Méthode EF2 et hyperréduction de modèle : vers des calculs massifs à l'échelle micro

Georges Peyre

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

MINES ParisTech

École doctorale n° 432 : SMI – Sciences et Métiers de l'Ingénieur

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

l'École nationale supérieure des mines de Paris

Spécialité " Mécanique "

présentée et soutenue publiquement par

Georges PEYRE

le 22 septembre 2015

Méthode EF2 et hyperréduction de modèle : vers des calculs massifs à l'échelle micro

Directeur de thèse : **Frédéric FEYEL**

Jury

M. Francisco CHINESTA, Professeur, Ecole Centrale Nantes	Rapporteur
M. Dominique EYHERAMENDY, Professeur, Ecole Centrale Marseille	Rapporteur
M. Christian REY, Professeur, SafranTech	Président
M. Felix FRITZEN, Group Leader, Université de Stuttgart	Examinateur
M. Vincent CHIARUTTINI, Chef d'équipe, Onera	Examinateur
M. Frédéric FEYEL, Chef du pôle M. & S., SafranTech	Examinateur

T H È S E

MINES ParisTech
Centre des Matériaux (CdM)
UMR CNRS 7633, BP 87, F-91003 Evry Cedex, France

Aux passionnés d'hyperréduction de modèle.

Remerciements

Je remercie en premier lieu Frédéric Feyel, mon directeur de thèse. Je lui dois la réalisation de ce projet.

Je remercie Francisco Chinesta pour avoir accepté de rapporter ce manuscrit. Son entrain et son habileté à la présentation en font un modèle pour moi et une source de fascination.

Je remercie de même Dominique Eyheramendy pour la rapporter. Mes réflexions sur l'organisation du code de l'hyperréduction m'ont ouvert à ses travaux dont je savoure maintenant toute la finesse.

Je remercie Felix Fritzen, dont le travail m'impressionne beaucoup, de m'avoir invité à Karlsruhe à participer à un workshop sur les méthodes de réduction de modèle. J'ai appris récemment qu'il semble avoir fourni une réponse plus satisfaisante au sujet de ma thèse que celle que l'on trouvera ici. J'ai le coeur rempli d'espoir de voir que l'on peut progresser réellement sur un sujet aussi difficile que celui-ci.

Je remercie Vincent Chiaruttini de sa bienveillance. Son soutien de longue haleine s'est révélé absolument nécessaire à l'avancée et à l'achèvement de ce doctorat. Ce soutien aurait été inutile sans ses grandes qualités humaines.

Je remercie Serge Kruch qui m'a frappé par la discrétion et la pertinence de ses remarques. J'ai rarement vu quelqu'un d'aussi juste. Sa constance m'a permis de mener à bien ce projet.

Je remercie Johann Rannou de son regard extérieur et de son aide ciblée et efficace notamment pour une présentation que je ne parvenais pas à améliorer. Je ne pensais pas que l'on pouvait être si extérieur à un sujet et si pertinent sur les améliorations de forme et de fond à apporter.

Je remercie Samuel Forest d'avoir diverti quelques heures de son emploi du temps pour s'enquérir, dans un mouvement de curiosité que je pensais impossible, de l'avancement de mes travaux et m'expliquer ce qui le motivait dans la recherche.

Je remercie Arjen Roos de m'avoir fait découvrir les ballades de Brahms dans un moment où je ne parvenais plus à avancer. Son humanité et sa finesse d'analyse ont été déterminantes.

Je remercie Teddy Chantrait des discussions que j'ai pu mener avec lui et de sa sympathie. Grâce à ces échanges, j'ai pu exprimer plus clairement et simplement certaines de mes idées. Il a même accepté de relire aussi mon manuscrit.

Je remercie Amélie Morançais pour le sérieux, la tranquillité, la persévérance qu'elle affichait et l'écoute dont elle a fait preuve au cours de ces années. Ces quatre éléments ont constitué pour moi un repère.

Je remercie Jérôme Crépin pour la discution décisive, pleine d'humanité et éclairante que j'ai menée avec lui en début de deuxième année de thèse.

