)_2\,/\,(45/-45)_3]_s$	peau (S)	28
$[45_2/0_2/-45_2/90_4/-45_2/0_2/45_2]_s$	raidisseur (B)	32

Tableau 6.2 – Solution de référence pour le problème de conception du panneau raidi

Afin de calculer la réponse élastique linéaire du stratifié, il est courant d'introduire les matrices de rigidité normalisées :

$$\mathbf{A}^* = \frac{1}{t}\mathbf{A}, \quad \mathbf{B}^* = \frac{2}{t^2}\mathbf{B}, \quad \mathbf{D}^* = \frac{12}{t^3}\mathbf{D}, \quad \mathbf{H}^* = \begin{cases} \frac{1}{t}\mathbf{H}, \\ \frac{12}{5t}\mathbf{H}. \end{cases} \tag{6.7}$$

avec t l'épaisseur totale du stratifié.

Dans le cadre du formalisme polaire, il est possible d'exprimer les composantes de ces matrices en fonction de leurs invariants d'élasticité [233,234]. En particulier, dans le cas de le théorie FSDT, il est possible de montrer que, pour un stratifié quasi-homogène et complètement orthotrope (i.e. un stratifié ayant le même comportement orthotrope pour les matrices de rigidité normalisées en membrane et en flexion et dont la matrice de rigidité de couplage membrane/flexion est nulle), le nombre de variables mécaniques de conception indépendantes permettant de décrire sa réponse mécanique s'élève seulement à trois. Ces trois variables sont les paramètres polaires anisotropes $R_{0K}^{A^*}$ et $R_1^{A^*}$ et l'angle polaire $\Phi_1^{A^*}$ (i.e. l'angle représentant l'orientation de l'axe principale d'orthotropie) de la matrice $\mathbf{A}^*$. Le lecteur intéressé par le formalisme polaire est renvoyé vers les références [233,234,236] pour approfondir le sujet.

Dans la formulation du problème d'optimisation de premier niveau de la stratégie MS2L, il est important de prendre en compte les contraintes de faisabilité sur les paramètres polaires (contraintes issues de la combinaison des orientations et des positions des plis dans l'empilement). Ces contraintes permettent notamment d'assurer qu'il est possible de trouver un stratifié (dans le second niveau de la stratégie MS2L) correspondant aux valeurs optimales des paramètres polaires trouvées dans le premier niveau [237]. Comme le stratifié est quasi-homogène, ces contraintes s'appliquent seulement à la matrice $\mathbf{A}^*$:

$$\begin{cases} -R_0 \leq R_{0K}^{A^*} \leq R_0, \\ 0 \leq R_1^{A^*} \leq R_1, \\ 2\left(\dfrac{R_1^{A^*}}{R_1}\right)^2 - 1 - \dfrac{R_{0K}^{A^*}}{R_0} \leq 0. \end{cases} \tag{6.8}$$

Dans l'éq. (6.8), R_0 et R_1 sont les modules d'anisotropie de la matrice de rigidité réduite du pli [3,233]. Il peut être utile de considérer les quantités sans dimensions suivantes :

$$\rho_0 = \frac{R_{0K}^{A^*}}{R_0}, \quad \rho_1 = \frac{R_1^{A^*}}{R_1}. \tag{6.9}$$

Dans ce cas, les contraintes de l'éq. (6.8) deviennent :

$$\begin{cases} -1 \leq \rho_0 \leq 1, \\ 0 \leq \rho_1 \leq 1, \\ 2\left(\rho_1\right)^2 - 1 - \rho_0 \leq 0. \end{cases} \tag{6.10}$$

Ces variables mécaniques doivent être considérées pour chacun des stratifiés constituant l'UR du panneau (i.e. le stratifié composant la peau et celui constituant le raidisseur). De plus, la direction principale d'orthototropie des stratifiés peut être fixée à zero (i.e. $\Phi_1^{A^*} = 0$ pour la peau comme pour le raidisseur), ce qui implique que l'axe principale d'orthotropie est confondu avec la direction de la charge appliquée. Le vecteur ξ_m^T regroupe les variables mécaniques de conception sans dimensions précédentes :

$$\boldsymbol{\xi}_m^T = \left\{\rho_{0S}, \rho_{1S}, \rho_{0B}, \rho_{1B}\right\}. \tag{6.11}$$

Problème d'optimisation et stratégie de résolution

L'objectif du problème d'optimisation de premier niveau (à l'échelle macroscopique) est de minimiser la masse de l'UR du panneau raidi en satisfaisant un ensemble de contraintes de différentes natures. Les variables de conception sont regroupées dans le vecteur ξ^T de la manière suivante :

$$\boldsymbol{\xi}^T = \left\{\boldsymbol{\xi}_g^T, \boldsymbol{\xi}_m^T\right\}. \tag{6.12}$$

Le problème de minimisation qui en résulte est un CNLPP de la forme :

$$\begin{gathered} \min_{\boldsymbol{\xi}} \frac{M\left(\boldsymbol{\xi}\right)}{M_{ref}}, \\ \text{sujet à :} \\ \begin{cases} 1.05 - \dfrac{\lambda\left(\boldsymbol{\xi}\right)}{\lambda_{ref}} \leq 0, \\ 2\left(\rho_{1S}\right)^2 - 1 - \rho_{0S} \leq 0, \\ 2\left(\rho_{1B}\right)^2 - 1 - \rho_{0B} \leq 0, \end{cases} \end{gathered} \tag{6.13}$$

Le domaine de définition ainsi que le type de chacune des variables de conception du problème de premier niveau sont disponibles dans le tableau 6.3. Dans le problème (6.13), M est la masse totale de l'UR et λ est la charge critique du premier mode de flambage du panneau raidi tandis que M_{ref} et λ_{ref} sont les mêmes quantités pour une *solution de référence* soumise aux mêmes conditions limites que celles appliquées à l'UR du panneau et dont la configuration est donnée au tableau 6.2.

Variable de conception	Type	Limite inférieure	Limite supérieure	Pas
ρ_{0S}	continu	-1.0	1.0	-
ρ_{1S}	continu	0.0	1.0	-
ρ_{0B}	continu	-1.0	1.0	-
ρ_{1B}	continu	0.0	1.0	-
n_S	entier	20	32	1
n_B	entier	20	32	1

Tableau 6.3 – Domaine de définition des variables de conception du problème de premier niveau de la stratégie MS2L

Le CNLPP (6.13) est un problème d'optimisation non-convexe à la fois des variables géométriques et mécaniques de conception. La non-linéarité et la non-convexité résultent de la nature de la contrainte sur la charge critique de flambage qui est une fonction non-convexe. La complexité du problème est également augmentée par les contraintes non-linéaires de faisabilité sur les paramètres polaires des stratifiés. A noter également que les variables de conception ne sont pas toutes du même type et, en particulier, les variables n_S et n_B sont de type entier, ce qui rend les méthodes déterministes classiques de résolution inutilisables (cf. chapitre 3).

Ce problème a été traité par Montemurro [3] au moyen de la métaheuristique ERASMUS (présentée au chapitre 3) couplée à un modèle de calcul par éléments finis (pour l'évaluation de la charge critique du premier mode de flambage de la structure). Bien que cette approche ait prouvée son efficacité (gain de masse de 11.5% et augmentation de la charge critique d'environ 9% par rapport à la solution de référence), le temps nécessaire à la recherche de la solution est directement lié au temps de calcul de la charge critique du premier mode de flambage du modèle par éléments finis couplé à l'AG ERASMUS. Dans ce genre de situation, l'emploi d'un métamodèle peut permettre de gagner un temps considérable dans le processus d'optimisation. C'est pourquoi l'objet de la section 6.3 de ce chapitre est l'obtention d'un métamodèle permettant d'évaluer la charge critique du premier mode de flambage. Le métamodèle obtenu sera utilisé pour la résolution du problème de premier niveau (6.13) dans la section 6.4 et les résultats obtenus seront comparés à ceux de [3].

6.2.4 Formulation mathématique du second niveau de la stratégie MS2L

La stratégie de réduction de modèle employée n'intervient pas dans le second niveau de la stratégie MS2L, toutefois, une brève description du problème considéré est fournie dans cette section à titre d'information pour le lecteur. Cette étape vise à fournir les séquences d'empilement des stratifiés composant la peau et le raidisseur. L'objectif est de déterminer au moins un empilement qui satisfasse, d'une part, les valeurs optimales des paramètres géométriques et polaires trouvés par la résolution du problème de premier niveau et, d'autre part, les symétries élastiques imposées aux stratifiés dans la formulation du problème de premier niveau, i.e. stratifiés quasi-homogènes et complètement orthotropes. Dans le cadre de la théorie FSDT et avec le

formalisme polaire pour représenter les matrices de rigidité des stratifiés, ce problème peut être formulé comme un problème de minimisation sans contraintes de la forme suivante :

$$\min_{\boldsymbol{\delta}} I\left(f_i\left(\boldsymbol{\delta}\right)\right), \tag{6.14}$$

avec

$$I\left(f_i\left(\boldsymbol{\delta}\right)\right) = \sum_{i=1}^{6} f_i\left(\boldsymbol{\delta}\right), \tag{6.15}$$

où $\boldsymbol{\delta} \in \mathbb{R}^n$ est le vecteur contenant les orientations des différentes couches composant le stratifié, i.e. les variables de conception de ce problème de deuxième niveau, tandis que $f_i\left(\boldsymbol{\delta}\right)$ sont des fonctions quadratiques dans l'espace des paramètres polaires, chacune représentant une contrainte à satisfaire comme l'orthotropie, le découplage membrane/flexion, etc. Pour le problème considéré, les différentes fonctions sont les suivantes :

$$f_1\left(\boldsymbol{\delta}\right) = \left(\frac{\left|\Phi_0^{A^*}\left(\boldsymbol{\delta}\right) - \Phi_1^{A^*}\left(\boldsymbol{\delta}\right)\right|}{\frac{\pi}{4}} - K^{A^*(opt)} \right)^2, \tag{6.16}$$

$$f_2\left(\boldsymbol{\delta}\right) = \left(\frac{R_0^{A^*}\left(\boldsymbol{\delta}\right) - R_0^{A^*(opt)}}{R_0} \right)^2, \tag{6.17}$$

$$f_3\left(\boldsymbol{\delta}\right) = \left(\frac{R_1^{A^*}\left(\boldsymbol{\delta}\right) - R_1^{A^*(opt)}}{R_1} \right)^2, \tag{6.18}$$

$$f_4\left(\boldsymbol{\delta}\right) = \left(\frac{\left|\Phi_1^{A^*}\left(\boldsymbol{\delta}\right) - \Phi_1^{A^*(opt)}\right|}{\frac{\pi}{4}} \right)^2, \tag{6.19}$$

$$f_5\left(\boldsymbol{\delta}\right) = \left(\frac{\left\|\mathbf{C}\left(\boldsymbol{\delta}\right)\right\|}{\mathbf{Q}} \right)^2, \tag{6.20}$$

$$f_6\left(\boldsymbol{\delta}\right) = \left(\frac{\left\|\mathbf{B}^*\left(\boldsymbol{\delta}\right)\right\|}{\mathbf{Q}} \right)^2, \tag{6.21}$$

où $f_1\left(\boldsymbol{\delta}\right)$ représente la contrainte d'orthotropie du stratifié (imposée par la valeur de K^{A^*} qui est liée au signe de ρ_0 obtenu à la fin de la première étape de la stratégie MS2L), $f_2\left(\boldsymbol{\delta}\right)$, $f_3\left(\boldsymbol{\delta}\right)$ et $f_4\left(\boldsymbol{\delta}\right)$ sont les fonctions liées aux valeurs optimales des paramètres polaires fournis par la résolution du premier niveau de la stratégie tandis que $f_5\left(\boldsymbol{\delta}\right)$ et $f_6\left(\boldsymbol{\delta}\right)$ sont reliées aux conditions de quasi-homogénéité.

La fonction $I\left(f_i\left(\boldsymbol{\delta}\right)\right)$ est une fonction semi-définie positive et convexe dans l'espace des paramètres polaires du stratifié car elle est définie comme une somme de fonctions convexes. Cependant, cette fonction est hautement non-convexe dans l'espace des orientations des plis car les paramètres polaires du stratifié dépendent de fonctions trigonométriques des angles d'orientation des différentes couches. De plus, le minimum global de $I\left(f_i\left(\boldsymbol{\delta}\right)\right)$ est connu *a priori* puisqu'il correspond aux valeurs nulles des fonctions qui la composent [233,234]. Il est important de noter

que le problème (6.14) doit être résolu deux fois : une fois pour le stratifié constituant la peau et une fois pour celui constituant le raidisseur.

Dans le but de simplifier la résolution du problème (6.14) l'espace de recherche de l'empilement optimal est restreint à une classe particulière de stratifiés quasi-homogènes : les séquences d'empilement QT (*Quasi-Trivial*) qui représentent des solutions exactes par rapport aux contraintes de quasi-homogénéité (i.e. les fonctions $f_5(\boldsymbol{\delta})$ et $f_6(\boldsymbol{\delta})$ sont nulles pour ces empilements). Bien que très intéressante, la résolution de ce problème n'est pas l'objet de ce chapitre et lecteur souhaitant approfondir le sujet est renvoyé vers les références [3,238,239].

6.3 Utilisation de la stratégie HERO pour obtenir un métamodèle de la charge critique du premier mode de flambage du panneau raidi $\mathbb{R}^6 \rightarrow \mathbb{R}$

6.3.1 Génération de la base de données

Comme évoqué précédemment, le temps nécessaire à la résolution du problème d'optimisation de premier niveau de la stratégie MS2L est directement lié au temps de calcul de la charge critique du premier mode de flambage de la structure. Afin d'améliorer le temps nécessaire à l'optimisation des paramètres géométriques et mécaniques du panneau raidi, nous utiliserons un métamodèle généré à l'aide de HERO pour obtenir la charge critique du premier mode de flambage de la structure. Pour cela, les points cibles permettant de paramétrer le métamodèle sont obtenus à l'aide du logiciel commercial de calcul par éléments finis ANSYS. La charge critique de flambage de la structure est obtenue au moyen d'une analyse linéaire de flambage (i.e. la résolution d'un problème aux valeurs propres).

L'objectif du métamodèle est donc de fournir la charge critique de flambage λ en fonction des variables de conception géométriques et mécaniques du panneau raidi :

$$\begin{aligned} \lambda &= \lambda(\boldsymbol{\xi}) \\ &= \lambda(n_S, n_B, \rho_{0S}, \rho_{1S}, \rho_{0B}, \rho_{1B}). \end{aligned} \tag{6.22}$$

Les contraintes de faisabilité des stratifiés de la peau et du raidisseur étant connues (i.e. les contraintes permettant d'assurer qu'une solution, au moins, existe pour le problème de second niveau), cf. éq. (6.10), les points cibles ne seront pas pris dans les zones de l'espace ne respectant ces contraintes. De plus, ces contraintes étant dues à des fonctions trigonométriques, la morphologie de l'espace des points cibles varie avec les paramètres de conception mécaniques $\boldsymbol{\xi}_m$. Cela impose, comme dans le troisième exemple du chapitre précédent (cf. section 5.4), de filtrer les points de contrôle afin de prendre en compte uniquement ceux dont le support local se situe dans les zones respectant les contraintes (6.10).

La base de données des points cibles est composée des valeurs de la charge critique de flambage λ pour différentes valeurs des variables n_S, n_B, ρ_{0S}, ρ_{1S}, ρ_{0B} et ρ_{1B}, et est stockée sous la forme d'une matrice à six dimensions $\mathbf{Q}$, avec

$$Q_{s_1,s_2,s_3,s_4,s_5,s_6} = \lambda(n_S^{s_1}, n_B^{s_2}, \rho_{0S}^{s_3}, \rho_{1S}^{s_4}, \rho_{0B}^{s_5}, \rho_{1B}^{s_6}), \quad s_k = 0, ..., r_k. \tag{6.23}$$

Les différents paramètres $u_{s_k}^{(k)}$, $k = 1, ..., 6$, sont donnés par les relations suivantes :

$$u_{s_1}^{(1)} = \frac{n_S^{s_1} - 20}{32 - 20}, \quad n_S^{s_1} \in [20, 32]\,, \tag{6.24}$$

$$u_{s_2}^{(2)} = \frac{n_B^{s_2} - 20}{32 - 20}, \quad n_B^{s_2} \in [20, 32]\,, \tag{6.25}$$

$$u_{s_3}^{(3)} = \frac{\rho_{0S}^{s_3} - (-1)}{1 - (-1)}, \quad \rho_{0S}^{s_3} \in [-1, 1]\,, \tag{6.26}$$

$$u_{s_4}^{(4)} = \rho_{1S}^{s_4}, \quad \rho_{1S}^{s_4} \in [0, 1]\,, \tag{6.27}$$

$$u_{s_5}^{(5)} = \frac{\rho_{0B}^{s_5} - (-1)}{1 - (-1)}, \quad \rho_{0B}^{s_5} \in [-1, 1]\,, \tag{6.28}$$

$$u_{s_6}^{(6)} = \rho_{1B}^{s_6}, \quad \rho_{1B}^{s_6} \in [0, 1]\,. \tag{6.29}$$

L'hypersurface NURBS permettant la représentation de la charge critique de flambage est une application $\mathbf{H}(\mathbf{u}) : \mathbb{R}^6 \to \mathbb{R}$. Les différents paramètres sans dimension $u^{(k)}$ sont donnés par les relations suivantes :

$$u^{(1)} = \frac{n_S - 20}{32 - 20}, \quad n_S \in [20, 32]\,, \tag{6.30}$$

$$u^{(2)} = \frac{n_B - 20}{32 - 20}, \quad n_B \in [20, 32]\,, \tag{6.31}$$

$$u^{(3)} = \frac{\rho_{0S} - (-1)}{1 - (-1)}, \quad \rho_{0S} \in [-1, 1]\,, \tag{6.32}$$

$$u^{(4)} = \rho_{1S}, \quad \rho_{1S} \in [0, 1]\,, \tag{6.33}$$

$$u^{(5)} = \frac{\rho_{0B} - (-1)}{1 - (-1)}, \quad \rho_{0B} \in [-1, 1]\,, \tag{6.34}$$

$$u^{(6)} = \rho_{1B}, \quad \rho_{1B} \in [0, 1]\,. \tag{6.35}$$

Il est important de noter que, même si le métamodèle est capable d'utiliser des valeurs de n_α ($\alpha = S, B$) non entières, ces valeurs n'aurait aucun sens physique et seules des valeurs de $u^{(1)}$ et $u^{(2)}$ correspondant à des valeurs entières de n_S et n_B doivent être utilisées.

La base de données des points cibles est obtenue grâce à une modélisation dans le logiciel de calcul par éléments finis ANSYS qui permet d'obtenir la charge critique de flambage à partir l'UR du panneau raidi, et non pas du panneau complet. La fig. 6.2 présente le modèle éléments finis de l'UR utilisé. Ce modèle est construit avec des éléments de type coque à huit nœuds (SHELL281) et des éléments *non-linear multi-point constraints* (MPC184), tous deux constitués de six degrés de liberté (DOFs) par nœud. Le comportement mécanique des éléments SHELL281 est directement décrit par les matrices de rigidité homogènes $\mathbf{A}^*$, $\mathbf{B}^*$, $\mathbf{D}^*$ et $\mathbf{H}^*$. Les éléments MPC184 permettent d'assurer la compatibilité des champs de déplacement entre la peau et le raidisseur. La formulation de ces éléments est basée sur un schéma classique d'éléments *multi-point constraint* [240]. Des éléments MPC184 sont définis entre chaque couple de nœuds en vis à vis sur la peau et le raidisseur (cf. fig. 6.2). Plus précisément, des éléments MPC184 sont définis

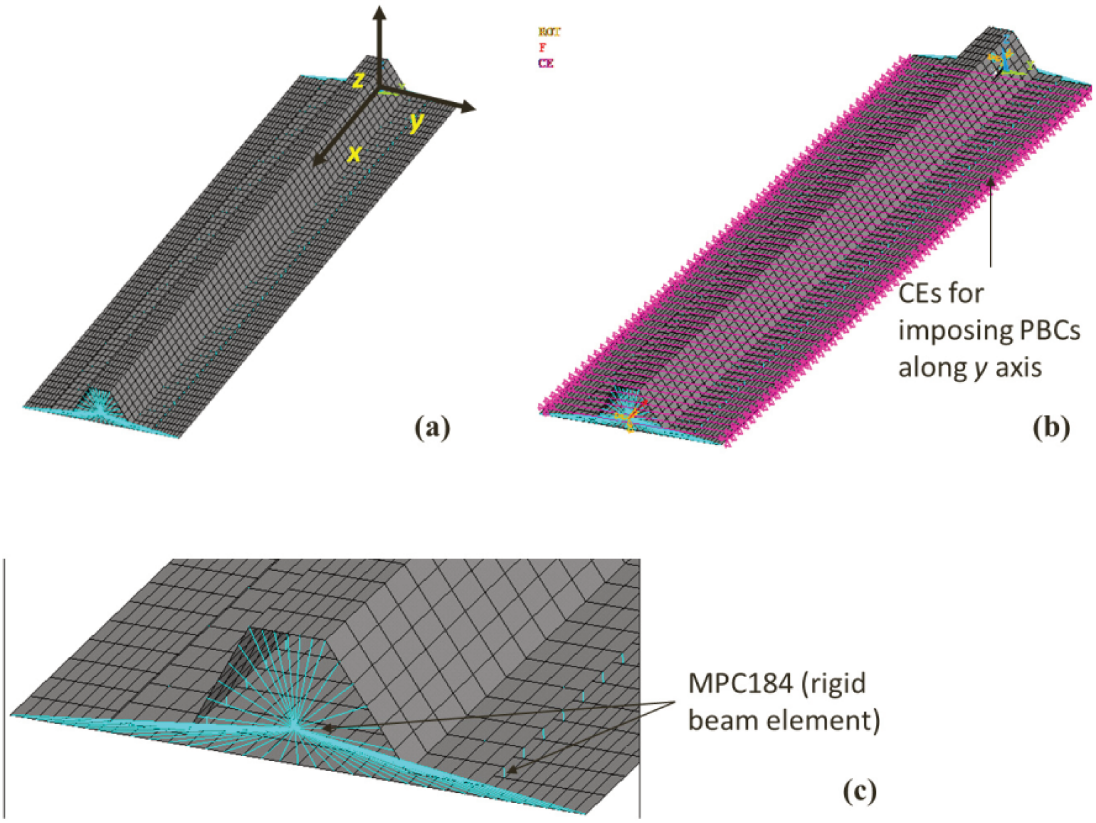


FIGURE 6.2 – (a) Modèle éléments finis de l'unité répétitive du panneau raidi, (b) équations de contraintes pour les conditions limites périodiques imposées selon l'axe y et (c) zoom sur les éléments MPC184

entre chacun des nœuds du plan moyen de la peau (nœuds "maîtres") et ceux du plan moyen du raidisseur (nœuds "esclaves") dans la partie correspondante aux brides inférieures du raidisseur. Pour ce faire, il est évident que ces surfaces doivent être maillées de la même manière.

En outre, des éléments MPC184 ont également été utilisés pour rigidifier les sections transverses finales de l'UR, dans le but de simuler la présence des nervures (ces dernières ont une rigidité dans le plan supérieure d'un ou deux ordres de grandeur par rapport à la rigidité en flexion de l'UR). Plus particulièrement, deux nœuds pilotes (i.e. sur lesquels sont appliquées les conditions limites) $A = \{0, 0, z_g\}$ et $B = \{b, 0, z_g\}$ sont utilisés comme représenté sur la fig. 6.2. La composante z_g est la coordonnée selon l'axe z du barycentre de la section transverse de l'UR. Les nœuds A et B sont liés, par des éléments MPC184, aux nœuds composant la section transverse correspondante, i.e. les nœuds du plan $x = 0$ et les nœuds du plan $x = b$ respectivement. Les conditions limites appliquées sur ces nœuds sont :

$$\begin{aligned} &\text{nœud } A : u_i = 0, \quad \beta_i = 0, \quad i = x, y, z, \\ &\text{nœud } B : F_x = -1\text{N}, \quad u_y = u_z = 0, \quad \beta_i = 0, \quad i = x, y, z, \end{aligned} \tag{6.36}$$

avec u_i et β_i les déplacements et rotations nodales respectivement, tandis que F_x est la composante selon l'axe x de l'effort nodal.

Il est important de noter que le calcul de la charge critique de flambage du panneau raidi est effectué grâce à son UR et, de ce fait, des conditions limites pertinentes doivent être appliquées. En particulier, cela impose implicitement l'hypothèse que le panneau est de longueur "infinie" selon l'axe y. Cette hypothèse se traduit par des conditions limites périodiques (PBCs) de la forme suivante :

$$\begin{aligned} u_i\left(x,-\frac{a}{2},0\right)-u_i\left(x,\frac{a}{2},0\right)=0, \quad \forall x\in]0,b[\,, \quad i=x,y,z,\\ \beta_i\left(x,-\frac{a}{2},0\right)-\beta_i\left(x,\frac{a}{2},0\right)=0, \quad \forall x\in]0,b[\,, \quad i=x,y,z. \end{aligned} \tag{6.37}$$

Les PBCs de l'éq. (6.37) doivent être définies pour chaque couple de nœuds appartenant aux bords latéraux de la peau (i.e. les lignes situées à $y=\pm\frac{a}{2}$) à l'exception de ceux situés à $x=0$ et $x=b$ qui sont déjà reliés aux nœuds pilotes A et B respectivement. Les PBCs sont définies à l'aide des équations de contraintes (CEs) d'ANSYS [240] entre les nœuds homologues des bords latéraux de la peau.

6.3.2 Métamodèle obtenu par HERO

Afin de paramétrer le métamodèle, la stratégie HERO (décrite au chapitre 4) est employée. Dans un premier temps, le CNLPP traité par ERASMUS est de la forme suivante :

$$\min_{\mathbf{x}_\mathrm{I}} \phi(\mathbf{x}_\mathrm{I}) = a\times\frac{1}{6}\sum_{k=1}^{6}\frac{n_k}{r_k}+(1-a)\times\frac{1}{6}\sum_{k=1}^{6}\frac{p_k}{p_k^{max}}, \quad a=0.5,$$

sujet à :

$$\begin{cases} \epsilon_\lambda^{(max)}(\mathbf{x}_\mathrm{I})\le\epsilon_{max}^{(max)},\\ \epsilon_\lambda^{(moy)}(\mathbf{x}_\mathrm{I})\le\epsilon_{max}^{(moy)},\\ 0<U_{l_k}^{(k)}<1, \quad l_k=p_k+1,...,m_k-p_k+1, \quad k=1,...,6,\\ U_{l_k}^{(k)}\le U_{l_k+1}^{(k)}, \quad l_k=p_k+1,...,m_k-p_k, \quad k=1,...,6,\\ n_k-p_k\ge 0, \quad k=1,...,6,\\ n_{CP}\le n_{TP}, \end{cases} \tag{6.38}$$

où $\mathbf{x}_\mathrm{I}$ est le vecteur contenant les variables d'optimisation, i.e. les 6 degrés p_k, les 6 tailles des *knot vectors* m_k+1 et les m_k-2p_k-1 valeurs internes des *knot vectors*. L'erreur maximale d'approximation de la charge critique de flambage $\epsilon_\lambda^{(max)}$ est donnée par :

$$\epsilon_\lambda^{(max)}=\max_{\mathbf{u}_{s_1,...,s_6}}\left(\frac{|H(\mathbf{u}_{s_1,...,s_6})-\lambda\left(n_S^{s_1},n_B^{s_2},\rho_{0S}^{s_3},\rho_{1S}^{s_4},\rho_{0B}^{s_5},\rho_{1B}^{s_6}\right)|}{\lambda_{\max}-\lambda_{\min}}\right), \tag{6.39}$$

où $\mathbf{u}_{s_1,...,s_6}=\left(u_{s_1}^{(1)},u_{s_2}^{(2)},u_{s_3}^{(3)},u_{s_4}^{(4)},u_{s_5}^{(5)},u_{s_6}^{(6)}\right)$ représente les coordonnées sans dimensions du point cible $\lambda\left(n_S^{s_1},n_B^{s_2},\rho_{0S}^{s_3},\rho_{1S}^{s_4},\rho_{0B}^{s_5},\rho_{1B}^{s_6}\right)$, $s_k=0,...,r_k$, $k=1,...,6$, $\lambda_{\max}$ est la charge critique de flambage maximale sur l'ensemble des points cibles, tandis que λ_{min} est la charge critique minimale

sur ce même ensemble. L'erreur moyenne d'approximation, quant à elle, est donnée par :

$$\epsilon_{\lambda}^{(moy)} = \frac{1}{n_{TP}} \sum_{s_1=0}^{r_1} \cdots \sum_{s_6=0}^{r_6} \frac{|H(\mathbf{u}_{s_1,\ldots,s_6}) - \lambda\left(n_S^{s_1}, n_B^{s_2}, \rho_{0S}^{s_3}, \rho_{1S}^{s_4}, \rho_{0B}^{s_5}, \rho_{1B}^{s_6}\right)|}{\lambda_{\max} - \lambda_{\min}}, \tag{6.40}$$

avec n_{TP} le nombre de points cibles. Les erreurs maximale et moyenne étant formulées par l'éq. (6.39) et l'éq. (6.40) respectivement, les seuils d'erreurs maximale et moyenne ont été fixés à $\epsilon_{max}^{(max)} = 5e^{-2}$ et $\epsilon_{max}^{(moy)} = 5e^{-3}$ respectivement. La structure d'un individu d'ERASMUS pour traiter ce problème, dans le cadre de l'approximation, est donnée à la fig. 6.3. Elle est composée d'une section standard, contenant les degrés p_1, p_2, p_3, p_4, p_5, p_6 et les nombres de chromosomes des différentes sections modulaires, ainsi que de six sections modulaires indépendantes, contenant les valeurs internes des *knot vectors* $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$, $\mathbf{U}^{(3)}$, $\mathbf{U}^{(4)}$, $\mathbf{U}^{(5)}$ et $\mathbf{U}^{(6)}$.

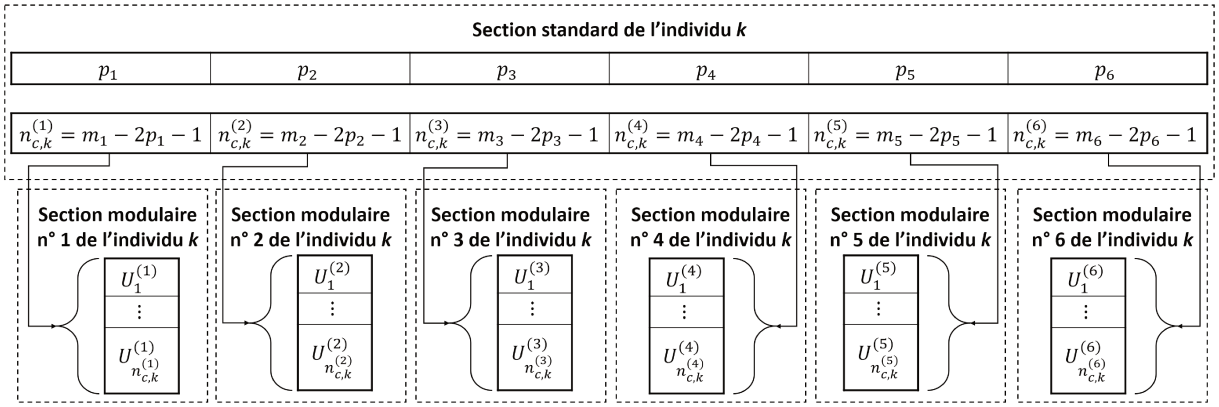


FIGURE 6.3 – Structure d'un individu d'ERASMUS pour le CNLPP (6.38)

Les différents paramètres génétiques d'ERASMUS sont définis comme suit pour ce problème :

1. Le nombre de populations a été fixé à $n_{pop} = 3$
2. Le nombre d'individus a été fixé à $n_{ind} = 200$
3. Le critère d'arrêt est le nombre d'itérations, fixé à $n_{gen} = 150$ pour ce cas.
4. La probabilité de croisement est fixée à la valeur par défaut $p_{cross} = 0.85$.
5. La probabilité de mutation des gènes est fixée à une valeur de $p_{mut} = 0.005$.
6. La probabilité de décalage est fixée à $p_{shift} = 0.5$.
7. Le type de sélection retenu est la méthode de la *roulette*.
8. Le paramètre de *fitness pressure* est fixé à $f_{pres} = 1$, afin de ne pas brider l'exploration du domaine.
9. L'opérateur d'*élitisme* est actif.
10. Le nombre de générations pendant lesquelles les populations évoluent de manière isolée les unes des autres est fixé à $I_{time} = 5$.

11. Le nombre de contraintes est, dans le cas du CNLPP (6.38), $n_{constr} = 2$.
12. Le nombre de chromosomes est variable.
13. Le nombre de composants modulaires est égal au nombre de *knot vectors*, soit $n_{modular} = 6$.
14. Les nombres minimums et maximums de chromosomes sont fixés à $n^{mod_k}_{chrom,min} = 0$ et $n^{mod_k}_{chrom,max} = r_k$ pour chaque type de module (i.e. $k = 1, ..., 6$).
15. Dans le cas de l'approximation, le nombre de chromosomes de la partie standard des individus est toujours $n^{std}_{chrom} = 1$.
16. Le nombre de gènes de la partie standard est égal au nombre de degrés à fixer, soit $n^{std}_{gene} = 6$.
17. Les nombres de gènes des chromosomes des différents composants modulaires sont, quelle que soit l'utilisation (interpolation ou approximation), $n^{mod_k}_{gene} = 1$, $k = 1, ..., 6$.
18. Le nombre de variables différentes est $n_{var} = 2$, dans le cas de l'approximation. Il s'agit des degrés p_k et des valeurs internes des *knot vectors* :
    - *Degré* :
        - *Nom de la variable* : degré
        - *Nature de la variable* : régulièrement discrétisée (*regular_ discr*).
        - *La limite inférieure de la variable* : $p^{min}_k = 1$.
        - *La limite supérieure de la variable* : $p^{max}_k = 7$.
        - *Le pas de discrétisation* : 1
    - *Valeur de nœud* :
        - *Nom de la variable* : valeur de nœud
        - *Nature de la variable* : continue (*extended*), dans ce cas, une discrétisation est réalisée automatiquement par ERASMUS en fonction de la précision machine (i.e. pas de pas de discrétisation à fixer).
        - *La limite inférieure de la variable* : $0 + \epsilon$ avec $\epsilon = 0.0001$.
        - *La limite supérieure de la variable* : $1 - \epsilon$.

Après qu'ERASMUS ait atteint son critère d'arrêt, la solution qu'il renvoie sert de point de départ pour le CNLPP suivant :

$$\min_{\mathbf{x}_{\mathrm{II}}} \psi^{(30)}(\mathbf{x}_{\mathrm{II}}) = \frac{\left[\sum_{s_1=0}^{r_1} \cdots \sum_{s_6=0}^{r_6} \left(H_{\mathbf{x}_{\mathrm{II}}}(\mathbf{u}_{s_1,\ldots,s_6}) - \lambda(n_S^{s_1}, n_B^{s_2}, \rho_{0S}^{s_3}, \rho_{1S}^{s_4}, \rho_{0B}^{s_5}, \rho_{1B}^{s_6})\right)^{30}\right]^{(\frac{1}{30})}}{\left[\sum_{s_1=0}^{r_1} \cdots \sum_{s_6=0}^{r_6} \left(H_0(\mathbf{u}_{s_1,\ldots,s_6}) - \lambda(n_S^{s_1}, n_B^{s_2}, \rho_{0S}^{s_3}, \rho_{1S}^{s_4}, \rho_{0B}^{s_5}, \rho_{1B}^{s_6})\right)^{30}\right]^{(\frac{1}{30})}},$$

sujet à :

$$\begin{cases} 0 < U^{(k)}_{l_k} < 1, \quad l_k = p_k + 1, ..., m_k - p_k + 1, \quad k = 1, ..., 6, \\ U^{(k)}_{l_k} \leq U^{(k)}_{l_k+1}, \quad l_k = p_k + 1, ..., m_k - p_k, \quad k = 1, ..., 6, \\ \omega_{i_1,\ldots,i_6} \geq 0, \quad i_k = 0, ..., n_k, \end{cases} \tag{6.41}$$

où $H_{\mathbf{x}_{\text{II}}}$ est l'hypersurface NURBS obtenue avec le vecteur des variables d'optimisation $\mathbf{x}_{\text{II}}$ tandis que H_0 est l'hypersurface NURBS obtenue avec le vecteur des variables d'optimisation $\mathbf{x}_{\text{II}}^{(0)}$ fourni par ERASMUS. Le vecteur $\mathbf{x}_{\text{II}}$ regroupe les variables d'optimisation de la manière suivante :

$$\mathbf{x}_{\text{II}} = \left(U^{(1)}_{p_1+1}, \cdots, U^{(1)}_{m_1-p_1+1}, \cdots, U^{(6)}_{p_6+1}, \cdots, U^{(6)}_{m_6-p_6+1}, \boldsymbol{\Omega}\right). \tag{6.42}$$

Contrairement aux applications présentées au chapitre 5, certaines solutions renvoyées par ERASMUS ont nécessité l'introduction des poids parmi les variables d'optimisation du vecteur $\mathbf{x}_{\text{II}}$ (i.e. $\boldsymbol{\Omega} \neq \emptyset$). Cependant, afin de diminuer les temps de calcul, le CNLPP (6.41) est directement traité par une méthode déterministe. Les paramètres de *fmincon* pour résoudre le CNLPP (6.41) sont les mêmes que pour les exemples précédents.

Les résultats présentés ci-après ont été obtenus à partir de plusieurs bases de données de points différentes dont les propriétés sont fournies dans le tableau 6.4. Afin d'utiliser la stratégie d'obtention des coordonnées des points de contrôle "dimension par dimension", introduite dans la section 4.3, il est nécessaire de définir un ensemble de points cibles ordonné. Or, le choix de la fonction liant les coordonnées paramétriques d'un point de l'hypersurface NURBS aux variables de conception, associé aux contraintes de faisabilité des stratifiés (cf. CNLPP (6.13), impliquent que certains points cibles sont en dehors du domaine faisable. La charge critique de flambage n'a bien sûr pas été calculée en ces points mais il est important de noter que parmi les points de chacune des base de données, seuls les points dans le domaine faisable sont considérés. C'est pourquoi, le nombre de points cibles dans le domaine faisable est également indiqué dans le tableau 6.4. Nous rappelons également que les points de contrôle dont le support local n'est que dans le domaine non-faisable des stratifiés sont, de toute façon, filtrés en amont du calcul des coordonnées.

Base de données	Nombre de points (r_k+1)						Nombre total de points	Nombre de points dans le domaine faisable
	n_S	n_B	ρ_{0S}	ρ_{1S}	ρ_{0B}	ρ_{1B}		
BDD 1	7	7	7	7	7	7	117 649	47 089
BDD 2	9	9	9	9	9	9	531 441	210 681
BDD 3	2	2	5	5	5	5	11 664	5 184
BDD 4	3	3	10	10	10	10	160 000	65 536

Tableau 6.4 – Propriétés des bases de données utilisées pour paramétrer et valider les métamodèles

Le tableau 6.5 présente les résultats obtenus en utilisant la base de données 1 pour paramétrer le métamodèle. Les résultats sont également comparés à ceux obtenus par la méthode itérative présentée en début de chapitre 5. Le métamodèle est paramétré dans une optique d'approximation et pas d'interpolation (i.e. $n_{CP} < n_{TP}$). Dans ce contexte, il est très intéressant de remarquer que le nombre de points de contrôle, ainsi que les degrés, sont les mêmes pour les deux approches. Concernant les degrés, cela ne paraît absolument pas étonnant car, le nombre de points de contrôle dans chaque direction étant relativement faible, augmenter le degré se résume à diminuer le nombre de variables d'optimisation, et donc les possibilités de modifier l'hypersurface. En effet, lorsque $n_k - p_k = 0$, il n'y a plus de valeurs internes dans le *knot-vector* $\mathbf{U}^{(k)}$. Dans ce cas

particulier (nombre de points cibles peu élevé dans chaque direction), la nécessité d'avoir des degrés p_k faibles est d'autant plus important.