Je remercie les chercheurs du DMSM/MNU et DMSM/MCE d'avoir maintenu un environnement où il fait très bon de travailler. C'est une grande force et c'est très rare.

Je remercie mes parents pour avoir relu en détail ce manuscrit. Toutes les imperfections me sont imputables.

Je remercie mes camarades des Mines, en particulier ceux de l'option Matériaux, partis en thèse, avec qui j'ai gardé de nombreux liens. Ils m'ont aidé à mesurer ce qu'était un projet de thèse.

Je ne vois pas comment j'aurais pu réussir sans le concours, la confiance et l'appui de ces chercheurs, doctorants et amis.

Table des matières

Introduction

Au sein des bureaux d'étude des industries aéronautiques comme ceux de Snecma ou de Turboméca, pour optimiser la géométrie des structures ou pour optimiser l'utilisation des matériaux, ou encore, pour mener des études paramétriques, les ingénieurs ont recours de façon systématique à un grand nombre de calculs éléments finis pourtant assez similaires. Aujourd'hui, ils ne tirent pas partie de cette similarité pour en diminuer le coût. Plus particulièrement, parmi les méthodes d'étude paramétrique couramment utilisées, l'absence de méthode générique applicable dans un cadre industriel pour prendre en compte l'effet de la microstructure de manière rapide et efficace dans la réponse macroscopique d'une structure se fait sentir. En effet, pouvoir mener des calculs multiéchelle de grande dimension, autrement dit posséder des méthodes d'homogénéisation numériques efficaces dans un cadre indutriel, épargnerait l'effort de développer un modèle mécanique pour chaque microstructure particulière.

Le travail pionnier de [28] a permis l'avènement de ces méthodes d'homogénéisation numériques génériques. C'est alors que les premiers jalons de la EF^2 ont été établis et qu'une implémentation a été proposée. Mais ces méthodes, très coûteuses, restent en pratique inapplicables au calcul de structures réelles. Dans les années 90, [29] a réalisé la parallélisation des méthodes EF^2 dans l'espoir de diminuer ce coût de calcul prohibitif. Mais en dépit de l'utilisation d'architectures massivement parallèles, en dépit de l'augmentation graduelle de la puissance du matériel informatique, l'accélération obtenue demeure encore insuffisante d'un facteur 100 à 1000.

Par ailleurs, dans le domaine de la mécanique des fluides, afin de modéliser les écoulements tourbillonaires, des méthodes de réduction de modèle ont émergé au cours des années 80 [61]. Mais aucune application systématique de ces méthodes n'a été menée dans le champ du calcul de la mécanique des structures. Ces méthodes de réduction de modèle ont été perfectionnées dans les travaux de [20] et [8] qui les ont appliquées à des cas de calcul de structure linéaire. La précision du résultat obtenu n'est pas contrôlée. [54] a donné un estimateur rigoureux a posteriori permettant de quantifier l'erreur entre le modèle réduit et le modèle complet. Cependant, les applications concernées se limitent aux cas de fonctionnelles d'énergie affines par rapport aux paramètres. Et même si, par la suite, plusieurs extensions à des fonctionnelles d'énergie plus générales ont été proposées par ce même auteur [57], l'application à des structures mécaniques non-linéaires, i. e. possédant une loi matériau standard, n'est pas abordée. [58] développe une méthode générique d'hyperréduction adaptative permettant de traiter certains types de non-linéarités mécaniques et propose un indicateur d'erreur. Néanmoins, les facteurs d'accélération ne sont pas substantiels. En 2012, il améliore la performance de la méthode d'hyperréduction tout en préservant sa généricité : il appelle cette méthode l'hyperréduction multidimensionnelle [60]. Il l'utilise pour mener des études paramétriques hyperréduites en parallèle. Cependant il ne l'applique pas au cas EF^2, là où les gains en temps de calcul promettent d'être colossaux.