Variable		n_S	n_B	ρ_{0S}	ρ_{1S}	ρ_{0B}	ρ_{1B}	Nombre total de points	Erreur maximale (TPs) ϵ_λ^{max} (BDD 1)	Erreur moyenne (TPs) ϵ_λ^{moy} (BDD 1)
$r_k + 1$ (TPs)		7	7	7	7	7	7	117 649		
Méthode itérative	p_k	2	2	2	2	2	2	100 842	$7.2835e^{-2}$	$1.2135e^{-3}$
	n_k	6	6	6	6	6	5			
HERO	p_k	2	2	2	2	2	2	100 842	$3.0691e^{-2}$ (ERASMUS)	$7.7238e^{-4}$ (ERASMUS)
	n_k	6	6	6	6	6	5		$3.0676e^{-2}$ (*fmincon*)	$7.7206e^{-4}$ (*fmincon*)

Tableau 6.5 – Erreurs d'approximation sur l'ensemble des points cibles des métamodèles obtenus par HERO et une méthode itérative (Base de données des points cibles BDD 1)

On remarque également que la solution renvoyée par HERO donne un meilleur métamodèle de la charge critique de flambage que la méthode itérative. En effet, dans le cas de l'approximation, la méthode itérative n'est pas capable de respecter la contrainte fixée sur l'erreur d'approximation maximale ($7.3e^{-2} > \epsilon_{max}^{(max)} = 5e^{-2}$). La contrainte sur l'erreur moyenne d'approximation est, quant à elle, respectée ($1.2e^{-3} < \epsilon_{max}^{(moy)} = 5e^{-3}$). On note en revanche que, pour les mêmes nombre de points de contrôle et degrés, le métamodèle généré par HERO respecte, lui, les deux contraintes d'erreur d'approximation (maximale et moyenne). Ce qui différencie ces deux hypersurfaces NURBS, ce sont les valeurs des composantes internes des différents *knot-vectors* et, de même que dans les exemples présentés au chapitre 5, leurs impacts n'est clairement pas négligeable. Enfin, il est intéressant de noter que la solution renvoyée par l'algorithme ERASMUS est déjà très proche de l'optimum local renvoyé par l'algorithme déterministe. Pour ce métamodèle, l'erreur maximale d'approximation étant respectée, les poids n'ont pas été introduits dans les variables d'optimisation $\mathbf{x}_{\text{II}}$ du CNLPP (6.41).

Métamodèle BDD 1	ϵ_λ^{max} (TPs) BDD 1	ϵ_λ^{max} (validation) BDD 3	ϵ_λ^{max} (validation) BDD 4	ϵ_λ^{moy} (TPs) BDD 1	ϵ_λ^{moy} (validation) BDD 3	ϵ_λ^{moy} (validation) BDD 4
Méthode itérative	$7.2835e^{-2}$	1.0798	0.9299	$1.2135e^{-3}$	$1.4768e^{-2}$	$1.1976e^{-2}$
HERO	$3.0676e^{-2}$	$1.6569e^{-1}$	$2.4510e^{-1}$	$7.7206e^{-4}$	$1.2266e^{-2}$	$7.5474e^{-3}$

Tableau 6.6 – Erreurs d'approximation des métamodèles obtenus par HERO et une méthode itérative à partir de la base de données BDD 1

L'erreur maximale d'approximation souhaitée étant atteinte, le tableau 6.6 présente les erreurs d'approximation sur les différentes bases de données permettant de valider le métamodèle avant son utilisation. La première constatation est que la méthode itérative génère un métamodèle dont l'erreur maximale d'approximation est de l'ordre de 100%. Ces points aberrants correspondent à des zones de l'hypersurface souffrant d'*over-fitting*. En comparaison, le métamodèle généré par HERO est caractérisé par une erreur maximale d'approximation de 16.6% et de 24.1% sur les bases de données BDD 3 et BDD 4 respectivement. Cette erreur est supérieure à l'erreur d'approximation souhaitée pour le métamodèle mais elle est bien plus faible que celle de la méthode itérative. L'erreur moyenne d'approximation est, elle aussi, plus faible pour le métamodèle généré par HERO. Même si le métamodèle généré par HERO est bien meilleur que celui généré par la méthode itérative, l'erreur d'approximation est trop importante pour l'utiliser. Afin d'améliorer cette erreur, le choix a été fait d'introduire les poids parmi les variables d'optimisation $\mathbf{x}_{\text{II}}$ du CNLPP (6.41). Comme le seuil d'erreur maximale d'approximation est respecté pour le métamodèle généré par HERO, ce seuil ne peut pas être utilisé pour sélectionner les poids à introduire. Il n'est pas non plus possible d'introduire tous les poids de l'hypersurface car leur nombre est trop important. Nous avons donc choisi d'introduire les poids dans les zones ou l'erreur d'approximation est supérieure à $1e^{-2}$. Le nombre de poids introduits parmi les variables d'optimisation avec ce seuil est de 3192. Ce choix a été fait de manière à n'utiliser que la base de données de points cibles pour paramétrer le métamodèle. Il est cependant envisageable d'utiliser les autres bases de données pour déterminer les zones où l'erreur maximale d'approximation est supérieure au seuil fixé par l'utilisateur.

Métamodèle BDD 1	ϵ_λ^{max} (TPs) BDD 1	ϵ_λ^{max} (validation) BDD 3	ϵ_λ^{max} (validation) BDD 4	ϵ_λ^{moy} (TPs) BDD 1	ϵ_λ^{moy} (validation) BDD 3	ϵ_λ^{moy} (validation) BDD 4
Méthode itérative	$7.2835e^{-2}$	1.0798	0.9299	$1.2135e^{-3}$	$1.4768e^{-2}$	$1.1976e^{-2}$
HERO $\boldsymbol{\Omega} = \emptyset$	$3.0676e^{-2}$	$1.6569e^{-1}$	$2.4510e^{-1}$	$7.7206e^{-4}$	$1.2266e^{-2}$	$7.5474e^{-3}$
HERO $\boldsymbol{\Omega} \neq \emptyset$	$3.0618e^{-2}$	$1.6564e^{-1}$	$2.4528e^{-1}$	$7.6177e^{-4}$	$1.2292e^{-2}$	$8.0544e^{-3}$

Tableau 6.7 – Comparaison des erreurs d'approximation des métamodèles générés avec introduction des poids à partir de la base de données BDD 1

Les résultats obtenus après introduction des poids sont présentés et comparés dans le tableau 6.7. Contrairement à ce qui aurait pu être attendu, les poids ont eut un effet très négligeable sur l'erreur d'approximation du métamodèle. Ce résultat se vérifie également sur la valeur des poids qui respectent la propriété suivante :

$$\forall \omega_{i_1,\ldots,i_6} \in \boldsymbol{\Omega}, \quad 0.99999 < \omega_{i_1,\ldots,i_6} < 1.00001. \tag{6.43}$$

Cela signifie que les valeurs de poids renvoyées après la résolution du CNLPP (6.41) sont très proches de 1. Les erreurs d'approximation, moyenne et maximale, ont cependant toutes deux

diminuées par rapport à la solution renvoyée sans introduction des poids. Toutefois, l'effet apporté est trop négligeable par rapport au temps supplémentaire de calcul engendré. D'autant plus que les erreurs d'approximation sur les points de validation n'ont pas diminuées mais, au contraire, augmentées dans la plupart des cas. En effet, mis à part l'erreur maximale d'approximation sur la base de données BDD 3 qui a très légèrement diminuée, toutes les autres ont augmentées. L'erreur d'approximation n'est donc toujours pas satisfaisante pour utiliser le métamodèle. Il semble alors pertinent d'utiliser toutes les données disponibles pour générer le métamodèle et donc de ne plus générer l'hypersurface NURBS dans le cadre de l'approximation mais dans le cadre de l'interpolation.

Variable		n_S	n_B	ρ_{0S}	ρ_{1S}	ρ_{0B}	ρ_{1B}	Nombre total de points	Erreur maximale (validation) ϵ_λ^{max}	Erreur moyenne (validation) ϵ_λ^{moy}
$r_k + 1$ (TPs)		7	7	7	7	7	7	117 649	(BDD 4)	(BDD 4)
HERO ($n_{pop} = 3$, $n_{ind} = 200$, $n_{gen} = 150$)	p_k	1	6	6	1	6	3	117 649	$1.9224e^{-1}$ (ERAS-MUS)	$1.6636e^{-2}$ (ERAS-MUS)
	n_k	6	6	6	6	6	6		$1.8538e^{-1}$ (*fmincon*)	$1.9321e^{-2}$ (*fmincon*)

Tableau 6.8 – Erreurs d'approximation du métamodèle généré par HERO dans le cas de l'interpolation des points cibles de la base de données BDD 1

Dans le cas de l'interpolation, on ne peut plus vraiment parler de méthode "itérative" puisque le nombre de points de contrôle est fixé, alors même que les itérations ne concernent que l'évolution de ce nombre de points. C'est pourquoi, le tableau 6.8 ne présente que les résultats du métamodèle généré par HERO. De plus, en interpolation, l'erreur d'approximation aux points cibles est théoriquement nulle. De ce fait, les erreurs maximale et moyenne d'approximation doivent être estimées directement sur un ensemble de points différents des points cibles. L'erreur étant maximale pour les points de la base de données BDD 4, dans le cas de l'appoximation, nous avons décider de l'utiliser pour le paramétrage du métamodèle dans le cas de l'interpolation. Le tableau 6.8 montre les résultats obtenus dans ce cas. En particulier, on peut remarquer que les degrés p_k renvoyés par ERASMUS, dans ce cas, ne sont plus égaux à 2. On remarque également que l'erreur maximale d'approximation sur la base de données BDD 4 est inférieure à celle du cas de l'approximation.

Le tableau 6.9 présente les erreurs d'approximation sur les base de données BDD 3 et BDD 4 pour le métamodèle renvoyé par HERO et les paramètres de la méthode itérative. Contrairement au cas de l'approximation, cette fois-ci l'erreur maximale d'approximation de la méthode itérative atteint jusqu'à 200%. On peut également remarquer que les erreurs moyennes d'approximation ont été augmentées par rapport au cas de l'approximation. Il semble donc que, pour ce cas particulier, le phénomène d'*over-fitting* ait été accentué par l'augmentation du nombre de points de contrôle avec les paramètres de la méthode itérative. En revanche, concernant le

Métamodèles BDD 1 (interpolation)	ϵ_λ^{max} (validation) BDD 3	ϵ_λ^{max} (validation) BDD 4	ϵ_λ^{moy} (validation) BDD 3	ϵ_λ^{moy} (validation) BDD 4
Méthode itérative	1.3541	2.1786	$1.9201e^{-2}$	$1.6747e^{-2}$
HERO	$1.8793e^{-1}$	$1.8538e^{-1}$	$1.5930e^{-2}$	$1.9321e^{-2}$

Tableau 6.9 – Erreurs d'approximation des métamodèles obtenus par HERO et une méthode itérative à partir de la base de données BDD 1 dans le cas de l'interpolation

métamodèle généré par HERO, l'erreur maximale d'approximation, ainsi que l'erreur moyenne, ont été diminuées sur la base de données BDD 4, ce qui n'est pas étonnant puisqu'elle a servi à paramétrer le métamodèle. Les erreurs maximale et moyenne ont cependant été augmentées sur la base de données BDD 3, ce qui est plus surprenant. Cela peut venir d'un nombre trop faible de points cibles. Par ailleurs, même si l'erreur a été diminuée sur la base de données BDD 4, l'erreur d'approximation n'est toujours pas satisfaisante pour utiliser le métamodèle. Le nombre maximal de points cibles étant atteint, il est nécessaire d'utiliser une base de données de points cibles plus importante. Pour cela, nous allons maintenant utiliser la base de données BDD 2 pour calculer les coordonnées des points de contrôle.

Variable		n_S	n_B	ρ_{0S}	ρ_{1S}	ρ_{0B}	ρ_{1B}	Nombre total de points	Erreur maximale (TPs) ϵ_λ^{max} (BDD 2)	Erreur moyenne (TPs) ϵ_λ^{moy} (BDD 2)
$r_k + 1$ (TPs)		9	9	9	9	9	9	531 441		
Méthode itérative	p_k	2	2	2	2	2	2			
	n_k	8	8	8	7	7	7	373 248	$4.5862e^{-2}$	$1.8913e^{-3}$
	n_k	8	8	8	8	7	7	419 904	$2.3983e^{-2}$	$6.9836e^{-4}$
HERO ($n_{pop} = 3$, $n_{ind} = 200$, $n_{gen} = 150$)	p_k	2	2	2	2	2	2	413 343	$2.9113e^{-2}$ (ERASMUS)	$5.0604e^{-4}$ (ERASMUS)
	n_k	8	8	8	8	8	6		$1.8562e^{-2}$ (*fmincon*)	$5.2000e^{-4}$ (*fmincon*)

Tableau 6.10 – Erreurs d'approximation sur l'ensemble des points cibles des métamodèles obtenus par HERO et une méthode itérative (Base de données des points cibles BDD 2)

Le tableau 6.10 présente les résultats obtenus sur cette nouvelle base de données de points cibles. La première chose à noter est que, contrairement aux cas précédents, la solution renvoyée par ERASMUS était relativement éloignée de l'optimum local trouvée par la méthode déterministe. Ceci est certainement dû à la formulation du CNLPP (6.38). En effet, dans tous les cas précédents, le nombre de points de contrôle trouvé par ERASMUS était bien plus faible que celui des points cibles. Dans de telles conditions, la stratégie de traitement des contraintes

permet de discriminer convenablement les solutions les unes par rapport aux autres. Or, dans le cas présent, il y a beaucoup de solutions équivalentes, en termes de points de contrôle et de degrés, qui ne peuvent pas être discriminées par les contraintes. En effet, en considérant le vecteur $\mathbf{n} = (n_1, ..., n_6)$, toutes les solutions d'une permutation du vecteur $\mathbf{n}_{eq} = (8, 8, 8, 8, 8, 6)$ sont équivalentes, à degrés fixés, du point de vue de la fonction objectif. De plus, la plupart de ces solutions respecte les contraintes sur les erreurs d'approximations moyenne et maximale. Il n'est alors pas possibles de discriminer ces solutions. Dans un tel cas, il est envisageable de modifier la formulation du CNLPP (6.38) afin que l'erreur d'approximation apparaisse dans la fonction objectif.

Métamodèles BDD 2	ϵ_λ^{max} (validation) BDD 1	ϵ_λ^{max} (validation) BDD 3	ϵ_λ^{max} (validation) BDD 4	ϵ_λ^{moy} (validation) BDD 1	ϵ_λ^{moy} (validation) BDD 3	ϵ_λ^{moy} (validation) BDD 4
Méthode itérative (373 248 CP)	$3.2122e^{-1}$	$2.4586e^{-1}$	$2.4432e^{-1}$	$6.1155e^{-3}$	$6.3467e^{-3}$	$4.5903e^{-3}$
Méthode itérative (419 904 CP)	1.6034	2.0662	$5.5500e^{-1}$	$1.5097e^{-2}$	$5.0774e^{-2}$	$1.2273e^{-2}$
HERO	1.2608	6.7240	1.2431	$3.0939e^{-2}$	$1.3816e^{-1}$	$2.7101e^{-2}$

Tableau 6.11 – Erreurs d'approximation des métamodèles obtenus par HERO et une méthode itérative à partir de la base de données 2

Les erreurs d'approximation sur les bases de données n'ayant pas servis à paramétrer le métamodèle sont fournies dans le tableau 6.11. Les deux résultats de la méthode itérative correspondent à deux itérations consécutives. Comme on peut le constater, que ce soit pour la méthode itérative ou pour HERO, il semblerait que diminuer l'erreur d'approximation sur l'ensemble des points cibles se traduise, dans ce cas, par un phénomène d'*over-fitting*. En effet, l'erreur maximale d'approximation du métamodèle généré par HERO sur l'ensemble des points cibles est inférieure à 2% tandis que l'erreur commise sur les autres bases de données atteint les 600%. Le même constat peut être fait sur l'erreur moyenne d'approximation qui admet jusqu'à deux ordres de grandeur d'écart entre les points cibles et les points de validation. En comparaison, la méthode itérative ayant moins de points de contrôle admet des erreurs plus grandes au niveau des points cibles mais bien plus faible sur les autres points. Le métamodèle n'est clairement pas utilisable et deux possibilités peuvent être envisagées :

- Les erreurs, maximale et moyenne, d'approximation des contraintes du CNLPP (6.38) ne sont plus seulement calculées pour les points cibles mais également sur un ensemble de point de validation. Cela permet notamment de pouvoir intégrer l'information de points échantillonnés de manière désordonnée.
- Générer un métamodèle d'interpolation des points cibles, les erreurs étant alors calculées directement sur un ensemble de points de validation.

Variable		n_S	n_B	ρ_{0S}	ρ_{1S}	ρ_{0B}	ρ_{1B}	Nombre total de points	Erreur maximale (validation) ϵ_λ^{max} (BDD 1)	Erreur moyenne (validation) ϵ_λ^{moy} (BDD 1)
$r_k + 1$ (TPs)		9	9	9	9	9	9	531 441		
HERO ($n_{pop} = 3$, $n_{ind} = 200$, $n_{gen} = 150$)	p_k	2	2	2	2	2	3	531 441	$2.2992e^{-1}$ (ERAS-MUS)	$2.1759e^{-3}$ (ERAS-MUS)
	n_k	8	8	8	8	8	8		$2.2977e^{-1}$ (*fmincon*)	$2.1391e^{-3}$ (*fmincon*)

Tableau 6.12 – Erreurs d'approximation du métamodèle généré par HERO dans le cas de l'interpolation des points cibles de la base de données BDD 2

L'approche que nous avons choisie de suivre est l'interpolation. La base de données permettant de paramétrer le métamodèle est la BDD 2. Le tableau 6.12 présente les résultats du métamodèle généré par HERO. De même que dans le cas de l'interpolation de la base de données BDD 1, les résultats de la méthode itérative ne sont pas exposés dans ce tableau puisque le nombre de points de contrôle ne peut pas évoluer. Le tableau 6.13 compare, quant à lui, les erreurs d'approximation des méthodes sur les bases de données disponibles. Les erreurs sur la base de données BDD 2 sont théoriquement nulle en interpolation. En réalité, l'erreur commise est de l'ordre de 10^{-13} à cause des approximations numériques. Cependant, il est clair que ces erreurs sont négligeables par rapport aux erreurs commises sur les autres bases de données.

Métamodèles BDD 2 (interpolation)	ϵ_λ^{max} (validation) BDD 1	ϵ_λ^{max} (validation) BDD 3	ϵ_λ^{max} (validation) BDD 4	ϵ_λ^{moy} (validation) BDD 1	ϵ_λ^{moy} (validation) BDD 3	ϵ_λ^{moy} (validation) BDD 4
Méthode itérative	2.7215	12.2499	$5.5500e^{-1}$	$1.8681e^{-2}$	$6.6582e^{-2}$	$1.4733e^{-2}$
HERO	$2.2977e^{-1}$	$2.2456e^{-1}$	$2.1623e^{-1}$	$2.1391e^{-3}$	$4.7298e^{-3}$	$3.5354e^{-3}$

Tableau 6.13 – Erreurs d'approximation des métamodèles obtenus par HERO et une méthode itérative dans le cas de l'interpolation de la base de données BDD 2

Comme on pouvait s'y attendre, les paramètres de la méthode itérative ont conduit à un métamodèle souffrant grandement du phénomène d'*over-fitting*. A contrario, utiliser une seconde base de données pour paramétrer le metamodèle avec HERO a permis d'obtenir un métamodèle beaucoup moins impacté par le phénomène. On remarque d'ailleurs que l'erreur moyenne d'approximation est inférieure d'un ordre de grandeur sur les bases de données BDD 3 et BDD 4 par rapport au métamodèle généré à partir de la base de données BDD 1. L'erreur maximale d'approximation est, quant à elle, supérieure par rapport à ce même métamodèle. Cependant, une analyse des zones où l'erreur est maximale permet de remarquer qu'elles se situent à proximité

de la frontière du domaine de faisabilité (i.e. des contraintes liées à l'existence de solutions pour le CNLPP (6.14), cf. éq. (6.13). Afin d'utiliser le métamodèle, le CNLPP (6.13) a été modifié pour prendre en compte le domaine de validité du métamodèle. Il est à noter que cette erreur peut être due au choix des points cibles. En effet, le choix de la fonction permettant d'obtenir les paramètres sans dimensions d'entrée de l'hypersurface ne prend pas en compte la morphologie de l'espace de conception. De ce fait, il est possible que les valeurs aux limites du domaine de faisabilité ne fassent pas partie des points cibles. Utiliser une paramétrisation qui prend en compte la topologie de l'espace de conception est une perspective intéressante pour pallier ces problèmes.

6.4 Résolution du problème d'optimisation de premier niveau de la stratégie MS2L avec le métamodèle

Le métamodèle généré par HERO n'étant pas valide à proximité des frontières du domaine de faisabilité du CNLPP (6.13), celui-ci a été modifié afin de prendre en compte l'erreur commise :

$$\begin{gathered}\min_{\boldsymbol{\xi}} \frac{M(\boldsymbol{\xi})}{M_{ref}}, \\ \text{sujet à :} \\ \begin{cases} 1.05 - \dfrac{\lambda(\boldsymbol{\xi})}{\lambda_{ref}} \leq 0, \\ 2\left(\rho_{1S}\right)^2 - \rho_{0S} - (1-\alpha) \leq 0, \\ 2\left(\rho_{1B}\right)^2 - \rho_{0B} - (1-\alpha) \leq 0, \end{cases}\end{gathered} \tag{6.44}$$

avec α un paramètre permettant de déplacer la frontière du domaine de faisabilité des stratifiés afin de prendre en considération le domaine de validité du métamodèle et $\boldsymbol{\xi}$ le vecteur des variables d'optimisation (cf. éq. (6.12)).

Afin de résoudre le CNLPP (6.44), la stratégie HERO développée dans cette thèse peut être appliquée. Dans ce cas, l'utilisation du métamodèle permet d'avoir accès aux dérivées de l'hypersurface NURBS directement (cf. éq. (3.13)) et ainsi d'accélérer le calcul lors de la seconde phase de la stratégie HERO (i.e. résolution du problème d'optimisation par une méthode déterministe). Toutefois, même si le nombre de variables d'optimisation est constant, les variables géométriques, à savoir n_S et n_B, sont des variables appartenant à l'espace des entiers naturels. De ce fait, elles ne peuvent pas faire partie des variables d'optimisation d'une méthode déterministe. Or la masse de l'UR est seulement liée à ces variables, ce qui implique que le problème résolu par la fonction *fmincon* doit être reformulé.

Afin d'appliquer la stratégie HERO, le CNLPP (6.44) est traité par ERASMUS dans un premier temps. La structure d'un individu pour la résolution de ce problème est donnée à la fig. 6.4. La charge critique de flambage est donnée par le métamodèle généré par HERO à partir de la base de données de points cibles BDD 2 en interpolation. Cette première étape a pour objectif de fixer les valeurs de n_S et n_B, ainsi que de fournir un bon point de départ pour la méthode déterministe utilisée dans un second temps.

FIGURE 6.4 – Strucutre d'un individu d'ERASMUS pour la résolution du CNLPP (6.44)

Les variables n_S et n_B étant des variables entières, elles ne peuvent pas être intégrées parmi les variables d'optimisation d'une méthode déterministe. Or, la fonction objectif du CNLPP (6.44), qui ne fait intervenir que la masse du panneau, ne dépend pas des variables ρ_{0S}, ρ_{1S}, ρ_{0B} et ρ_{1B}. De ce fait, le second CNLPP résolu a pour objectif de maximiser la charge critique de flambage pour les valeurs de n_S et n_B fournies par ERASMUS. Ce CNLPP est de la forme suivante :

$$\begin{array}{c} \max_{\boldsymbol{\xi}_m} \dfrac{\lambda\left(\boldsymbol{\xi}_m\right)}{\lambda_{ref}}, \\ \text{sujet à :} \\ \begin{cases} 2\left(\rho_{1S}\right)^2 - 1 - \rho_{0S} \leq 0, \\ 2\left(\rho_{1B}\right)^2 - 1 - \rho_{0B} \leq 0, \end{cases} \end{array} \tag{6.45}$$

Afin de résoudre le CNLPP (6.44), les différents paramètres d'ERASMUS sont les suivants :

1. Le nombre de populations a été fixé à $n_{pop} = 3$
2. Le nombre d'individus a été fixé à $n_{ind} = 200$
3. Le critère d'arrêt est le nombre d'itérations, fixé à $n_{gen} = 200$ pour ce cas.
4. La probabilité de croisement est fixée à la valeur par défaut $p_{cross} = 0.85$.
5. La probabilité de mutation des gènes est fixée à une valeur de $p_{mut} = 0.005$.
6. Le type de sélection retenu est la méthode de la *roulette*.
7. Le paramètre de *fitness pressure* est fixé à $f_{pres} = 1$, afin de ne pas brider l'exploration du domaine.
8. L'opérateur d'*élitisme* est actif.
9. Le nombre de générations pendant lesquelles les populations évoluent de manière isolée les unes des autres est fixé à $I_{time} = 5$.
10. Le nombre de contraintes est, dans le cas du CNLPP (6.44), $n_{constr} = 3$.
11. Le nombre de chromosomes de la partie standard des individus est de $n_{chrom}^{std} = 1$ (il n'y a pas de sections modulaires).
12. Le nombre de gènes de la partie standard correspondant aux six variables d'optimisation du vecteur $\boldsymbol{\xi}$, soit $n_{gene}^{std} = 6$.
13. Le nombre de variables différentes est $n_{var} = 3$. Il s'agit des nombres de plis n_α ainsi que des modules sans dimension d'anisotropies des matrices de rigidité réduites $\rho_{0\alpha}$ et $\rho_{1\alpha}$ $(\alpha = S, B)$:

— *Nombre de plis* :

- *Nom de la variable* : N_{plis}
- *Nature de la variable* : entière.
- *La limite inférieure de la variable* : $n_\alpha^{min} = 20$.
- *La limite supérieure de la variable* : $n_\alpha^{max} = 32$.
- *Le pas de discrétisation* : 1

— *Module d'anisotropie* ρ_0 :

- *Nom de la variable* : ρ_0
- *Nature de la variable* : continue (*extended*), dans ce cas, une discrétisation est réalisée automatiquement par ERASMUS en fonction de la précision machine (i.e. pas de pas de discrétisation à fixer).
- *La limite inférieure de la variable* : -1.
- *La limite supérieure de la variable* : 1.

— *Module d'anisotropie* ρ_1 :

- *Nom de la variable* : ρ_1
- *Nature de la variable* : continue (*extended*), dans ce cas, une discrétisation est réalisée automatiquement par ERASMUS en fonction de la précision machine (i.e. pas de pas de discrétisation à fixer).
- *La limite inférieure de la variable* : 0.
- *La limite supérieure de la variable* : 1.

Paramètre	Peau			Raidisseur			Masse [kg]	λ [N]
	n_S	ρ_{0S}	ρ_{1S}	n_B	ρ_{0B}	ρ_{1B}		
Référence	28	-	-	32	-	-	0.9194	466 830
HERO	28	-0.1852	0.0066	29	0.7952	0.7416	0.8799 (-4.30%)	495 855 (+6.22%)

Tableau 6.14 – Résultats de l'optimisation du panneau raidi par HERO couplé au métamodèle de la charge critique de flambage

Le CNLPP (6.45) est, quant à lui, traité par la fonction *fmincon* de MATLAB avec les paramètres par défaut, la fonction objectif étant sans dimension. Les résultats de l'optimisation du panneau raidi par la stratégie HERO sont présentés dans le tableau 6.14. On peut constater un gain de masse de 4.3% par rapport à la solution de référence et une rigidité améliorée de 6.2%. La valeur de la charge critique de flambage est la valeur fournie par le métamodèle. Afin de valider la solution trouvée, la charge critique de flambage a également été évaluée à l'aide du modèle EF qui a servi à l'obtention des bases de données de points cibles. Le tableau 6.15 présente les valeurs fournies par ces modèles. On peut remarquer que la valeur renvoyée par le métamodèle

est très proche de la valeur calculée par le modèle EF, ce qui permet de valider la configuration des paramètres géométriques et mécaniques trouvés par HERO pour l'UR du panneau. Le temps nécessaire à ERASMUS pour obtenir cette solution est d'environ deux minutes alors que le même problème, résolu en utilisant le modèle EF qui a permis de générer la base de données de points cibles, nécessite une centaine d'heure de calcul. Le temps d'évaluation du métamodèle permet donc d'améliorer grandement le temps nécessaire au processus d'optimisation.

	Métamodèle HERO interpolation BDD 2	Modèle EF ANSYS
Charge critique de flambage [N]	495 855	496 186

Tableau 6.15 – Charge critique de flambage de la solution optimisée

Il est cependant important de noter que le temps nécessaire à la génération de la base de données reste, lui, plutôt élevé. Il serait donc intéressant de comparer les résultats obtenus en utilisant directement la stratégie HERO interfacée avec le modèle EF décrit à la section 6.3.1. En particulier, le temps nécessaire à la résolution du CNLPP (6.13) pourrait être comparé à celui nécessaire à la création du métamodèle. En effet, dans notre cas, un grand nombre de points cibles a été nécessaire pour obtenir un métamodèle utilisable. Dans un tel cas, le nombre de points cibles nécessaires à l'obtention d'un bon métamodèle ne doit pas dépasser le nombre nécessaire d'évaluations de la fonction objectif pour obtenir une bonne solution. Pour cela, une perspective envisagée est le changement de la paramétrisation utilisée pour obtenir les variables sans dimension d'entrée de l'hypersurface NURBS afin de s'adapter à la morphologie de l'espace des points cibles. Une autre perspective intéressante est d'utiliser la phase d'optimisation d'ERASMUS pour obtenir les points cibles permettant de générer le métamodèle. En effet, dans un tel cas, le métamodèle serait obtenu à partir de densités de points cibles différentes selon les zones de l'espace de conception et, en particulier, les densités seront plus importantes dans les zones proches d'un optimum. Cela implique, cependant, d'établir des stratégies de calcul des coordonnées des points de contrôle n'impliquant pas le stockage de matrices de quelques téraoctets.

6.5 Conclusions

Dans ce chapitre, un problème d'optimisation dont la complexité est de niveau industriel a été présenté. Une stratégie de résolution originale (MS2L) de ce type de problème a été présentée et décrite brièvement. Le problème de temps de calcul de cette approche a été mis en avant et l'utilisation d'un métamodèle a été proposée. La stratégie de réduction de modèle employée est la stratégie développée dans le cadre de ces travaux de thèse. En particulier, la stratégie HERO a été appliquée pour obtenir l'estimation de la charge critique de flambage d'un panneau raidi en fonction de ces variables de conception géométriques et mécaniques. Le métamodèle obtenu a ensuite été utilisé afin de minimiser la masse dudit panneau tout en garantissant un seuil minimal de la charge critique de flambage de la structure.

Concernant la génération du métamodèle, plusieurs conclusions sont apparues. Tout d'abord, la première base de données utilisée était composée de trop peu de points et n'a pas permis d'obtenir un métamodèle utilisable. Les métamodèles obtenus par notre stratégie étaient tout de même bien meilleurs que ceux obtenus par la méthode itérative à laquelle ils ont été comparés. Les métamodèles obtenus n'étant pas satisfaisant, les poids de l'hypersurface NURBS ont été introduits parmi les variables d'optimisation. Cependant, contrairement à ce qu'on aurait pu attendre, leurs impacts sur le métamodèle obtenu était quasiment négligeable. Ce résultat est toutefois à relativiser. En effet, seuls les poids ont été introduits parmi les variables d'optimisation, alors que les coordonnées des points de contrôle ne l'ont pas été. Il serait intéressant, pour de futurs travaux, d'introduire, parmi les variables d'optimisation, les coordonnées des points de contrôle correspondant aux poids introduits. Il est à noter que la résolution de ce problème ne converge, en général, pas avec une méthode déterministe. De ce fait, cette résolution doit être faite avec une métaheuristique.

Une autre conclusion est que la formulation du problème, dans le cas de l'interpolation, permet de prendre en considération des points n'ayant pas été obtenus de manière ordonnée. En effet, s'il est préférable que les points à interpoler soient ordonnés, ce n'est pas le cas des points permettant de calculer l'erreur maximale d'approximation. Il est tout à fait envisageable d'étendre cette idée au cas de l'approximation. C'est une perspective intéressante mais qui nécessite quand même, au moins, un premier ensemble de points cibles ordonnés. L'erreur d'approximation, servant à paramétrer le métamodèle, peut ensuite être calculée à la fois sur cet ensemble et sur un autre ensemble pouvant ne pas être ordonné.

Enfin, la dernière conclusion sur les métamodèles générés concerne le nombre de points cibles nécessaires à l'obtention d'un modèle réduit utilisable. Ce nombre étant relativement élevé, la question de l'utilité de réaliser le métamodèle se pose. Pour cela, il est envisagé de comparer les résultats obtenus avec une méthode de résolution utilisant directement le modèle éléments finis. La comparaison se fera, notamment, sur le nombre d'évaluations nécessaires à obtenir une bonne solution par rapport au nombre de points cibles nécessaires à l'obtention d'un métamodèle utilisable. Un changement de paramétrisation entre les variables d'entrée du métamodèle et les paramètres sans dimension de l'hypersurface NURBS est également envisagé.

Pour ce qui est du problème d'optimisation traité, il est intéressant de remarquer que l'absence d'hypothèses restrictives, notamment sur les orientations des plis des stratifiés, permet des gains significatifs de masse et de rigidité. N'étant pas l'objet de ce chapitre, le problème de second niveau n'a pas été résolu pour la solution du problème de premier niveau. Il serait cependant très intéressant de trouver des stratifiés permettant d'obtenir les caractéristiques fournies par la résolution du problème de premier niveau utilisant notre métamodèle. Le développement de l'algorithme génétique ERASMUS offre une autre perspective intéressante pour ce problème d'optimisation. En effet, malgré une efficacité prouvée de la stratégie MS2L, le formalisme utilisé pour décrire le génotype d'un individu dans ERASMUS permet d'envisager de résoudre directement le problème d'optimisation du panneau raidi. Il serait intéressant de comparer les résultats obtenus par une telle approche avec les résultats de la stratégie MS2L.

Chapitre 7

Conclusions et perspectives générales

Une nouvelle méthode de réduction de modèle, basée sur le formalisme des hypersurfaces NURBS, a été présentée dans ce manuscrit. L'originalité de l'approche proposée réside dans le fait qu'aucune hypothèse simplificatrice ou règle empirique ne sont utilisées *a priori* pour fixer les paramètres de l'hypersurface. La méthode proposée a pour but de minimiser le nombre de paramètres définissant l'hypersurface tout en garantissant un certain seuil d'erreur d'approximation (au moins égal à celui obtenu par les méthodes itératives actuelles). Le problème de réduction de modèle qui en résulte est un problème d'*hypersurface fitting*, d'une entité NURBS sur ensemble de points cibles, pouvant être mis sous la forme d'un problème d'optimisation. L'inconvénient majeur, lorsque tous les paramètres de l'hypersurface NURBS font partie des variables d'optimisation, est que le problème est défini sur un espace de dimension variable. En effet, le nombre de variables d'optimisation n'est pas constant et est directement lié à certains des paramètres entiers de l'hypersurface. Après avoir introduit la notion de *système modulaire*, nous avons mis en évidence le fait que le problème d'*hypersurface fitting* pouvait être mis sous la forme d'un problème d'optimisation de système modulaire possédant plusieurs types de modules indépendants.

La résolution de ce problème a donné lieu au développement d'une nouvelle version de l'algorithme génétique BIANCA, qui est une métaheuristique capable de traiter des problèmes d'optimisation de systèmes modulaires ayant un seul type de module. Nous avons nommé ce nouvel algorithme ERASMUS (*EvolutionaRy Algorithm for optimiSation of ModUlar Systems*). Il a la particularité de pouvoir traiter des problèmes d'optimisation de systèmes modulaires possédant plusieurs types de composants modulaires, contrairement à son prédécesseur. En particulier, le concept d'*espèce* a été étendu et généralisé de telle manière que l'espèce d'un individu n'est plus seulement liée au nombre de chromosomes composant son génotype, mais aux nombres de chromosomes composants chacun de ses composants modulaires. Ceci a été rendu possible par une nouvelle formulation du génotype des individus, i.e. la représentation de l'information génétique dans ERASMUS. Cette nouvelle formulation a également conduit à généraliser les opérateurs génétiques permettant la reproduction, i.e. les phases de croisement et de mutation, entre deux individus d'espèces différentes. Cela permet à l'algorithme ERASMUS de déterminer automatiquement et simultanément les *meilleures espèces* et les *meilleurs individus* parmi ces meilleures espèces. De plus, ERASMUS bénéficie d'une stratégie très générale de traitement des contraintes

(égalité comme inégalité), à savoir, la méthode ADP (*Automatic Dynamic Penalisation*) qui s'adapte *automatiquement* au problème traité sans intervention de l'utilisateur (i.e. indépendant du problème considéré) et *dynamiquement* puisque l'évaluation du niveau de pénalisation est mis à jour à chaque génération. Enfin, même si l'algorithme ERASMUS a été utilisé pour résoudre le problème d'*hypersurface fitting* dans le cas de la réduction de modèle, il est présenté de manière générale et peut être utilisé pour résoudre une grande variété de problèmes d'optimisation.

ERASMUS a été développé dans le langage MATLAB dans le but de pouvoir être interfacé avec d'autres logiciels. De cette manière, ERASMUS est capable d'échanger des informations d'entrée et de sortie avec des modèles mathématiques externes. Cela permet notamment à ERASMUS de traiter des problèmes d'optimisation où les valeurs des fonctions objectifs et/ou des fonctions contraintes ne peuvent pas être calculées analytiquement. C'est un cas classique lorsqu'on est confronté à un problème d'optimisation structurel où, la plupart du temps, la réponse mécanique (et/ou thermique, magnétique, ...) de la structure est numériquement évaluée par le biais d'un code de calcul par éléments finis. Dans notre cas, les routines permettant de calculer les points d'une hypersurface NURBS, ainsi que les coordonnées des points de contrôle, ont été codées dans le langage C++ dans un soucis de temps de calcul. Grâce à la versatilité du langage MATLAB, il a notamment été possible d'interfacer ERASMUS avec ces routines par le biais de fichiers "mex" qui permettent d'utiliser des fonctions codées en langage C, C++ ou Fortran comme des fonctions classiques de MATLAB. Cela permet de ne pas utiliser de fichiers "texte" afin d'échanger les informations d'entrée et de sortie des routines utilisées. Nous avons également développé une architecture parallélisée d'ERASMUS permettant le calcul simultané de plusieurs individus, grâce notamment à la fonction "parfor" de MATLAB qui remplace les boucles "for" classiques.

Malgré le fort potentiel d'ERASMUS, l'introduction des poids de l'hypersurface NURBS parmi les variables d'optimisation rend le problème très difficile à traiter. Cependant, plutôt que de fixer les valeurs de l'ensemble des poids à 1 (utilisation d'hypersurface B-Spline), ou de les fixer par des règles empiriques, nous avons développé une stratégie indépendante des données traitées permettant de déterminer le nombre de poids à introduire parmi les variables d'optimisation de manière automatique. Pour parvenir à ce résultat une méthode hybride très générale de résolution de problèmes d'optimisation, la méthode HERO (*Hybrid EvolutionaRy Optimisation*), a été développée. Elle s'appuie sur l'utilisation de la métaheuristique ERASMUS et d'une méthode déterministe de résolution de problèmes d'optimisation. La méthode est présentée de manière très générale (sans hypothèses simplificatrices) et peut être appliquée à un grand nombre de problèmes d'optimisation, qu'ils concernent des systèmes modulaires ou non, et se destine, bien évidemment, à des problèmes d'optimisation non-convexes. L'idée principale est d'utiliser, dans un premier temps, l'algorithme génétique ERASMUS, et ainsi utiliser les capacités exploratoires de ce type de métaheuristiques, afin de trouver une solution proche d'un optimum (local ou global). Dans le cas de systèmes modulaires, cette première étape permet également de fixer le nombre de variables d'optimisation. Dans un second temps, une méthode déterministe de résolution est utilisée pour atteindre l'optimum le plus proche.

Cette stratégie a été appliquée pour la résolution du problème d'optimisation résultant de notre problème de réduction de modèle. Dans ce cas, l'algorithme génétique ERASMUS est utilisé, dans un premier temps, afin de fixer les paramètres entiers de l'hypersurface NURBS et,

ainsi, fixer le nombre de variables d'optimisation. Les poids sont tous fixés à 1 dans cette première exploration, i.e. seulement des hypersurfaces B-Splines sont utilisées. Suite à cette étape, si le seuil d'erreur d'approximation du métamodèle n'est pas respecté, seuls les poids impactant les zones de l'espace où le seuil d'erreur d'approximation n'est pas respecté sont introduits parmi les variables d'optimisation. Ceci est rendu possible grâce au caractère intrinsèquement local des entités NURBS. L'avantage de cette approche est que le métamodèle n'est pas complexifié inutilement et que le choix d'utilisation d'entité B-Splines ou NURBS n'incombe pas à l'utilisateur (i.e. la stratégie détermine elle même s'il est nécessaire d'introduire ou non les poids). Une seconde exploration à nombre de variables d'optimisation constant est effectué par ERASMUS afin de déterminer une solution proche d'un optimum (local ou global). Enfin, la solution renvoyée sert de point de départ pour une méthode de résolution déterministe et permet ainsi d'atteindre l'optimum le plus proche.