En somme, d'une part, les méthodes EF^2 offrent la généricité nécessaire pour s'affranchir du particularisme de la microstructure mais leur coût ruine toute velléité d'intérêt industriel. Et partant, elles n'ont jamais été testées sur des cas industriels réels. D'autre part, les méthodes de réduction permettent aujourd'hui de tenir compte à faible coût des non-linéarités des

matériaux standard qui interviennent dans les calculs EF. Toutefois, la réduction de modèle n'a jamais été appliquée aux EF^2.

C'est ce qui justifie cette étude. Elle consiste à développer une méthode de calcul éléments finis générique issue du couplage de la méthode d'hyperréduction et des approches EF^2 en vue de pouvoir ainsi enfin prendre en compte les effets microstructuraux sur des cas industriels réels.

La démarche consiste dans un premier temps à formuler et valider théoriquement une expression de l'hyperréduction en tenant compte des conditions aux limites (Dirichlet, Neumann, périodiques) afin de pouvoir transposer la formulation classique d'un problème de mécanique des milieux continus (MMC), traité aujourd'hui par n'importe quel code éléments finis, à un problème d'hyperréduction. Un triple objectif anime cette étude. On souhaite simplifier la théorie classique de l'hyperréduction, simplifier l'implémentation et tenir compte des conditions aux limites. Pour cela, on revient aux fondements de la formulation des problèmes MMC, en particulier la formulation des conditions aux limites. Puis on transpose les conditions aux limites MMC au problème de l'hyperréduction. On débouche alors sur une généralisation du modèle classique de l'hyperréduction.

Dans un deuxième temps, un code d'hyperréduction orienté objet, documenté, commenté, efficace et applicable pour des conditions aux limites de type Dirichlet, Neumann et périodique est réalisé à partir de la formulation mécaniste de l'hyperréduction. Le double objectif est d'offrir une architecture logicielle robuste et stable permettant d'intégrer de futurs développements de la méthode de réduction de modèle - facilité d'extension - et de rendre possible son utilisation dans un cadre industriel - facilité d'utilisation -. Pour cela, on recense les schémas de programmation et les concepts-objets classiquement mis en oeuvre dans l'élaboration des codes EF commerciaux. Puis on utilise en les adaptant ces schémas de programmation et ces concepts-objets pour développer le code d'hyperréduction.

Dans un troisième temps, afin d'illustrer la versatilité de l'outil informatique mis en place, on procède à la validation du code de calcul de réduction sur des cas de différentes natures. Pour cela, une validation est effectuée en sollicitant des structures homogènes par des conditions aux limites classiques (Dirichlet et Neumann) en choisissant une loi matériau non-linéaire standard. Un second volet consiste à l'appliquer à un volume élémentaire représentatif biphasique dont une phase, la fibre, est élastique linéaire, et l'autre, la matrice, comporte une loi d'écrouissage non-linéaire. Le volume élémentaire est sollicité en condition périodique. Ce calcul sur VER représente une étape de validation cruciale pour pouvoir effectuer un calcul EF^2 complet.

Enfin une procédure de couplage entre l'hyperréduction et les EF^2 est proposée et implémentée afin d'évaluer la diminution du coût de calcul. Les performances de l'algorithme sont analysées pour déterminer des pistes permettant d'améliorer l'efficacité du couplage.