La versatilité et l'efficacité de la méthode de réduction de modèle proposée dans cette thèse ont été présentées sur des *benchmarks* aux caractéristiques particulières ainsi que sur un problème de complexité industriel, à savoir, l'évaluation de la charge critique de flambage d'un panneau raidi. Les résultats obtenus par notre approche ont été comparés à ceux fournis par une méthode itérative issue de la littérature et utilisant les règles empiriques classiques héritées des courbes et surfaces NURBS. Notre méthode a permis, pour tous les cas traités, d'obtenir de bien meilleurs résultats que les méthodes itératives existantes. Cette constatation n'a rien de surprenant puisque les règles empiriques utilisées par ces méthodes ont été créées dans le seul but d'améliorer la stabilité numérique des calculs au détriment de l'optimisation des entités NURBS obtenues. Les principaux résultats obtenus peuvent être résumés de la manière suivante :

- **Problème thermique d'estimation du champ de température d'une plaque mince avec épaisseur et coefficient de convection variables** (chapitre 5) : le point clé de ce problème est son invariance par rapport à l'un des paramètres d'entrée du métamodèle. Dans ce cas, les méthodes itératives existantes ne sont pas capables de réduire efficacement le nombre de points de contrôle nécessaires, principalement par la nature du choix de la direction d'ajout des points de contrôle. Les résultats présentés pour ce cas d'application montrent que notre méthode est capable de réduire de manière significative le nombre de points de contrôle nécessaires par rapport à une méthode itérative classique (jusqu'à 82 fois moins de points de contrôle), pour la même erreur d'approximation du métamodèle. Ceci est simplement dû à l'absence d'hypothèses sur les paramètres de l'hypersurface NURBS.
- **Problème mécanique d'estimation du champ de déplacement d'une plaque mince avec épaisseur et norme de l'effort appliqué variables** (chapitre 5) : les principaux attraits de ce benchmark sont, d'une part, que l'un des paramètres d'entrée du métamodèle est une condition limite du problème considéré et, d'autre part, que le métamodèle doit générer plusieurs sorties. Comme dans le cas précédent, notre approche a montré des performances supérieures à celles de la méthode itérative à laquelle elle est comparée. Le nombre de points de contrôle nécessaires pour notre approche a, en particulier, été réduit par un facteur 24 par rapport à la méthode itérative.
- **Problème mécanique d'estimation du champ de déplacement d'une plaque mince trouée avec épaisseur et rayon du trou variables** (chapitre 5) : l'enjeu majeur de ce problème est que la morphologie de l'espace des points cibles varie avec l'une des entrées du

métamodèle. Dans ce cas d'application, notre approche a montré sa capacité à traiter de tels problèmes et a également exhibé de meilleures performances que la solution itérative à laquelle elle a été comparée. En particulier, le nombre de points de contrôle nécessaires à notre approche a été réduit d'un facteur allant jusqu'à 296 par rapport à la méthode itérative.

- **Problème mécanique d'estimation de la charge critique de flambage d'un panneau raidi** (chapitre 6) : ce problème est d'un niveau de complexité industriel. En particulier, la fonction du phénomène à l'étude est non-linéaire et la topologie de l'espace des points cibles varie avec les paramètres d'entrée du métamodèle. Contrairement aux autres cas d'application, celui-ci a nécessité l'introduction des poids parmi les variables d'optimisation dans la seconde phase de la stratégie HERO. Cependant, contrairement aux résultats attendus, l'introduction des poids a eu un impact peu significatif sur le métamodèle obtenu. Dans le cadre des courbes et surfaces NURBS, l'introduction des poids est connu pour résoudre, ou tout au moins diminuer, les problèmes d'effets "vaguelettes" (effets d'oscillation de la courbe ou de la surface). Il n'est donc pas si étonnant que l'introduction des poids ait eu pour effet de diminuer l'erreur moyenne plutôt que l'erreur maximale d'approximation du métamodèle. De plus, ce résultat est à relativiser. En effet, seuls les poids sont introduits parmi les variables d'optimisation mais il serait peut être pertinent d'ajouter les coordonnées des points de contrôle liés à ces poids parmi les variables d'optimisation.

A ces résultats, nous pouvons ajouter quelques remarques générales. Tout d'abord, nous avons remarqué que la solution renvoyée par ERASMUS est souvent proche du minimum atteint par la méthode déterministe (fonction "fmincon" de MATLAB). Il pourrait être intéressant de dresser un abaque permettant de fixer le nombre minimal d'évaluations de la fonction objectif (permettant d'obtenir un bon point de départ pour la méthode déterministe) en fonction du nombre d'entrées et de points cibles disponibles pour le problème traité. Cependant, il ne paraît pas invraisemblable de penser que ce nombre est lié à la nature du problème traité et pas seulement aux nombres d'entrées et de points cibles considérés. Il est également à noter que le choix des points cibles était purement arbitraire dans les cas d'applications présentés et que l'impact de ces choix n'a pas été quantifié. Enfin, il est important de noter que la principale limitation de la méthode, actuellement, est la résolution du système d'équations linéaires permettant d'obtenir les coordonnées des points de contrôle. En effet, la taille des matrices mises en jeu implique d'utiliser des stratégies qui brident les possibilités d'échantillonnage de l'espace de conception.

Ces remarques permettent de faire le lien avec les perspectives de cette thèse. En effet, l'une des perspectives les plus importantes de ces travaux concerne le couplage de cette méthode de réduction de modèle avec des méthodes d'échantillonnage de l'espace de conception ne nécessitant pas d'ensemble de points cibles ordonnés. Pour cela, quatre voies sont privilégiées :

1. L'ajout des coordonnées paramétriques aux coordonnées des points de contrôle. En effet, l'hypersurface NURBS utilisée actuellement est une application d'un espace de dimension N (correspondant au nombre d'entrées du métamodèle) dans un espace de dimension M (correspondant au nombre de sorties). Pour cela, une fonction bijective lie les paramètres d'entrée du métamodèle aux paramètres paramètres sans dimension de l'hypersurface NURBS. L'idée est donc de réaliser une hypersurface NURBS d'un espace de dimension N (relatif au nombre d'entrées) dans un espace de dimension $N + M$ (somme

des dimensions de l'espace des entrées et des sorties). Cela nécessite d'affecter de nouvelles valeurs aux paramètres sans dimension des points cibles et la méthode du *chord length* est une piste envisageable. Il est également possible d'utiliser les abscisses de Gréville, qui permettent de déterminer les N premières coordonnées des points de contrôle en fonction des différents *knot vectors* de manière à ce qu'un point de l'hypersurface NURBS évalué en une coordonnée x_k $(k = 1, ..., N)$ corresponde à cette coordonnée x_k. Ce choix permet de s'adapter directement à la topologie de l'espace des paramètres sans avoir à filtrer des points de contrôle "inactifs".

2. L'utilisation de "patchs" locaux. Malgré un comportement intrinsèque local des entités NURBS, une grande variété de méthodes utilisant des approximations ou interpolations locales existent, tout au moins dans le cas des courbes et surfaces. Ceci répond principalement à la limitation imposée par les tailles des matrices à stocker pour le calcul des coordonnées des points de contrôle. En effet, l'aspect local du patch permet de diminuer fortement le nombre de points de contrôle ainsi que le nombre de points cibles mis en jeu. De ce fait, la formulation générale décrite à la section 4.3 est utilisable. Il est alors possible d'utiliser des ensembles de points non-ordonnées afin d'obtenir les coordonnées des points de contrôle. Il est même possible, dans ce cas, de développer une méthode d'échantillonnage définissant elle même le nombre de patch et les densités de points cibles nécessaires, en utilisant une information sur le gradient de la fonction considérée par exemple. Cependant, même si la difficulté n'est plus dans le stockage des matrices, elle est reportée au niveau des raccordements des différents patchs NURBS qui est loin d'être une tâche triviale.
3. L'implémentation de méthodes d'inversion de matrices ne nécessitant pas le stockage de matrices de grandes tailles. Afin de résoudre le problème de stockage de la matrice à inverser pour le calcul des coordonnées des points de contrôle, dans le cas de la formulation générale qui permet l'utilisation de points cibles non-ordonnés, il peut être envisagé d'utiliser des méthodes similaires ou issues des techniques utilisées dans les logiciels de calcul par éléments finis. En effet, les problèmes traités lors de modélisation par éléments finis mettent en jeu des matrices creuses de très grandes dimensions et des techniques spécifiques permettent d'optimiser, à la fois, les temps de calcul et l'espace de stockage des données.
4. L'ajout d'une première couche d'abstraction. L'idée principale est de réaliser un premier métamodèle très "basique" permettant d'obtenir des points cibles ordonnées à partir d'une base de données de points cibles non-ordonnées. Dans le cas d'une modélisation par éléments finis, cela se traduit par l'utilisation des fonctions de forme pour obtenir des points cibles en dehors des points du maillage. Dans le cas d'un maillage très fin, et dans l'hypothèse où le nombre de points cibles ordonnés extraits à partir du maillage est bien plus faible que le nombre de nœuds, il est envisageable d'utiliser une méthode telle que *le plus proche voisin* (*nearest neighboor*). Il apparaît cependant fort probable que ce genre d'approche nécessite un nombre très important de points cibles pour être utilisée. Il faut toutefois noter que le problème de stockage de matrice intervient lorsque le nombre de points cibles, et de points de contrôle, sont élevés.

Le problème d'influence des poids, soulevé par le cas d'application présenté au chapitre 6, amène une autre perspective intéressante. En effet, notre méthode est capable de déterminer elle même les poids à introduire parmi les variables d'optimisation. Cependant, nous avons choisi,

comme c'est le cas dans la littérature, de ne pas introduire les coordonnées des points de contrôle correspondantes aux poids introduits parmi les variables d'optimisation. Il pourrait être intéressant d'introduire ces coordonnées parmi les variables d'optimisation afin d'en quantifier l'effet. Il faut toutefois noter que, dans ce cas, le nombre de variables d'optimisation peut devenir très grand. En effet, si n_{poids} est le nombre de poids introduits parmi les variables d'optimisation et M est le nombre de coordonnées des points de contrôle (ou le nombre de sorties du métamodèle), alors le nombre de variables d'optimisation supplémentaire est de $(M+1) \times n_{poids}$.

Une autre perspective intéressante, est de comparer notre approche à d'autres méthodes de réduction de modèle issues de la littérature. En effet, bien que les méthodes itératives de réduction de modèle basées sur les hypersurfaces NURBS aient déjà été comparées à d'autres méthodes existantes, notre approche a montré de bien meilleurs résultats que ces méthodes itératives. Il semble notamment intéressant de comparer nos résultats à ceux de méthodes plus vastement utilisées comme le krigeage, la POD (*Proper Orthogonal Decomposition*) ou la PGD (*Proper Generalized Decomposition*). La comparaison pourrait être faite sur des critères tels que la complexité du paramétrage à réaliser par l'utilisateur, le temps de calibration du modèle réduit, le temps d'évaluation du métamodèle obtenu, le nombre de données nécessaires à l'évaluation ou encore la précision d'approximation par exemple.

En outre, la stratégie de réduction de modèle présentée dans cette thèse a nécessité le développement d'une stratégie d'optimisation (HERO) ainsi que d'une métaheuristique de type algorithme génétique (ERASMUS), appliquées à notre problème de réduction de modèle formulé de manière très générale. De ce fait, ces deux méthodes offrent un grand potentiel pour la résolution de problèmes d'optimisation de systèmes modulaires. En particulier, le problème d'optimisation du panneau raidi présenté dans le chapitre 6 pourrait être résolu directement par le biais d'ERASMUS. En effet, l'unité répétitive du panneau raidi peut être vue comme un système modulaire composé de deux types de modules. Le premier type de module correspond à un pli composant le stratifié de la peau tandis que le second type de module correspond à un pli composant le stratifié du raidisseur. Il est donc tout à fait envisageable de résoudre ce problème d'optimisation par le biais d'ERASMUS sans utiliser la stratégie MS2L. Il serait d'ailleurs intéressant de comparer les résultats obtenus à ceux obtenus par la stratégie MS2L.

Pour conclure, nous pouvons dire que les travaux réalisés dans cette thèse ont permis, d'une part, de proposer une méthode de réduction de modèle basée sur les hypersurfaces NURBS originale ayant dépassée les limites classiquement imposées par les règles empiriques héritées des courbes et surfaces et, d'autre part, de développer une stratégie d'optimisation basée sur un algorithme génétique original spécialement conçu pour les problèmes d'optimisation de systèmes modulaires possédant plusieurs types de modules.

Annexe A

Codes développés en C++ pour une utilisation interfacée avec MATLAB

En complément des pseudo-codes fournis dans le chapitre 3, nous fournissons ici les trois principaux codes permettant de construire un point d'une hypersurface NURBS. Les versions proposées sont des versions *standalone* (i.e. directement utilisable dans MATLAB en les compilant). Toutefois, à des fins de rapidité d'exécution notamment, nous utilisons toutes ces routines dans un autre format. En effet, nous nous servons de la possibilité offerte par le langage C++ d'utiliser les fichiers *header* (fichier de type .h) permettant de regrouper l'ensemble des fonctions utilisées dans un seul fichier. Cela permet notamment, en phase de développement, de ne modifier les fonctions qu'à un seul endroit, sans avoir à passer en revue les codes précédemment établis.

- *Indice de portée.* Ce code est celui permettant de calculer l'indice de portée, ou *span index*, à partir des paramètres de l'hypersurface NURBS et de la coordonnée paramétrique dans une direction de l'espace paramétrique. Il renvoi une valeur d'indice adaptée à une numérotation d'indice débutant à 0 (comme c'est le cas en C++). Il peut être intéressant de noter que le type de variable renvoyé est un flottant de précision double pour cette version *standalone* tandis que, dans la fonction permettant le calcul d'un point de l'hypersurface, la valeur renvoyée est de type entier. Ceci est dû au fait qu'un entier renvoyé par la fonction "mex" est convertie en type double dans l'espace de travail de MATLAB.

```
/*==========================================================
 * This function compute the span index at parametric location u
 *
 [span]= findSpan_mex(n,p,u,U);
 *
 * This is the line to compile the function into matlab (need c++
 * compiler):
 *
 mex findSpan_mex.cpp
 *
 * This is a MEX-file for MATLAB.
 *=========================================================*/

#include "mex.h"
```

```
15 #include "matrix.h"
16
17 // findSpan function
18 double findspan(int n, int p, double u, double *U)
19 {
20 // This algorithm return the knot span index ("span") at the point "u" to be
       calculated
21 // Inputs:
22 //          - n : max index of control points for the B-Spline
23 //          - p : degree of the B-Spline polynomial
24 //          - u : parametric point
25 //          - U : Knot vector
26 // Output:
27 //          - span : knot span index at the parametric point u
28 // Algorithm A2.1 "The NURBS Book" p. 68
29
30     int low, high, mid;
31     // special case
32     if (u == U[(mwSize)n+1]){ return((double)n);}
33     // do binary search
34     low = p;
35     high = n + 1;
36     mid = (low + high) / 2;
37     while ( (u < U[(mwSize)mid] || u >= U[(mwSize)mid+1]) )
38     {
39         if (u < U[(mwSize)mid])
40         {
41             high = mid;
42         }
43         else
44         {
45             low = mid;
46         }
47         mid = (low + high) / 2;
48     }
49     return((double)mid);
50 }
51
52 //
       ////////////////////////////////////////////////////////////////////////////////


53
54 // mex that replace main
55 void mexFunction( int nlhs, mxArray *plhs[],
56                   int nrhs, const mxArray *prhs[])
57 {
58     /* Variable declarations here */
59     int n, p, i;
60     double u;
61     double *U;
62
63     double *span;
64
```

```
65      /* check for proper number of arguments */
66      if(nrhs!=4) {
67          mexErrMsgIdAndTxt("MyToolbox:arrayProduct:nrhs","4 inputs required.")
                ;
68      }
69      if(nlhs!=1) {
70          mexErrMsgIdAndTxt("MyToolbox:arrayProduct:nlhs","1 output required.")
                ;
71      }
72
73      /* get the value of the scalar inputs */
74      n = (int) mxGetScalar(prhs[0]);
75      p = (int) mxGetScalar(prhs[1]);
76
77      u = (double) mxGetScalar(prhs[2]);
78
79      /* create a pointer to the real data in the input matrix */
80      U = (double *) mxGetPr(prhs[3]);
81
82      /* create the output matrix */
83      plhs[0]  = mxCreateDoubleMatrix(1,1,mxREAL);
84
85      /* get a pointer to the real data in the output matrix */
86      span  = (double *) mxGetPr(plhs[0]);
87
88      /* call the computational routine */
89      *span = findspan(n, p, u, U);
90
91      return;
92  }
```

- *Fonctions de bases.* Le code, fourni ici, permet calculer uniquement les $p_i + 1$ valeurs non nulles des fonctions de base, à partir des paramètres de l'hypersurface NURBS, de l'indice de portée et de la coordonnée paramétrique dans une direction de l'espace paramétrique. Cette version est également une version *standalone* et nécessite d'avoir au préalable calculé l'indice de portée.

```
1   /*
        ==========================================================================

2    * This function compute the non-null basis function at parametric location u
3    *
4    [N]= basisFun_mex(span,u,p,U);
5    *
6    * This is the line to compile the function into matlab (need c++
7    * compiler):
8    *
9    mex basisFun_mex.cpp
10   *
11   * This is a MEX-file for MATLAB.
12   *==========================================================================
         */
13
```

```
14 #include "mex.h"
15 #include "matrix.h"
16
17 // basisFun function
18 void basisfun(int i, double u, int p, double *U, double *N)
19 {
20 // This algorithm return the non-null basis functions at the point "u" to be
21 // calculated
22 // Inputs:
23 //          - i : knot span index at the parametric point u from the function
24 //                "findspan"
25 //          - p : degree of the B-Spline polynomial
26 //          - u : parametric point
27 //          - U : Knot vector
28 // Output:
29 //          - N : Basis functions vector [p+1]
30 // Algorithm A2.2 from "The NURBS Book" p. 70
31     int j,r;
32     double saved, temp;
33     // memory allocation
34     double *left  = (double *) mxCalloc(mwSize(p+1),sizeof(double));     //
          new double[p+1];
35     double *right = (double *) mxCalloc(mwSize(p+1),sizeof(double));     //
          new double[p+1];
36     N[0] = 1.0;
37     for (j = 1; j <= p; j++)
38     {
39         left[j]  = u - U[(mwSize)(i+1-j)];
40         right[j] = U[(mwSize)(i+j)] - u;
41         saved = 0.0;
42
43         for (r = 0; r < j; r++)
44         {
45             temp = N[r] / (right[r+1] + left[j-r]);
46             N[r] = saved + right[r+1] * temp;
47             saved = left[j-r] * temp;
48         }
49
50         N[j] = saved;
51     }
52
53     // memory release
54     mxFree(left);   // delete[] left;
55     mxFree(right);  // delete[] right;
56 }
57
58 //
    ////////////////////////////////////////////////////////////////////////////////


59
60 // mex that replace main
61 void mexFunction( int nlhs, mxArray *plhs[],
62                   int nrhs, const mxArray *prhs[])
```

```
{
    /* Variable declarations here */
    int i, p;
    double u, *U, *N;

    /* check for proper number of arguments */
    if(nrhs!=4) {
        mexErrMsgIdAndTxt("MyToolbox:arrayProduct:nrhs","4 inputs required.")
            ;
    }
    if(nlhs!=1) {
        mexErrMsgIdAndTxt("MyToolbox:arrayProduct:nlhs","1 output required.")
            ;
    }

    /* get the value of the scalar inputs */
    i = (int) mxGetScalar(prhs[0]);
    p = (int) mxGetScalar(prhs[2]);

    u = (double) mxGetScalar(prhs[1]);

    /* create a pointer to the real data in the input matrix */
    U = (double *) mxGetPr(prhs[3]);

    /* create the output matrix */
    plhs[0]  = mxCreateDoubleMatrix(1,mwSize(p+1),mxREAL);

    /* get a pointer to the real data in the output matrix */
    N  = (double *) mxGetPr(plhs[0]);

    /* call the computational routine */
    basisfun(i, u, p, &U[0], &N[0]);

    return;
}
```

- *Point de l'hypersurface NURBS.* Ce code permet de calculer les coordonnées d'un point de l'hypersurface NURBS, **H**, en fonction des paramètres de celle-ci et des coordonnées paramétriques. La version proposée est un exemple d'utilisation d'une hypersurface dont la dimension de l'espace paramétrique est $N = 6$ et dont la dimension de l'hypersurface est $M = 2$. Cette version est également une version *standalone* et permet, notamment, le calcul de la réponse du métamodèle de l'application du chapitre 6.

```
/*
 ==========================================================================

 * This function compute the NURBS hypersurface point for a parametric
     dimension
 * N = 6 and a hypersurface dimension M = 2
 *
[H]= hyperSurfacePoint_N6_M2_mex(n1,n2,n3,n4,n5,n6, p1,p2,p3,p4,p5,p6,...
    u1,u2,u3,u4,u5,u6, U1,U2,U3,U4,U5,U6, P1,P2);
 *
```

```
 * This is the line to compile the function into matlab (need c++
 * compiler):
 *
mex hyperSurfacePoint_N6_M2_mex.cpp
 *
Inputs:
    - n : max index of control points for the B-Spline
    - p : degree of the B-Spline polynomial
    - u : parametric point
    - U : Knot vector
Output:
    - span : knot span index at the parametric point u
 *
 * This is a MEX-file for MATLAB.
 *=================================================================
     */

#include "mex.h"
#include "matrix.h"

// findSpan function
int findspan(int n, int p, double u, double *U)
{
// This algorithm return the knot span index ("span") at the point "u" to be
    calculated
// Inputs:
//          - n : max index of control points for the B-Spline
//          - p : degree of the B-Spline polynomial
//          - u : parametric point
//          - U : Knot vector
// Output:
//          - span : knot span index at the parametric point u
// Algorithm A2.1 "The NURBS Book" p. 68 adapted to the mex format of matlab

    int low, high, mid;
    // special case
    if (u == U[(mwSize)n+1]){ return(n);}
    // do binary search
    low = p;
    high = n + 1;
    mid = (low + high) / 2;
    while ( (u < U[(mwSize)mid] || u >= U[(mwSize)mid+1]) )
    {
        if (u < U[(mwSize)mid])
        {
            high = mid;
        }
        else
        {
            low = mid;
        }
        mid = (low + high) / 2;
    }
```

```
59      return(mid);
60  }
61
62  // basisFun function
63  void basisfun(int i, double u, int p, double *U, double *N)
64  {
65  // This algorithm return the non-null basis functions at the point "u" to be
66  // calculated
67  // Inputs:
68  //          - i : knot span index at the parametric point u from the function
69  //                "findspan"
70  //          - p : degree of the B-Spline polynomial
71  //          - u : parametric point
72  //          - U : Knot vector
73  // Output:
74  //          - N : Basis functions vector [p+1]
75  // Algorithm A2.2 "The NURBS Book" p. 70 adapted to the mex format of matlab
76      int j,r;
77      double saved, temp;
78      // memory allocation (mex version of new)
79      double *left  = (double *) mxCalloc(mwSize(p+1),sizeof(double));
80      double *right = (double *) mxCalloc(mwSize(p+1),sizeof(double));
81      N[0] = 1.0;
82      for (j = 1; j <= p; j++)
83      {
84          left[j]  = u - U[(mwSize)(i+1-j)];
85          right[j] = U[(mwSize)(i+j)] - u;
86          saved = 0.0;
87
88          for (r = 0; r < j; r++)
89          {
90              temp = N[r] / (right[r+1] + left[j-r]);
91              N[r] = saved + right[r+1] * temp;
92              saved = left[j-r] * temp;
93          }
94
95          N[j] = saved;
96      }
97
98      // memory release (mex version of delete)
99      mxFree(left);
100     mxFree(right);
101 }
102
103 // linearIndex_mex function
104 mwSize linearIndex_mex(int *vecCoor, int *vecSize, int dim)
105 {/* This algorithm return the linear index of a multidimensional array. Note
        that
106     the index is given for a linear index sorted by columns (matlab indexing)
            that
107     is different from C++ indexing (sorted by row). It acts like sub2ind
            matlab
108     function with no vectorization of the sub indexes.
```

```
109     Inputs:
110         - vecCoor: vector of "dim" values containing the coordinates of the
111                    desired linear index
112         - vecSize: vector of "dim" values containing the size of the
113                    multidimensional array in every direction
114         - dim: is the dimension of the array.
115                For a vector dim = 1 ; for a matrix dim = 2 ...
116     Outputs:
117         - idx: the linear index matching the coordinates vecCoor
118 */
119 int idx(0), prod(0);
120 
121 if(dim == 1)
122 {
123     idx = vecCoor[0];
124 }
125 else if (dim == 2)
126 {
127     idx = vecCoor[1]*vecSize[0] + vecCoor[0];
128 }
129 else
130 {
131     prod = vecSize[0];
132     idx = vecCoor[0] + vecCoor[1]*prod;
133     prod *= vecSize[1];
134     for (int i(2) ; i<dim ; i++)
135     {
136         idx += vecCoor[i]*prod;
137         prod *= vecSize[i];
138     }
139 }
140 return (mwSize)idx;
141 }
142 
143 // hyperSurfacePoint function
144 void surfacepoint_N6_M2(int n1,int p1,double *U1, int n2,int p2,double *U2,
    int n3,int p3,double *U3, int n4,int p4,double *U4, int n5,int p5,double
    *U5, int n6,int p6,double *U6, double *P1, double *P2, double u1, double
    u2, double u3, double u4, double u5, double u6, double *H)
145 {
146     // This algorithm return the surface point value S at parametric point u,
        v
147     // Inputs:
148     //          - ni : max index of control points along i-th axis
149     //          - pi : degree of the B-Spline polynomial along i-th axis
150     //          - Ui : Knot vector [ni+pi+2] for the i-th axis
151     //          - ui : parametric point along i-th axis
152     //          - Pj : [n1+1]...[n6+1] Control points matrix along the j-th
        axis
153     //                 of the hypersurface
154     // Output:
155     //          - H : Hyper-surface point coordinates
156     // Algorithm inspired from algorithm A3.5 ("The NURBS Book" p.103)
```

```
157     int uind1(0),uind2(0),uind3(0),uind4(0),uind5(0),uind6(0), uspan1(0),
            uspan2(0),uspan3(0),uspan4(0),uspan5(0),uspan6(0),dim(6);
158     int *vecCoor = (int *) mxCalloc(mwSize(dim),sizeof(int));
159     int vecSize[] = {n1+1,n2+1,n3+1,n4+1,n5+1,n6+1};
160     double *Nu1   = (double *) mxCalloc(mwSize(p1 + 1),sizeof(double));
161     double *Nu2   = (double *) mxCalloc(mwSize(p2 + 1),sizeof(double));
162     double *Nu3   = (double *) mxCalloc(mwSize(p3 + 1),sizeof(double));
163     double *Nu4   = (double *) mxCalloc(mwSize(p4 + 1),sizeof(double));
164     double *Nu5   = (double *) mxCalloc(mwSize(p5 + 1),sizeof(double));
165     double *Nu6   = (double *) mxCalloc(mwSize(p6 + 1),sizeof(double));
166     double *temp1 = (double *) mxCalloc(mwSize(dim),sizeof(double));
167     double *temp2 = (double *) mxCalloc(mwSize(dim),sizeof(double));
168
169
170     uspan1 = findspan(n1, p1, u1, U1);
171     basisfun(uspan1, u1, p1, U1, Nu1);
172     uspan2 = findspan(n2, p2, u2, U2);
173     basisfun(uspan2, u2, p2, U2, Nu2);
174     uspan3 = findspan(n3, p3, u3, U3);
175     basisfun(uspan3, u3, p3, U3, Nu3);
176     uspan4 = findspan(n4, p4, u4, U4);
177     basisfun(uspan4, u4, p4, U4, Nu4);
178     uspan5 = findspan(n5, p5, u5, U5);
179     basisfun(uspan5, u5, p5, U5, Nu5);
180     uspan6 = findspan(n6, p6, u6, U6);
181     basisfun(uspan6, u6, p6, U6, Nu6);
182     uind1 = uspan1 - p1;
183     uind2 = uspan2 - p2;
184     uind3 = uspan3 - p3;
185     uind4 = uspan4 - p4;
186     uind5 = uspan5 - p5;
187     uind6 = uspan6 - p6;
188
189     temp1[dim-1] = 0.0;
190     temp2[dim-1] = 0.0;
191     for (int i6(0) ; i6 <= p6 ; i6++)
192     {
193         vecCoor[dim-1] = uind6 + i6;
194         temp1[dim-2] = 0.0;
195         temp2[dim-2] = 0.0;
196         for (int i5(0) ; i5 <= p5 ; i5++)
197         {
198             vecCoor[dim-2] = uind5 + i5;
199             temp1[dim-3] = 0.0;
200             temp2[dim-3] = 0.0;
201             for (int i4(0) ; i4<=p4 ; i4++)
202             {
203                 vecCoor[dim-3] = uind4+i4;
204                 temp1[dim-4] = 0.0;
205                 temp2[dim-4] = 0.0;
206                 for (int i3(0) ; i3<=p3 ; i3++)
207                 {
208                     vecCoor[dim-4] = uind3+i3;
```

```
209                         temp1[dim-5] = 0.0;
210                         temp2[dim-5] = 0.0;
211                         for (int i2(0) ; i2 <= p2; i2++)
212                         {
213                             vecCoor[dim-5] = uind2+i2;
214                             temp1[dim-6] = 0.0;
215                             temp2[dim-6] = 0.0;
216                             for (int i1(0) ; i1 <= p1; i1++)
217                             {
218                                 vecCoor[dim-6] = uind1+i1;
219                                 temp1[dim-6] += Nu1[i1] * P1[linearIndex_mex(
                                        vecCoor,vecSize,dim)];
220                                 temp2[dim-6] += Nu1[i1] * P2[linearIndex_mex(
                                        vecCoor,vecSize,dim)];
221                             }
222                             temp1[dim-5] += Nu2[i2] * temp1[dim-6];
223                             temp2[dim-5] += Nu2[i2] * temp2[dim-6];
224                         }
225                         temp1[dim-4] += Nu3[i3] * temp1[dim-5];
226                         temp2[dim-4] += Nu3[i3] * temp2[dim-5];
227                     }
228                     temp1[dim-3] += Nu4[i4] * temp1[dim-4];
229                     temp2[dim-3] += Nu4[i4] * temp2[dim-4];
230                 }
231                 temp1[dim-2] += Nu5[i5] * temp1[dim-3];
232                 temp2[dim-2] += Nu5[i5] * temp2[dim-3];
233             }
234             temp1[dim-1] += Nu6[i6] * temp1[dim-2];
235             temp2[dim-1] += Nu6[i6] * temp2[dim-2];
236         }
237         H[0] = temp1[dim-1];
238         H[1] = temp2[dim-1];
239 
240         mxFree(vecCoor);
241         mxFree(Nu1);
242         mxFree(Nu2);
243         mxFree(Nu3);
244         mxFree(Nu4);
245         mxFree(Nu5);
246         mxFree(Nu6);
247         mxFree(temp1);
248         mxFree(temp2);
249         return ;
250     }
251 
252     //
        ////////////////////////////////////////////////////////////////////////////////
253 
254     // mex that replace main
255     void mexFunction( int nlhs, mxArray *plhs[],
256                       int nrhs, const mxArray *prhs[])
257     {
```

```
258     /* Variable declarations here */
259     int n1,n2,n3,n4,n5,n6;
260     int p1,p2,p3,p4,p5,p6;
261     double u1,u2,u3,u4,u5,u6;
262     double *U1,*U2,*U3,*U4,*U5,*U6;
263     double *P1, *P2, *H;
264 
265     /* check for proper number of arguments */
266     if(nrhs!=26) {
267         mexErrMsgIdAndTxt("MyToolbox:arrayProduct:nrhs","26 inputs required."
                ");
268     }
269     if(nlhs!=1) {
270         mexErrMsgIdAndTxt("MyToolbox:arrayProduct:nlhs","1 output required.")
                ;
271     }
272 
273     /* get the value of the scalar inputs */
274     n1 = (int) mxGetScalar(prhs[0]);
275     n2 = (int) mxGetScalar(prhs[1]);
276     n3 = (int) mxGetScalar(prhs[2]);
277     n4 = (int) mxGetScalar(prhs[3]);
278     n5 = (int) mxGetScalar(prhs[4]);
279     n6 = (int) mxGetScalar(prhs[5]);
280 
281     p1 = (int) mxGetScalar(prhs[6]);
282     p2 = (int) mxGetScalar(prhs[7]);
283     p3 = (int) mxGetScalar(prhs[8]);
284     p4 = (int) mxGetScalar(prhs[9]);
285     p5 = (int) mxGetScalar(prhs[10]);
286     p6 = (int) mxGetScalar(prhs[11]);
287 
288     u1 = (double) mxGetScalar(prhs[12]);
289     u2 = (double) mxGetScalar(prhs[13]);
290     u3 = (double) mxGetScalar(prhs[14]);
291     u4 = (double) mxGetScalar(prhs[15]);
292     u5 = (double) mxGetScalar(prhs[16]);
293     u6 = (double) mxGetScalar(prhs[17]);
294 
295     /* create a pointer to the real data in the input matrix */
296     U1 = (double *) mxGetPr(prhs[18]);
297     U2 = (double *) mxGetPr(prhs[19]);
298     U3 = (double *) mxGetPr(prhs[20]);
299     U4 = (double *) mxGetPr(prhs[21]);
300     U5 = (double *) mxGetPr(prhs[22]);
301     U6 = (double *) mxGetPr(prhs[23]);
302 
303     P1 = (double *) mxGetPr(prhs[24]);
304     P2 = (double *) mxGetPr(prhs[25]);
305 
306     /* create the output matrix */
307     plhs[0]  = mxCreateDoubleMatrix(1,2,mxREAL);
308 
```

```
309     /* get a pointer to the real data in the output matrix */
310     H  = (double *) mxGetPr(plhs[0]);
311 
312     /* call the computational routine */
313     surfacepoint_N6_M2(n1,p1,&U1[0], n2,p2,&U2[0], n3,p3,&U3[0], n4,p4,&U4
            [0], n5,p5,&U5[0], n6,p6,&U6[0], &P1[0],&P2[0], u1,u2,u3,u4,u5,u6, &H
            [0]);
314 
315     return;
316 }
```

Bibliographie

[1] Cameron J. Turner. *HyPerModels : hyperdimensional performance models for engineering design*. PhD thesis, 2005.

[2] Les Piegl and Wayne Tiller. *The NURBS Book*. Monograph in Visual Communication. Springer-Verlag Berlin Heidelberg, 1995.

[3] Marco Montemurro, Alfonso Pagani, Giacinto Alberto Fiordilino, Jérôme Pailhès, and Erasmo Carrera. A general multi-scale two-level optimisation strategy for designing composite stiffened panels. *Composite Structures*, 201 :968–979, October 2018.

[4] Francisco Chinesta, Roland Keunings, and Adrien Leygue. *The Proper Generalized Decomposition for Advanced Numerical Simulations*. SpringerBriefs in Applied Sciences and Technology. Springer International Publishing, Cham, 2014.

[5] Francisco Chinesta, Amine Ammar, and Elías Cueto. On the use of proper generalized decompositions for solving the multidimensional chemical master equation. *European Journal of Computational Mechanics*, 19(1-3) :53–64, January 2010.

[6] Emiliano Iuliano and Domenico Quagliarella. Proper Orthogonal Decomposition, surrogate modelling and evolutionary optimization in aerodynamic design. *Computers & Fluids*, 84 :327–350, September 2013.

[7] Anindya Chatterjee. An introduction to the proper orthogonal decomposition. *Current Science*, 78(7) :808–817, 2000.

[8] Régis Bettinger. *Inversion d'un système par krigeage : application à la synthèse des catalyseurs à haut débit*. PhD thesis, Université de Nice Sophia Antipolis, 2009.

[9] Michael James Sasena. Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations. page 237, 2002.

[10] Zhihua Zhang. Artificial Neural Network. In *Multivariate Time Series Analysis in Climate and Environmental Research*, pages 1–35. Springer, Cham, 2018.

[11] B. YEGNANARAYANA. *ARTIFICIAL NEURAL NETWORKS*. PHI Learning Pvt. Ltd., January 2009.

[12] Jérôme Lépione, François Guibault, Marie-Gabrielle Vallet, and Jean-Yves Trépanier. Optimization of a Curve Approximation Based on NURBS Interpolation. *International Conference on Curves and Surfaces [4th], Proceedings*, 1 :9, July 1999.

[13] Nallig Leal, Esmeide Leal, and John William Branch. Simple Method for Constructing NURBS Surfaces from Unorganized Points. In Suzanne Shontz, editor, *Proceedings of the*

19th International Meshing Roundtable, pages 161–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[14] M. Montemurro. *Optimal design of advanced engineering modular systems through a new genetic approach*. PhD thesis, Université Pierre et Marie Curie Paris VI, Paris, 2012.

[15] Angela Vincenti. *Conception et optimisation des composites stratifiés par méthode polaire et algorithme génétique*. PhD thesis, de Bourgogne, Dijon, France, January 2002.

[16] Marco Montemurro, Angela Vincenti, and Paolo Vannucci. A Two-Level Procedure for the Global Optimum Design of Composite Modular Structures—Application to the Design of an Aircraft Wing. *Journal of Optimization Theory and Applications*, 155(1) :24–53, October 2012.

[17] Marco Montemurro and Anita Catapano. A New Paradigm for the Optimum Design of Variable Angle Tow Laminates. In Aldo Frediani, Bijan Mohammadi, Olivier Pironneau, and Vittorio Cipolla, editors, *Variational Analysis and Aerospace Engineering : Mathematical Challenges for the Aerospace of the Future*, Springer Optimization and Its Applications, pages 375–400. Springer International Publishing, Cham, 2016.

[18] Marco Montemurro and Anita Catapano. On the effective integration of manufacturability constraints within the multi-scale methodology for designing variable angle-tow laminates. *Composite Structures*, 161 :145–159, February 2017.

[19] E. Barkanov, O. Ozoliņš, E. Eglītis, F. Almeida, M. C. Bowering, and G. Watson. Optimal design of composite lateral wing upper covers. Part I : Linear buckling analysis. *Aerospace Science and Technology*, 38 :1–8, October 2014.

[20] E. Barkanov, E. Eglītis, F. Almeida, M. C. Bowering, and G. Watson. Optimal design of composite lateral wing upper covers. Part II : Nonlinear buckling analysis. *Aerospace Science and Technology*, 51 :87–95, April 2016.

[21] C. Bisagni and L. Lanzi. Post-buckling optimisation of composite stiffened panels using neural networks. *Composite Structures*, 58(2) :237–247, November 2002.

[22] Luca Lanzi and Vittorio Giavotto. Post-buckling optimization of composite stiffened panels : Computations and experiments. *Composite Structures*, 73(2) :208–220, May 2006.

[23] DONALD R JONES. A Taxonomy of Global Optimization Methods Based on Response Surfaces. page 39, 2001.

[24] Jay Martin and Timothy Simpson. Use of Adaptive Metamodeling for Design Optimization. American Institute of Aeronautics and Astronautics, September 2002.

[25] H.-M. Gutmann. A Radial Basis Function Method for Global Optimization. *Journal of Global Optimization*, 19(3) :201–227, March 2001.

[26] G. Gary Wang and Timothy Simpson. Fuzzy clustering based hierarchical metamodeling for design space reduction and optimization. *Engineering Optimization*, 36(3) :313–335, June 2004.

[27] G. Gary Wang and S. Shan. Review of Metamodeling Techniques in Support of Engineering Design Optimization. *Journal of Mechanical Design*, 129(4) :370–380, April 2007.

[28] Russell R. Barton and Martin Meckesheimer. Chapter 18 Metamodel-Based Simulation Optimization. In Shane G. Henderson and Barry L. Nelson, editors, *Handbooks in Operations Research and Management Science*, volume 13 of *Simulation*, pages 535–574. Elsevier, January 2006.

[29] R. R. Barton. Simulation optimization using metamodels. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 230–238, December 2009.

[30] John Steuben, John Michopoulos, Athanasios Iliopoulos, and Cameron Turner. Inverse characterization of composite materials via surrogate modeling. *Composite Structures*, 132 :694–708, November 2015.

[31] Donald E. Myers. Spatial interpolation : an overview. *Geoderma*, 62(1) :17–28, March 1994.

[32] Brian D. Ripley. *Spatial Statistics*. Wiley Series in Probability and Statistics. Wiley Online Library, 1981.

[33] Vittorio Capecchi, Massimo Buscema, Pierluigi Contucci, and Bruno D'Amore, editors. *Applications of Mathematics in Models, Artificial Neural Networks and Arts*. Springer Netherlands, Dordrecht, 2010.

[34] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis : CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39-41) :4135–4195, October 2005.

[35] Gaetan Kerschen, Jean-claude Golinval, Alexander F. Vakakis, and Lawrence A. Bergman. The Method of Proper Orthogonal Decomposition for Dynamical Characterization and Order Reduction of Mechanical Systems : An Overview. *Nonlinear Dynamics*, 41(1-3) :147–169, August 2005.

[36] F. Chinesta, A. Ammar, A. Leygue, and R. Keunings. An overview of the proper generalized decomposition with applications in computational rheology. *Journal of Non-Newtonian Fluid Mechanics*, 166(11) :578–592, June 2011.

[37] Russell R. Barton. Metamodels for Simulation Input-output Relations. In *Proceedings of the 24th Conference on Winter Simulation*, WSC '92, pages 289–299, New York, NY, USA, 1992. ACM.

[38] Anthony A. Giunta. Aircraft Multidisciplinary Design Optimization using Design of Experiments Theory and Response Surface Modeling Methods. May 1997.

[39] Chen Wang, Qingyun Duan, Wei Gong, Aizhong Ye, Zhenhua Di, and Chiyuan Miao. An evaluation of adaptive surrogate modeling based optimization with two benchmark problems. *Environmental Modelling & Software*, 60 :167–179, October 2014.

[40] Wei Shyy, Nilay Papila, Rajkumar Vaidyanathan, and Kevin Tucker. Global design optimization for aerodynamics and rocket propulsion components. *Progress in Aerospace Sciences*, 37(1) :59–118, January 2001.