Partie I

Etat de l'art et outil logiciel

1

Méthodes de réduction de modèle

L'objectif de ce chapitre est de situer l'étude qui va suivre en présentant quelques méthodes de réduction de modèle. Les méthodes de réduction de modèle sont utilisées pour connaître la réponse $\boldsymbol{s}$ d'un système à une sollicitation donnée, que ce système soit par exemple un système dynamique $\frac{d\boldsymbol{u}}{dt} = f(\boldsymbol{u}, t)$ avec des conditions initiales $\boldsymbol{e}$ ou une équation aux dérivées partielles, par exemple l'équilibre mécanique $\operatorname{div} \boldsymbol{\sigma} = 0$ *sous conditions aux limites appropriées $\boldsymbol{e}$. La réponse ou grandeur d'intérêt peut être par exemple l'état du système lui-même $\boldsymbol{u}$ ou tout autre quantité calculé à partir de cet état $\boldsymbol{u}$. Trouver $\boldsymbol{s}$ à partir des données d'entrées $\boldsymbol{e}$ peut se révéler très coûteux si le modèle comporte un très grand nombre de variables $\boldsymbol{x}$ ou de paramètres $\boldsymbol{\mu}$. Construire un modèle réduit, c'est donner une fonctionnelle permettant de déterminer à faible coût la sortie $\boldsymbol{s}$ étant donné un jeu de paramètres $\boldsymbol{\mu}$ et des entrées $\boldsymbol{e}$. Dans ce chapitre nous présenterons quelques méthodes de réduction de modèle en insistant en particulier sur les étapes clés de la construction d'un modèle réduit que sont le choix des snapshots, la méthode de construction de la base réduite, la méthode d'élaboration du modèle réduit ainsi que l'évaluation de l'erreur commise entre le modèle réduit et le modèle réel. Nous présenterons le type de système auquel nous avons appliqué la méthode. Il s'agit de problème de type EF^2. Nous tenterons de justifier le choix de la méthode de réduction retenue, i. e. la méthode d'hyperréduction, pour la suite de la thèse par rapport au corpus bibliographique et au type de problème à traiter.*

Sommaire

1.1 Présentation générale

Les modèles réduits sont utilisés pour accélérer certains types de calcul. Ainsi qu'illustré sur la figure 1.1, on distingue essentiellement deux étapes clés : une phase d'élaboration du modèle réduit en bleu, très coûteuse en temps de calcul, appelée phase *offline* et une phase de calcul *online* d'exploitation du modèle réduit en rouge, très peu coûteuse. Lors de la phase de calcul *offline*, il est toujours nécessaire de réaliser des calculs éléments finis pour produire un *snapshot*, i. e. un état mécanique du système, sauf si l'on utilise la méthode PGD (Proper Generalized Decomposition) en mode solveur qui permet de construire progressivement le *snapshot* en résovant des problèmes de faible dimension. C'est au niveau de la phase *online* qu'est réalisé le gain. Enfin des méthodes permettant d'estimer l'erreur commise entre le modèle réduit et le modèle réel représentée en gris sur la figure 1.1 peuvent être mises en place soit au niveau de la phase *online* soit au niveau de la phase *offline*.

Parmi les méthodes de la phase *online*, on distingue les méthodes de réduction de modèle, l'hyperréduction, la méthode GNAT, la NTFA. Ces méthodes exploitent des modèles réduits construits à l'aide d'une *POD* lors d'une étape *offline*. La *Gappy POD*, la *Goal Oriented POD* et la *Multidimensionnal POD* constituent des améliorations à la *POD* dont elles font usage. La *POD* permet de compresser un ensemble de *snapshots*. La construction de ces *snapshots* lors de la phase *offline* s'avère coûteuse en ressource et en temps car des calculs éléments finis complets sont nécessaires. Des alternatives comme la *PGD* ont été proposées. La *PGD* utilise les techniques développées dans le champ des méthodes tensorielles.

Afin d'estimer l'erreur entre le modèle réduit et le modèle réel, différents types d'estimateur d'erreur ont été développés. Ces estimateurs peuvent être utilisés lors de la phase *online*. A ce moment, l'estimation doit être faite à faible coût. Ils peuvent être utilisés également lors de la phase *offline* pour déterminer automatiquement quels *snapshots* calculer en vue de construire un modèle réduit. En l'absence d'un estimateur d'erreur pilotant la construction du modèle réduit, le choix des *snapshots* est réalisé en faisant appel au sens mécanique de l'utilisateur qui donne une justification *a posteriori* de la pertinence de son modèle.

Ce chapitre est une revue de l'ensemble de ces méthodes.

1.2 Notations

Les méthodes de réduction de modèle ont envahi le domaine de la mécanique des matériaux ces dernières années. Nous précisons quelques notations dans le tableau 1.1.