[41] Martin Meckesheimer, Andrew J. Booker, Russell R. Barton, and Timothy W. Simpson. Computationally Inexpensive Metamodel Assessment Strategies. *AIAA Journal*, 40(10) :2053–2060, October 2002.

[42] T. W. Simpson, J. D. Poplinski, P. N. Koch, and J. K. Allen. Metamodels for Computer-based Engineering Design : Survey and recommendations. *Engineering with Computers*, 17(2) :129–150, July 2001.

[43] Sophie Baillargeon. *Le krigeage : Revue de la théorie et application à l'interpolation spatiale de données de précipitation.* PhD thesis, Université de Laval Québec, Québec, April 2005.

[44] Noel A. C. Cressie. *Statistics for spatial data.* J. Wiley, 1993.

[45] M. Arnaud and X. Emery. *Estimation et interpolation spatiale : methodes deterministes et methodes geostatistiques.* Hermes, Paris, 2000.

[46] Gérard Collomb and Wolfgang Härdle. Strong uniform convergence rates in robust nonparametric time series analysis and prediction : Kernel regression estimation from dependent observations. *Stochastic Processes and their Applications*, 23(1) :77–89, October 1986.

[47] Y. P. Mack and B. W. Silverman. Weak and strong uniform consistency of kernel regression estimates. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 61(3) :405–415, September 1982.

[48] Jeffrey D. Hart and Thomas E. Wehrly. Kernel Regression Estimation Using Repeated Measurements Data. *Journal of the American Statistical Association*, 81(396) :1080–1088, December 1986.

[49] Jeffrey D. Hart. Kernel Regression Estimation With Time Series Errors. *Journal of the Royal Statistical Society. Series B (Methodological)*, 53(1) :173–187, 1991.

[50] Rice John. Boundary modification for kernel regression. *Communications in Statistics - Theory and Methods*, 13(7) :893–900, January 1984.

[51] Hiroyuki Takeda, Sina Farsiu, and Peyman Milanfar. *Kernel Regression for Image Processing and Reconstruction.* PhD thesis, February 2007.

[52] Yansheng Li, Yihua Tan, Jin-Gang Yu, Shengxiang Qi, and Jinwen Tian. Kernel regression in mixed feature spaces for spatio-temporal saliency detection. *Computer Vision and Image Understanding*, 135 :126–140, June 2015.

[53] F. Aurenhammer and H. Edelsbrunner. An optimal algorithm for constructing the weighted voronoi diagram in the plane. *Pattern Recognition*, 17(2) :251–257, January 1984.

[54] Alok Aggarwal, Leonidas J. Guibas, James Saxe, and Peter W. Shor. A linear-time algorithm for computing the voronoi diagram of a convex polygon. *Discrete & Computational Geometry*, 4(6) :591–604, December 1989.

[55] Jean-Daniel Boissonnat and Frédéric Cazals. Smooth surface reconstruction via natural neighbour interpolation of distance functions. *Computational Geometry*, 22(1) :185–203, May 2002.

[56] P. J. Green and R. Sibson. Computing Dirichlet Tessellations in the Plane. *The Computer Journal*, 21(2) :168–173, May 1978.

[57] B. N. Boots. Weighting Thiessen Polygons. *Economic Geography*, 56(3) :248–259, July 1980.

[58] A. Bowyer. Computing Dirichlet tessellations. *The Computer Journal*, 24(2) :162–166, January 1981.

[59] H. Imai, M. Iri, and K. Murota. Voronoi Diagram in the Laguerre Geometry and Its Applications. *SIAM Journal on Computing*, 14(1) :93–105, February 1985.

[60] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, Sung Nok Chiu, and D. G. Kendall. Spatial Tessellations : Concepts and Applications of Voronoi Diagrams, Second Edition. In *Wiley Series in Probability and Statistics*, pages i–xv. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2000.

[61] Robin Sibson. The Dirichlet Tessellation as an Aid in Data Analysis. *Scandinavian Journal of Statistics*, 7(1) :14–20, 1980.

[62] R. Sibson. A brief description of natural neighbour interpolation. *Series in Probability and Mathematical Statistics : Applied Probability and Statistics*, pages 21–36, 1981.

[63] E. Cueto, N. Sukumar, B. Calvo, M. A. Martínez, J. Cegoñino, and M. Doblaré. Overview and recent advances in natural neighbour galerkin methods. *Archives of Computational Methods in Engineering*, 10(4) :307–384, December 2003.

[64] Richard J. Kopec. An Alternative Method for the Construction of Thiessen Polygons. *The Professional Geographer*, 15(5) :24–26, September 1963.

[65] Thomas E. Croley and Holly C. Hartmann. Resolving Thiessen polygons. *Journal of Hydrology*, 76(3) :363–379, February 1985.

[66] Hugo Ledoux and Christopher Gold. An Efficient Natural Neighbour Interpolation Algorithm for Geoscientific Modelling. In *Developments in Spatial Data Handling*, pages 97–108. Springer, Berlin, Heidelberg, 2005.

[67] Bing She, Xinyan Zhu, Xinyue Ye, Wei Guo, Kehua Su, and Jay Lee. Weighted network Voronoi Diagrams for local spatial analysis. *Computers, Environment and Urban Systems*, 52 :70–80, July 2015.

[68] Halbert White. Maximum Likelihood Estimation of Misspecified Models. *Econometrica*, 50(1) :1, January 1982.

[69] Abou El-Makarim A. Aboueissa and Michael R. Stoline. Maximum likelihood estimators of population parameters from doubly left-censored samples. *Environmetrics*, 17(8) :811–826, December 2006.

[70] D. Zeng and D. Y. Lin. Maximum likelihood estimation in semiparametric regression models with censored data. *Journal of the Royal Statistical Society : Series B (Statistical Methodology)*, 69(4) :507–564, 2007.

[71] Frederick Crimins. Numerical recipes in C++ : The art of scientific computing. *Applied Biochemistry and Biotechnology*, 104(1) :95–96, January 2003.

[72] Jack P. C. Kleijnen and Robert G. Sargent. A methodology for fitting and validating metamodels in simulation1two anonymous referees' comments on the first draft lead to an improved organization of our paper.1. *European Journal of Operational Research*, 120(1) :14–29, January 2000.

[73] William S. Cleveland. Robust Locally Weighted Regression and Smoothing Scatterplots. *Journal of the American Statistical Association*, 74(368) :829–836, December 1979.

[74] William S. Cleveland and Susan J. Devlin. Locally Weighted Regression : An Approach to Regression Analysis by Local Fitting. *Journal of the American Statistical Association*, 83(403) :596–610, September 1988.

[75] D. G. Krige. A statistical approach to some basic mine valuation problems on the Witwatersrand. *Journal of the Southern African Institute of Mining and Metallurgy*, 52(6) :119–139, December 1951.

[76] G. Matheron. Le krigeage universel, 1969.

[77] G. Matheron. Le krigeage disjonctif. Technical Report N-360, Fontainebleau, December 1973.

[78] Hans Wackernagel. *Multivariate Geostatistics : An Introduction with Applications*. Springer Science & Business Media, February 2003.

[79] Jay D. Martin and Timothy W. Simpson. On the Use of Kriging Models to Approximate Deterministic Computer Models. volume 2004, pages 481–492. ASME, 2004.

[80] Jack P. C. Kleijnen. Kriging metamodeling in simulation : A review. *European Journal of Operational Research*, 192(3) :707–716, February 2009.

[81] Praveen Chandrashekarappa and Regis Duvigneau. Radial Basis Functions and Kriging Metamodels for Aerodynamic Optimization. report, INRIA, 2006.

[82] Martin Dietrich Buhmann. Radial basis functions. *Acta Numerica 2000*, 9 :1–38, 2000.

[83] Mohammed F. Hussain, Russel R. Barton, and Sanjay B. Joshi. Metamodeling : Radial basis functions, versus polynomials. *European Journal of Operational Research*, 138(1) :142–154, April 2002.

[84] David S. Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, DTIC Document, 1988.

[85] Zong-min Wu and Robert Schaback. Local error estimates for radial basis function interpolation of scattered data. *IMA journal of Numerical Analysis*, 13(1) :13–27, 1993.

[86] Shmuel Rippa. An algorithm for selecting a good value for the parameter c in radial basis function interpolation. *Advances in Computational Mathematics*, 11(2-3) :193–210, 1999.

[87] Anoop A Mullur and Achille Messac. Extended Radial Basis Functions : More Flexible and Effective Metamodeling. 2004.

[88] Anoop A. Mullur and Achille Messac. Extended Radial Basis Functions for Metamodeling : A Comparative Study. pages 743–757, January 2005.

[89] Emmanuel Viennet. Réseaux à fonctions de base radiales. *Apprentissage connexionniste*, page 105, 2006.

[90] S. Haykin. *Neural Networks : A Comprehensive Foundation, 2nd édition*. Prentice-Hall. New Jerey, 1999.

[91] J.E. Rayas-Sanchez. EM-Based Optimization of Microwave Circuits Using Artificial Neural Networks : The State-of-the-Art. *IEEE Transactions on Microwave Theory and Techniques*, 52(1) :420–435, January 2004.

[92] J. Sreekanth and Bithin Datta. Multi-objective management of saltwater intrusion in coastal aquifers using genetic programming and modular neural network based surrogate models. *Journal of Hydrology*, 393(3) :245–256, November 2010.

[93] Huizhuo Shi, Yuehua Gao, and Xicheng Wang. Optimization of injection molding process parameters using integrated artificial neural network model and expected improvement function method. *The International Journal of Advanced Manufacturing Technology*, 48(9-12) :955–962, June 2010.

[94] John Eason and Selen Cremaschi. Adaptive sequential sampling for surrogate model generation with artificial neural networks. *Computers & Chemical Engineering*, 68 :220–232, September 2014.

[95] Raymond H. Myers, Douglas C. Montgomery, and Christine M. Anderson-Cook. *Response Surface Methodology : Process and Product Optimization Using Designed Experiments*. John Wiley & Sons, 2009. Google-Books-ID : 89oznEFHF_MC.

[96] Kevin Gurney. *An Introduction to Neural Networks*. CRC Press, April 2014.

[97] Kishan Mehrotra, Chilukuri K. Mohan, and Sanjay Ranka. *Elements of Artificial Neural Networks*. MIT Press, 1997. Google-Books-ID : 6d68Y4Wq_R4C.

[98] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3) :273–297, September 1995.

[99] Stella M. Clarke, Jan H. Griebsch, and Timothy W. Simpson. Analysis of Support Vector Regression for Approximation of Complex Engineering Analyses. *Journal of Mechanical Design*, 127(6) :1077–1087, August 2004.

[100] Bernhard Scholkopf. Support Vector Machines : A Practical Consequence of Learning Theory. 1998.

[101] Harris Drucker, Christopher J C Burges, Linda Kaufman, Alex J Smola, and Vladimir Vapnik. Support Vector Regression Machines. page 7, 1997.

[102] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2) :415–425, March 2002.

[103] J A K SUYKENS and J VANDEWALLE. Least Squares Support Vector Machine Classifiers. page 8, 1999.

[104] J. M. Bourinet, F. Deheeger, and M. Lemaire. Assessing small failure probabilities by combined subset simulation and Support Vector Machines. *Structural Safety*, 33(6) :343–353, September 2011.

[105] O. L. Mangasarian. Support vector machine classification via parameterless robust linear programming. *Optimization Methods and Software*, 20(1) :115–125, February 2005.

[106] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on Information and knowledge management - CIKM '98*, pages 148–155, Bethesda, Maryland, United States, 1998. ACM Press.

[107] Guodong Guo, Stan Z. Li, and Kap Luk Chan. Support vector machines for face recognition. *Image and Vision Computing*, 19(9) :631–638, August 2001.

[108] CHRISTOPHER J C BURGES. A Tutorial on Support Vector Machines for Pattern Recognition. *SUPPORT VECTOR MACHINES*, page 47, 1998.

[109] Tony Van Gestel, Johan A. K. Suykens, Bart Baesens, Stijn Viaene, Jan Vanthienen, Guido Dedene, Bart de Moor, and Joos Vandewalle. Benchmarking Least Squares Support Vector Machine Classifiers. *Machine Learning*, 54(1) :5–32, January 2004.

[110] Hui-Yuan Fan, George S. Dulikravich, and Zhen-Xue Han. Aerodynamic data modeling using support vector machines. *Inverse Problems in Science and Engineering*, 13(3) :261–278, June 2005.

[111] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer Science & Business Media, September 2008. Google-Books-ID : HUnqnrpYt4IC.

[112] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel Methods in Machine Learning. *The Annals of Statistics*, 36(3) :1171–1220, 2008.

[113] Nello Cristianini, John Shawe-Taylor, Department of Computer Science Royal Holloway John Shawe-Taylor, and John (Royal Holloway Shawe-Taylor London), University of. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, March 2000. Google-Books-ID : _PXJn_cxv0AC.

[114] Bernhard Schölkopf, Alexander J. Smola, and Managing Director of the Max Planck Institute for Biological Cybernetics in Tubingen Germany Profe Bernhard Scholkopf. *Learning with Kernels : Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002. Google-Books-ID : y8ORL3DWt4sC.

[115] Grace Wahba. *Spline Models for Observational Data*. SIAM, January 1990. Google-Books-ID : BS6fGsebFNkC.

[116] Jean Duchon. Interpolation des fonctions de deux variables suivant le principe de la flexion des plaques minces. 10(3), 1976.

[117] Carl de Boor. A Practical Guide to Splines. *Mathematics of Computation*, 34(149) :325, January 1980.

[118] Jean Meinguet. Multivariate interpolation at arbitrary points made simple. *Zeitschrift für angewandte Mathematik und Physik ZAMP*, 30(2) :292–304, 1979.

[119] Ming-Jun Lai. Multivariate splines for data fitting and approximation. *Approximation Theory XII : San Antonio*, pages 210–228, 2007.

[120] D. Barrera, P. González, F. Ibáñez, and M.J. Ibáñez. A general spline differential quadrature method based on quasi-interpolation. *Journal of Computational and Applied Mathematics*, 275 :465–479, February 2015.

[121] Helena Mitasova and Lubos Mitas. Interpolation by regularized spline with tension : I. Theory and implementation. *Mathematical Geology*, 25(6) :641–655, August 1993.

[122] Helena Mitasova and Jaroslav Hofierka. Interpolation by regularized spline with tension : II. Application to terrain modeling and surface geometry analysis. *Mathematical Geology*, 25(6) :657–669, August 1993.

[123] M.N. Benbourhim and A. Bouhamidi. Approximation of vectors fields by thin plate splines with tension. *Journal of Approximation Theory*, 136(2) :198–229, October 2005.

[124] Jaroslav Hofierka, Juraj Parajka, Helena Mitasova, and Lubos Mitas. Multivariate Interpolation of Precipitation Using Regularized Spline with Tension. *Transactions in GIS*, 6(2) :135–150, 2002.

[125] M. F. HUTCHINSON. Interpolating mean rainfall using thin plate smoothing splines. *International Journal of Geographical Information Systems*, 9(4) :385–403, July 1995.

[126] Michael F Hutchinson. Interpolation of Rainfall Data with Thin Plate Smoothing Splines - Part II : Analysis of Topographic Dependence. *Journal of Geographic Information and Decision Analysis*, 2(2) :152–176, 1998.

[127] Eric P.J. Boer, Kirsten M de Beurs, and A Dewi Hartkamp. Kriging and thin plate splines for mapping climate variables. *International Journal of Applied Earth Observation and Geoinformation*, 3(2) :146–154, January 2001.

[128] Maria C. Mariani and Kanadpriya Basu. Spline interpolation techniques applied to the study of geophysical data. *Physica A : Statistical Mechanics and its Applications*, 428 :68–79, June 2015.

[129] HELENA MITASOVA. Modeling spatially and temporally distributed phenomena : new methods and tools for GRASS GIS. page 23, 1995.

[130] Jerome H. Friedman. Multivariate Adaptive Regression Splines. *The Annals of Statistics*, 19(1) :1–67, 1991.

[131] Jerome H Friedman and Charles B Roosen. An introduction to multivariate adaptive regression splines. *Statistical Methods in Medical Research*, 4(3) :197–217, September 1995.

[132] J. R. Leathwick, D. Rowe, J. Richardson, J. Elith, and T. Hastie. Using multivariate adaptive regression splines to predict the distributions of New Zealand's freshwater diadromous fish. *Freshwater Biology*, 50(12) :2034–2052, December 2005.

[133] J. R. Leathwick, J. Elith, and T. Hastie. Comparative performance of generalized additive models and multivariate adaptive regression splines for statistical modelling of species distributions. *Ecological Modelling*, 199(2) :188–196, November 2006.

[134] Jane Elith and John Leathwick. Predicting species distributions from museum and herbarium records using multiresponse models fitted with multivariate adaptive regression splines. *Diversity and Distributions*, 13(3) :265–275, May 2007.

[135] Tian-Shyug Lee and I-Fei Chen. A two-stage hybrid credit scoring model using artificial neural networks and multivariate adaptive regression splines. *Expert Systems with Applications*, 28(4) :743–752, May 2005.

[136] Tian-Shyug Lee, Chih-Chou Chiu, Yu-Chao Chou, and Chi-Jie Lu. Mining the customer credit using classification and regression tree and multivariate adaptive regression splines. *Computational Statistics & Data Analysis*, 50(4) :1113–1130, February 2006.

[137] M. Preston and W. T. Hewitt. Animation using NURBS. *Computer Graphics Forum*, 13(4) :229–241, 1994.

[138] Cameron J. Turner and Richard H. Crawford. Selecting an Appropriate Metamodel : The Case for NURBs Metamodels. pages 759–771, January 2005.

[139] Cameron J. Turner and Richard H. Crawford. N-Dimensional Nonuniform Rational B-Splines for Metamodeling. *Journal of Computing and Information Science in Engineering*, 9(3) :031002, 2009.

[140] Cameron J. Turner and Richard H. Crawford. Modeling Design Spaces With Discontinuous Variables Using NURBs HyPerModels. pages 61–72, January 2006.

[141] Cameron J. Turner, Richard H. Crawford, and Matthew I. Campbell. Multidimensional sequential sampling for NURBs-based metamodel development. *Engineering with Computers*, 23(3) :155–174, September 2007.

[142] Abiola M. Ajetunmobi, Cameron J. Turner, and Richard H. Crawford. Robust Optimization With NURBS HyPerModels. pages 317–327, January 2008.

[143] John Steuben and Cameron J. Turner. Robust Optimization Exploration Using NURBs-Based Metamodeling Techniques. pages 239–250, January 2010.

[144] K. Karhunen. Uber Lineare Methoden in der Wahrscheinlichkeitsrechnung. *Annals of Academic Science Fennicae*, 37 :3–79, 1946.

[145] D. Kosambi. Statistics in Function Space. In *Journal of Indian Mathematical Society*, volume 7, pages 76–88. Springer, New Delhi, 1943.

[146] M. Loeve. Fonctions Aléatoires du Second Ordre. *Processus stochastiques et mouvement Brownien*, 1948.

[147] M. A. Obukhov. Statistical description of continuous fields. *Transactions of the Geophysical International Academy Nauk USSR*, 24 :3–42, 1954.

[148] V. S. Pougachev. General theory of the correlations of random functions. *Izvestiya Akademii Nauk USSR*, 17 :1401–1402, 1953.

[149] J L Lumley. The structure of inhomogeneous turbulent flows. *Atmospheric Turbulence and Radio Wave Propagation*, pages 166–178, 1967.

[150] K. Kunisch and S. Volkwein. Galerkin Proper Orthogonal Decomposition Methods for a General Equation in Fluid Dynamics. *SIAM Journal on Numerical Analysis*, 40(2) :492–515, January 2002.

[151] C. W. Rowley. Model reduction for fluids, using balanced proper orthogonal decomposition. *International Journal of Bifurcation and Chaos*, 15(03) :997–1013, March 2005.

[152] Nadine Aubry, Philip Holmes, John L. Lumley, and Emily Stone. The dynamics of coherent structures in the wall region of a turbulent boundary layer. *Journal of Fluid Mechanics*, 192 :115–173, July 1988.

[153] Philip Holmes, John L. Lumley, Gahl Berkooz, and Clarence W. Rowley. *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*. Cambridge University Press, February 2012. Google-Books-ID : x93aZaFz5p4C.

[154] Bernd R. Noack, Konstantin Afanasiev, Marek Morzyński, Gilead Tadmor, and Frank Thiele. A hierarchy of low-dimensional models for the transient and post-transient cylinder wake. *Journal of Fluid Mechanics*, 497 :335–363, December 2003.

[155] Dietmar Rempfer and Hermann F. Fasel. Dynamics of three-dimensional coherent structures in a flat-plate boundary layer. *Journal of Fluid Mechanics*, 275 :257–283, September 1994.