$\Omega = \text{Vect}\left\{\boldsymbol{N}_{1<\text{i},\text{j}<\text{N}}\right\}$ où $\boldsymbol{N}$ sont les fonctions de forme et $\hat{\Omega} = \text{Vect}\left\{\boldsymbol{\Psi}_{1<\text{i}<\text{n}}\right\}$.

On a les inclusions suivantes : $\hat{\Omega} \subset \Omega$ et $\tilde{\Omega} \subset \Omega$. Nous désignons également par $\hat{\boldsymbol{P}}$ la projection orthogonale pour la norme usuelle de Ω dans $\hat{\Omega}$ et suivant la même logique $\tilde{\boldsymbol{P}}$ la projection de Ω dans $\tilde{\Omega}$.

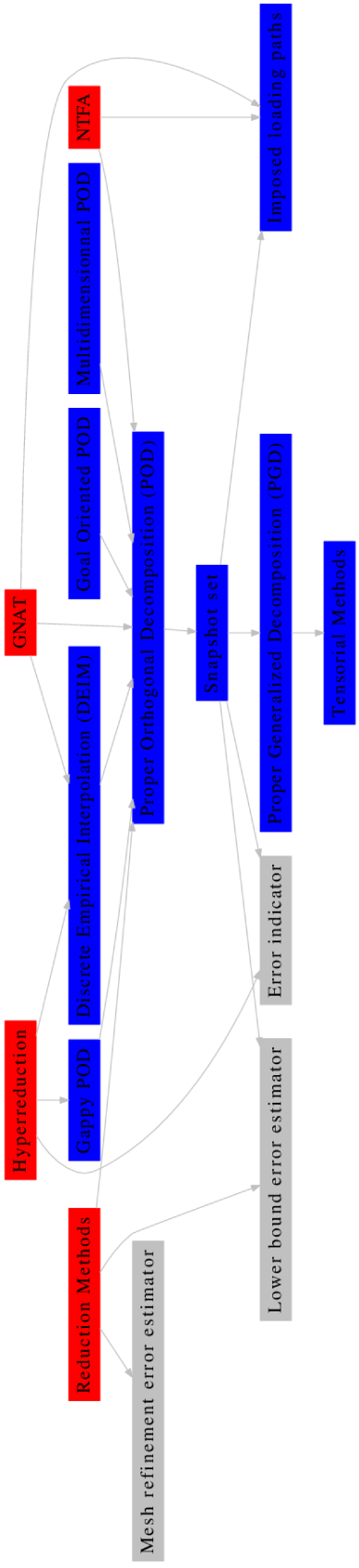


Figure 1.1 – Aperçu des méthodes de réduction de modèles.

Ω	Espace des degrés de liberté éléments finis
$\hat{\Omega}$	Espace réduit
$\tilde{\Omega}$	Espace d'échantillonage aussi appelé domaine réduit
$\boldsymbol{\Psi}$	Base réduite
$\boldsymbol{u}$	Champ
$\boldsymbol{\sigma}$	Champ de contrainte
$\hat{\boldsymbol{u}}$	Champ reconstruit
$\tilde{\boldsymbol{u}}$	Champ projeté sur l'espace d'échantillonage $\tilde{\Omega}$

Table 1.1 – Notations usuelles

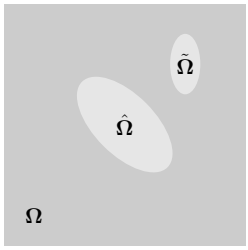


Figure 1.2 – Imbrication des espaces réduits

1.3 La POD (Proper Othrogonal Decomposition)

Définition On trouve une étude théorique détaillée de la méthode POD dans [66]. Il s'agit d'une méthode de compression qui permet d'approximer un espace généré par une famille $\boldsymbol{u}_1, ..., \boldsymbol{u}_m$ appelé espace des *snapshots* par un autre espace vectoriel de dimension inféreure et généré par $\boldsymbol{\phi}_1, ..., \boldsymbol{\phi}_n$ appelé espace réduit en ne conservant que les modes les plus énergétiques. Plus précisément, les modes $\boldsymbol{\phi}_1, ..., \boldsymbol{\phi}_n$ sont solutions du problème de minimisation sous contrainte associé au lagrangien :

$$\mathcal{G}(\boldsymbol{\phi}_1, ..., \boldsymbol{\phi}_n, \boldsymbol{\beta}) = \sum_{i=1}^{n}\sum_{j=1}^{m} |\boldsymbol{u}_j \cdot \boldsymbol{\phi}_i| + \frac{\boldsymbol{\beta}}{2}\sum_{j=1}^{m}(1 - {}^T\boldsymbol{\phi}_j\boldsymbol{\phi}_j)^2 + \frac{\boldsymbol{\beta}}{2}\sum_{\substack{i,j=1 \\ i\neq j}} ({}^T\boldsymbol{\phi}_i\boldsymbol{\phi}_j)^2 \tag{1.1}$$

En pratique, pour résoudre 1.1, on effectue une *SVD* (*Singular Value Decomposition*) sur la matrice des *snapshots* $\boldsymbol{U}$: $\boldsymbol{U} = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{W}^T$ en ne sélectionnant que les vecteurs propres de $\boldsymbol{U}$ correspondant aux plus grandes valeurs singulières. On choisit un certain nombre de valeurs singulières à conserver selon un critère : $E = \frac{\sum_{i=1}^{l}\lambda_i}{\sum_{i=1}^{n}\lambda_i} > E_{error}$.

Erreur commise avec la POD Quand on applique la POD à des systèmes dynamiques pour déterminer un système dynamique réduit, il convient d'apporter quelques modifications. Sur la figure 1.3, on présente l'écart entre le modèle réduit et complet discrétisé. Dans la discrétisation entrent en jeu les paramètres $\boldsymbol{\alpha}$ obtenus par un schéma d'intégration temporel quelconque :

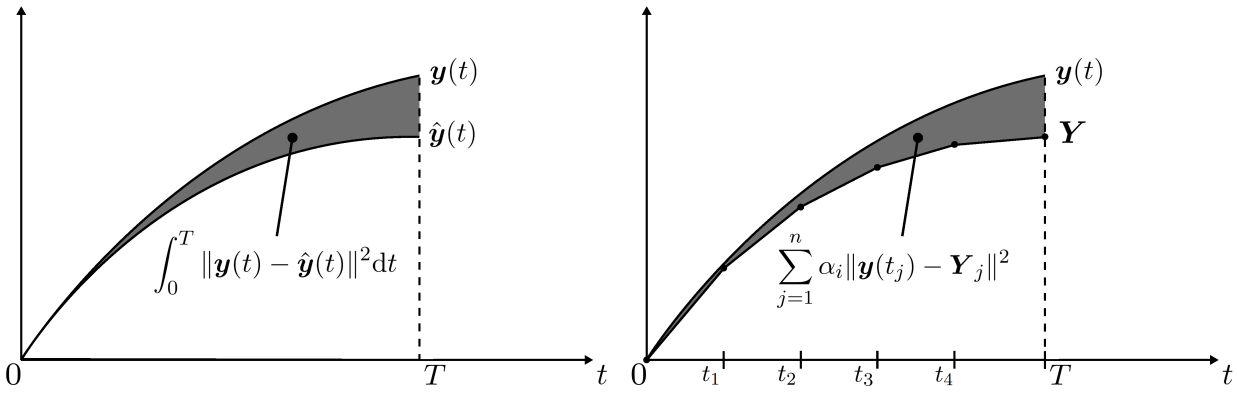


Figure 1.3 – Ecart entre le modèle complet et réduit (à gauche). Ecart entre le modèle complet discrétisé et le modèle réduit discrétisé (à droite)

$$\sum_{j=1}^{n} \boldsymbol{\alpha}_i \|\boldsymbol{u}(t_j)\|^2 \to \int_0^T \|\boldsymbol{u}(t)\|^2 \mathrm{d}t, \text{ quand } \Delta t \to 0 \tag{1.2}$$

si bien que l'approximation à effectuer n'est pas tellement celle de 1.1 mais plutôt :