[156] M. J. Mifsud, S. T. Shaw, and D. G. MacManus. A high-fidelity low-cost aerodynamic model using proper orthogonal decomposition. *International Journal for Numerical Methods in Fluids*, 63(4) :468–494, 2009.

[157] G Berkooz, P Holmes, and J L Lumley. The Proper Orthogonal Decomposition in the Analysis of Turbulent Flows. *Annual Review of Fluid Mechanics*, 25(1) :539–575, 1993.

[158] Michael Hinze and Stefan Volkwein. Proper Orthogonal Decomposition Surrogate Models for Nonlinear Dynamical Systems : Error Estimates and Suboptimal Control. pages 261–305, 2005.

[159] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition*. Elsevier, October 2013. Google-Books-ID : BIJZTGjTxBgC.

[160] Ernesto Arias, Hal Eden, Gerhard Fischer, Andrew Gorman, and Eric Scharff. Transcending the individual human mind—creating shared understanding through collaborative design. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1) :84–113, 2000.

[161] Hung V. Ly and Hien T. Tran. Modeling and control of physical processes using proper orthogonal decomposition. *Mathematical and Computer Modelling*, 33(1) :223–236, January 2001.

[162] Stanislav Y. Shvartsman and Ioannis G. Kevrekidis. Nonlinear model reduction for control of distributed systems : A computer-assisted study. *AIChE Journal*, 44(7) :1579–1595, July 1998.

[163] W. R. Graham, J. Peraire, and K. Y. Tang. Optimal control of vortex shedding using low-order models. Part I-open-loop model development. *International Journal for Numerical Methods in Engineering*, 44(7) :945–972, March 1999.

[164] Frédéric Da Silva. *Méthodologies de réduction de modèles multiphysiques pour la conception et la commande d'une chaîne de traction électrique*. PhD thesis, 2015.

[165] H. T. Banks, Michele L. Joyner, Buzz Wincheski, and William P. Winfree. Nondestructive evaluation using a reduced-order computational methodology. *Inverse Problems*, 16(4) :929, 2000.

[166] Salah U. Hamim and Raman P. Singh. Proper Orthogonal Decomposition–Radial Basis Function Surrogate Model-Based Inverse Analysis for Identifying Nonlinear Burgers Model Parameters From Nanoindentation Data. *Journal of Engineering Materials and Technology*, 139(4) :041010–041010–8, July 2017.

[167] Jin Hwan Ko, Kwangsub Jung, Seongseop Kim, Wonbae Kim, and Maenghyo Cho. A proper orthogonal decomposition for parametric study of the mechanical behavior of nanowires. *Journal of Mechanical Science and Technology*, 25(1) :157–162, January 2011.

[168] Gal Berkooz. Observations on the Proper Orthogonal Decomposition. In *Studies in Turbulence*, pages 229–247. Springer, New York, NY, 1992.

[169] Philip Holmes, John L. Lumley, and Gal Berkooz. *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*. Cambridge University Press, October 1996. Google-Books-ID : DAFiQgAACAAJ.

[170] Lawrence Sirovich. Turbulence and the dynamics of coherent structures. Part II : Symmetries and transformations. *Quarterly of Applied Mathematics*, 45(3) :573–582, October 1987.

[171] Lawrence Sirovich. Turbulence and the dynamics of coherent structures. Part I : Coherent structures. *Quarterly of Applied Mathematics*, 45(3) :561–571, October 1987.

[172] Y. C. Liang, H. P. Lee, S. P. Lim, W. Z. Lin, K. H. Lee, and C. G. Wu. PROPER ORTHOGONAL DECOMPOSITION AND ITS APPLICATIONS—PART I : THEORY. *Journal of Sound and Vibration*, 252(3) :527–544, May 2002.

[173] Pierre Ladeveze. The large time increment method for the analyze of structures with nonlinear constitutive relation described by internal variables. *C. R. Acad. Sci. Paris*, 309 :1095–1099, 1989.

[174] Ph Boisse, P. Bussy, and P. Ladeveze. A new approach in non-linear mechanics : The large time increment method. *International Journal for Numerical Methods in Engineering*, 29(3) :647–663, March 1990.

[175] J.-Y. Cognard and P. Ladeveze. The Large Time Increment Method Applied to Cyclic Loadings. In Michał Życzkowski, editor, *Creep in Structures*, International Union of Theoretical and Applied Mechanics, pages 555–562. Springer Berlin Heidelberg, 1991.

[176] Pierre-Alain Boucard, P. Ladevèze, M. Poss, and P. Rougée. A nonincremental approach for large displacement problems. *Computers & Structures*, 64(1-4) :499–508, July 1997.

[177] P. Ladevèze, J. C. Passieux, and D. Néron. The LATIN multiscale computational method and the Proper Generalized Decomposition. *Computer Methods in Applied Mechanics and Engineering*, 199(21–22) :1287–1296, April 2010.

[178] David Néron and Pierre Ladevèze. Proper Generalized Decomposition for Multiscale and Multiphysics Problems. *Archives of Computational Methods in Engineering*, 17(4) :351–372, December 2010.

[179] Pierre Ladeveze. *Nonlinear Computational Structural Mechanics : New Approaches and Non-Incremental Methods of Calculation*. Springer Science & Business Media, 1999. Google-Books-ID : STzaBwAAQBAJ.

[180] J.-Y. Cognard and P. Ladevèze. A large time increment approach for cyclic viscoplasticity. *International Journal of Plasticity*, 9(2) :141–157, January 1993.

[181] J.-Y. Cognard, P. Ladevèze, and P. Talbot. A large time increment approach for thermomechanical problems. *Advances in Engineering Software*, 30(9-11) :583–593, September 1999.

[182] P. Boisse, P. Ladeveze, M. Poss, and P. Rougee. A new large time increment algorithm for anisotropic plasticity. *International Journal of Plasticity*, 7(1-2) :65–77, January 1991.

[183] P. Ladeveze, J. Y. Cognard, and P. Talbot. A Non-incremental and Adaptive Computational Approach in Thermo-viscoplasticity. In O. T. Bruhns and E. Stein, editors, *IUTAM*

Symposium on Micro- and Macrostructural Aspects of Thermoplasticity, Solid Mechanics and its Applications, pages 281–291. Springer Netherlands, 2002.

[184] P. Ladevèze and U. Perego. Duality preserving discretization of the large time increment methods. *Computer Methods in Applied Mechanics and Engineering*, 189(1) :205–232, August 2000.

[185] B. Bognet, F. Bordeu, F. Chinesta, A. Leygue, and A. Poitou. Advanced simulation of models defined in plate geometries : 3d solutions with 2d computational complexity. *Computer Methods in Applied Mechanics and Engineering*, 201–204 :1–12, January 2012.

[186] H. Lamari, A. Ammar, P. Cartraud, G. Legrain, F. Chinesta, and F. Jacquemin. Routes for Efficient Computational Homogenization of Nonlinear Materials Using the Proper Generalized Decompositions. *Archives of Computational Methods in Engineering*, 17(4) :373–391, December 2010.

[187] A. Ammar, M. Normandin, and F. Chinesta. Solving parametric complex fluids models in rheometric flows. *Journal of Non-Newtonian Fluid Mechanics*, 165(23–24) :1588–1601, December 2010.

[188] Olivier Allix, Pierre Ladevèze, Daniel Gilletta, and Roger Ohayon. A damage prediction method for composite structures. *International Journal for Numerical Methods in Engineering*, 27(2) :271–283, September 1989.

[189] X. Aubard, C. Cluzel, L. Guitard, and Pierre Ladevèze. Damage modeling at two scales for 4d carbon/carbon composites. *Computers & Structures*, 78(1-3) :83–91, November 2000.

[190] Pierre Ladevèze, L. Guitard, Laurent Champaney, and X. Aubard. Debond modeling for multidirectional composites. *Computer Methods in Applied Mechanics and Engineering*, 185(2-4) :109–122, May 2000.

[191] P. Ladevèze. Multiscale modelling and computational strategies for composites. *International Journal for Numerical Methods in Engineering*, 60(1) :233–253, May 2004.

[192] David Violeau, Pierre Ladevèze, and Gilles Lubineau. Micromodel-based simulations for laminated composites. *Composites Science and Technology*, 69(9) :1364–1371, July 2009.

[193] Laurent Champaney, J. Y. Cognard, and Pierre Ladevèze. Modular analysis of assemblages of three-dimensional structures with unilateral contact conditions. *Computers & Structures*, 73(1-5) :249–266, October 1999.

[194] Pierre-Alain Boucard, M. Dérumaux, Pierre Ladeveze, and Ph. Roux. Macro-meso models for joint submitted to pyrotechnic shock. *Computational Fluid and Solid Mechanics 2003*, pages 139–142, January 2003.

[195] Pierre Ladevèze, H. Lemoussu, and Pierre-Alain Boucard. A modular approach to 3-D impact computation with frictional contact. *Computers & Structures*, 78(1-3) :45–51, November 2000.

[196] H. Lemoussu, Pierre-Alain Boucard, and Pierre Ladevèze. A 3d shock computational strategy for real assembly and shock attenuator. *Advances in Engineering Software*, 33(7-10) :517–526, July 2002.

[197] F. El Halabi, D. González, Agustí Chico-Roca, and M. Doblaré. Multiparametric response surface construction by means of proper generalized decomposition : An extension of the PARAFAC procedure. *Computer Methods in Applied Mechanics and Engineering*, 253 :543–557, January 2013.

[198] Christophe Heyberger, Pierre-Alain Boucard, and David Néron. Multiparametric analysis within the proper generalized decomposition framework. *Computational Mechanics*, 49(3) :277–289, March 2012.

[199] E. Pruliere, F. Chinesta, and A. Ammar. On the deterministic solution of multidimensional parametric models using the Proper Generalized Decomposition. *Mathematics and Computers in Simulation*, 81(4) :791–810, December 2010.

[200] Amine Ammar, Antonio Huerta, Francisco Chinesta, Elías Cueto, and Adrien Leygue. Parametric solutions involving geometry : A step towards efficient shape optimization. *Computer Methods in Applied Mechanics and Engineering*, 268 :178–193, January 2014.

[201] A. Leygue and E. Verron. A First Step Towards the Use of Proper General Decomposition Method for Structural Optimization. *Archives of Computational Methods in Engineering*, 17(4) :465–472, December 2010.

[202] Francisco Chinesta, Amine Ammar, and Elías Cueto. Recent Advances and New Challenges in the Use of the Proper Generalized Decomposition for Solving Multidimensional Models. *Archives of Computational Methods in Engineering*, 17(4) :327–350, December 2010.

[203] F. Chinesta, A. Leygue, F. Bordeu, J. V. Aguado, E. Cueto, D. Gonzalez, I. Alfaro, A. Ammar, and A. Huerta. PGD-Based Computational Vademecum for Efficient Design, Optimization and Control. *Archives of Computational Methods in Engineering*, 20(1) :31–59, March 2013.

[204] Francisco Chinesta and Elías Cueto. *PGD-Based Modeling of Materials, Structures and Processes*. Springer Science & Business, April 2014.

[205] Gerald Farin. *Curves and Surfaces for CAGD*. Elsevier, 2002.

[206] The Mathworks Inc. External Interfaces, March 2009.

[207] Giulio Costa, Marco Montemurro, and Jérôme Pailhès. A General Hybrid Optimization Strategy for Curve Fitting in the Non-uniform Rational Basis Spline Framework. *Journal of Optimization Theory and Applications*, pages 1–27, November 2017.

[208] Giulio Costa. *Design and Optimisation Methods for Structures produced by means of additive Manufacturing*. PhD thesis, Arts et Métiers ParisTech, France, 2018.

[209] Giulio Costa, Marco Montemurro, and Jérôme Pailhès. A 2d topology optimisation algorithm in NURBS framework with geometric constraints. *International Journal of Mechanics and Materials in Design*, pages 1–28, November 2017.

[210] Giulio Costa, Marco Montemurro, and Jérôme Pailhès. Hypersurfaces for 3d Topology Optimisation Problems. *International Journal of Solids and Structures (submitted)*, 2018.

[211] Marco Montemurro and Anita Catapano. A general B-Spline surfaces theoretical framework for optimisation of variable angle-tow laminates. *Composite Structures*, October 2018.

[212] Marco Montemurro. *A contribution to the development of design strategies for the optimisation of lightweight structures.* Habilitation à diriger des recherches, Université de Bordeaux, France, 2018.

[213] Jorge Nocedal and Stephen J. Wright. *Numerical optimization.* Springer series in operations research. Springer, New York, NY, corr. 2. print edition, 2000. OCLC : 247330880.

[214] Optimization Toolbox User's Guide. Technical report, The Mathworks Inc., 3 Apple Hill Drive, Natick, 2018.

[215] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system : optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1) :29–41, February 1996.

[216] James Kennedy. Particle Swarm Optimization. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning*, pages 760–766. Springer US, Boston, MA, 2010.

[217] Leandro Nunes Castro and Jonathan Timmis. *Artificial Immune Systems : A New Computational Intelligence Approach.* Springer Science & Business Media, September 2002. Google-Books-ID : aMFP7p8DtaQC.

[218] John Henry Holland. *Adaptation in Natural and Artificial Systems : An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* MIT Press, 1992. Google-Books-ID : 5EgGaBkwvWcC.

[219] David E. Goldberg and John H. Holland. Genetic Algorithms and Machine Learning. *Machine Learning*, 3(2) :95–99, October 1988.

[220] Michel Gendreau and Jean-Yves Potvin, editors. *Handbook of metaheuristics.* Number 146 in International series in operations research & management science. Springer, New York, NY, 2. ed edition, 2010. OCLC : 729853221.

[221] Charles Darwin. *On the Origin of Species, 1859.* Routledge, 1859.

[222] Marco Montemurro, Angela Vincenti, and Paolo Vannucci. The Automatic Dynamic Penalisation method (ADP) for handling constraints with genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 256 :70–87, April 2013.

[223] Olvi L. Mangasarian. *Nonlinear Programming.* Society for Industrial and Applied Mathematics, 1994. Google-Books-ID : VdMcrxUfeMUC.

[224] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, June 2013. Google-Books-ID : _WuAvIx0EE4C.

[225] Delint Ira Setyo Adi, Siti Mariyam bt Shamsuddin, and Siti Zaiton Mohd Hashim. NURBS Curve Approximation Using Particle Swarm Optimization. In *2010 Seventh International Conference on Computer Graphics, Imaging and Visualization*, pages 73–79, Sydney, TBD, Australia, August 2010. IEEE.

[226] W. Ma and J. P. Kruth. NURBS curve and surface fitting for reverse engineering. *The International Journal of Advanced Manufacturing Technology*, 14(12) :918–927, December 1998.

[227] M Montemurro, P Vannucci, and A Vincenti. BIANCA, A Genetic Algorithm for Engineering Optimisation. page 87.

[228] The Mathworks Inc. Getting Started with MATLAB. page 187, 2005.

[229] Anita Catapano and Marco Montemurro. A multi-scale approach for the optimum design of sandwich plates with honeycomb core. Part I : homogenisation of core properties. *Composite Structures*, 118 :664–676, December 2014.

[230] Anita Catapano and Marco Montemurro. A multi-scale approach for the optimum design of sandwich plates with honeycomb core. Part II : the optimisation strategy. *Composite Structures*, 118 :677–690, December 2014.

[231] G. Verchery. Les Invariants des Tenseurs d'Ordre 4 du Type de l'Élasticité. In Jean-Paul Boehler, editor, *Mechanical Behavior of Anisotropic Solids / Comportment Méchanique des Solides Anisotropes*, pages 93–104. Springer Netherlands, Dordrecht, 1982.

[232] Marco Montemurro. The polar analysis of the Third-order Shear Deformation Theory of laminates. *Composite Structures*, 131 :775–789, November 2015.

[233] Marco Montemurro. An extension of the polar method to the First-order Shear Deformation Theory of laminates. *Composite Structures*, 127 :328–339, September 2015.

[234] Marco Montemurro. Corrigendum to "An extension of the polar method to the First-order Shear Deformation Theory of laminates" [Compos. Struct. 127 (2015) 328–339]. *Composite Structures*, 131 :1143–1144, November 2015.

[235] J. N. Reddy. *Mechanics of Laminated Composite Plates and Shells : Theory and Analysis, Second Edition*. CRC Press, June 2004. Google-Books-ID : eeUr_AJiGRcC.

[236] Paolo Vannucci. Plane Anisotropy by the Polar Method*. *Meccanica*, 40(4) :437–454, December 2005.

[237] Paolo Vannucci. A Note on the Elastic and Geometric Bounds for Composite Laminates. *Journal of Elasticity*, 112(2) :199–215, July 2013.

[238] P. Vannucci and G. Verchery. A special class of uncoupled and quasi-homogeneous laminates. *Composites Science and Technology*, 61(10) :1465–1473, August 2001.

[239] Torquato Garulli, Anita Catapano, Marco Montemurro, Julien Jumel, and Daniele Fanteria. Quasi-trivial stacking sequences for the design of thick laminates. *Composite Structures*, 200 :614–623, September 2018.

[240] Inc. ANSYS. Mechanical APDL Modeling and Meshing Guide. *275 Technology Drive Canonsburg, PA 15317*, November 2013.

Développement d'une nouvelle méthode de réduction de modèle basée sur les hypersurfaces NURBS (Non-Uniform Rational B-Splines)

RESUME :

Malgré des décennies d'incontestables progrès dans le domaine des sciences informatiques, un certain nombre de problèmes restent difficiles à traiter en raison, soit de leur complexité numérique (problème d'optimisation, …), soit de contraintes spécifiques telle que la nécessité de traitement en temps réel (réalité virtuelle, augmentée, …). Dans ce contexte, il existe des méthodes de réduction de modèle qui permettent de réduire les temps de calcul de simulations multi-champs et/ou multi-échelles complexes. Le processus de réduction de modèle consiste à paramétrer un métamodèle qui requiert moins de *ressources* pour être évalué que le modèle complexe duquel il a été obtenu, tout en garantissant une certaine précision. Les méthodes actuelles nécessitent, en général, soit une expertise de l'utilisateur, soit un grand nombre de choix arbitraires de sa part. De plus, elles sont bien souvent adaptées à une application spécifique mais difficilement transposable à d'autres domaines. L'objectif de notre approche est donc d'obtenir, s'il n'est pas le meilleur, un *bon* métamodèle quel que soit le problème considéré. La stratégie développée s'appuie sur l'utilisation des hypersurfaces NURBS et se démarque des approches existantes par l'absence d'hypothèses simplificatrices sur les paramètres de celles-ci. Pour ce faire, une métaheuristique (de type algorithme génétique), capable de traiter des problèmes d'optimisation dont le nombre de variables n'est pas constant, permet de déterminer automatiquement l'ensemble des paramètres de l'hypersurface sans transférer la complexité des choix à l'utilisateur.

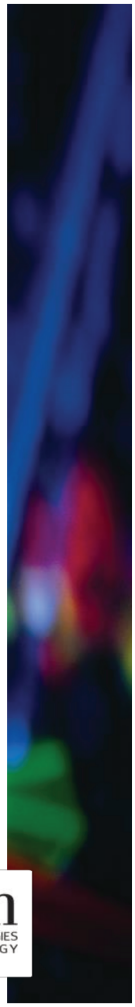
Mots clés : Algorithme génétique ; Métamodèle ; Modèle réduit ; Réduction de modèle ; Optimisation ; Conception

Development of a new metamodeling method based on NURBS (Non-Uniform Rational B-Splines) hypersurfaces

ABSTRACT:

Despite undeniable progress achieved in computer sciences over the last decades, some problems remain intractable either by their numerical complexity (optimisation problems, …) or because they are subject to specific constraints such as real-time processing (virtual and augmented reality, …). In this context, metamodeling techniques can minimise the computational effort to realize complex multi-field and/or multi-scale simulations. The metamodeling process consists of setting up a metamodel that needs less *resources* to be evaluated than the complex one that is extracted from by guaranteeing, meanwhile, a minimal accuracy. Current methods generally require either the user's expertise or arbitrary choices. Moreover, they are often tailored for a specific application, but they can be hardly transposed to other fields. Thus, even if it is not the best, our approach aims at obtaining a metamodel that remains a *good* one for whatever problem at hand. The developed strategy relies on NURBS hypersurfaces and stands out from existing ones by avoiding the use of empiric criteria to set its parameters. To do so, a metaheuristic (a genetic algorithm) able to deal with optimisation problems defined over a variable number of optimisation variables sets automatically all the hypersurface parameters so that the complexity is not transferred to the user.

Keywords : Genetic algorithm; Metamodel; Surrogate model; Metamodeling; Optimization; Design



ARTS ET MÉTIERS ParisTech

ParisTech INSTITUT DES SCIENCES ET TECHNOLOGIES PARIS INSTITUTE OF TECHNOLOGY