$$\min_{\boldsymbol{u}_1,\ldots,\boldsymbol{u}_n} \sum_{j=1}^{m} \boldsymbol{\alpha}_j \|\boldsymbol{u}_j - \sum_{i=1}^{n} (\boldsymbol{u}_j \cdot \boldsymbol{\phi}_i)\boldsymbol{\phi}_i\|^2 \tag{1.3}$$

sous la condition :

$$\boldsymbol{\phi}_i \cdot \boldsymbol{\phi}_j = \delta_{ij} \tag{1.4}$$

La pondération de la POD par les coefficients $\boldsymbol{\alpha}_i$ permet de prendre en compte convenablement la discrétisation temporelle. Le choix de la norme permet de prendre en compte les phénomènes de discrétisation spatiale. L'article [66] illustre ces deux aspects. Souvent les POD sont appliquées brutalement aux calculs issus des champs éléments finis alors qu'il serait pertinent de faire entrer le paramètre de maille h dans l'élaboration de la POD pour s'affranchir des effets de maillage. Nous n'avons pas poussé dans ce travail l'étude plus avant. Mais il serait intéressant de considérer genre de problématique.

De plus, dans [66] les auteurs fournissent, pour une certaine classe d'équations non-linéaires, une estimation de l'erreur entre le modèle réduit et le modèle complet :

$$\int_0^T \|\boldsymbol{u} - \hat{\boldsymbol{u}}\|^2 \,\mathrm{dt} \leq \mathrm{C}(\mathrm{E}_{\boldsymbol{u}} - \mathrm{E}_{\boldsymbol{F}}) \tag{1.5}$$

$$E_{\boldsymbol{u}} = \int_0^T \|\boldsymbol{u} - \hat{\boldsymbol{P}}_{\boldsymbol{u}}\boldsymbol{u}\|^2 \,\mathrm{dt} \tag{1.6}$$

$$E_{\boldsymbol{F}} = \int_0^T \|\boldsymbol{F} - \hat{\boldsymbol{P}}_{\boldsymbol{F}}\boldsymbol{F}\|^2 \,\mathrm{dt} \tag{1.7}$$

où $\hat{\boldsymbol{u}}$ est la solution de l'équation différentielle à laquelle on a appliqué une projection de POD-Galerkin, $\hat{\boldsymbol{P}}$ et $\boldsymbol{u}$ est la solution du problème complet. $E_{\boldsymbol{F}}$ et $E_{\boldsymbol{u}}$ représentent alors

l'écart à l'espace réduit. Ce résultat n'est pas trivial et il est important pour comprendre de noter que $\hat{\boldsymbol{u}} \neq \hat{\boldsymbol{P}}_{\boldsymbol{u}}\boldsymbol{u}$.

POD multidimensionnelle [67] s'intéresse à la création d'un modèle réduit représentatif des états microstructuraux de VER dans le cas d'un procédé de forgeage. A l'échelle macroscopique, la surface d'un palet est déformée aléatoirement puis compressée. A l'échelle microscopique, les VER, qui présentent la même microstructure dans l'état initial, se déforment ; les grains dont ils sont constitués glissent et changent d'orientation. Le champ de déplacement $\boldsymbol{u}$ est décomposé par une méthode dite de *POD bi-orthogonale* : $\boldsymbol{u}(\boldsymbol{x},\boldsymbol{s},\boldsymbol{\omega}) = \overline{\boldsymbol{u}(\boldsymbol{x},\boldsymbol{s})} + \sum_{i=1}^{d}\sqrt{\rho_i}\boldsymbol{\Psi}_i(\boldsymbol{s})\boldsymbol{\Phi}_i(\boldsymbol{x},\boldsymbol{\omega})$. Une telle décomposition permet de séparer les paramètres microscopiques $\boldsymbol{s}$ de la position spatiale macroscopique $\boldsymbol{x}$ et de $\boldsymbol{\omega}$. Une deuxième décomposition du même type est réalisée sur les champs $\boldsymbol{\Phi}(\boldsymbol{x},\boldsymbol{\omega})$ calculés lors de la première décomposition. Ceci donne : $\boldsymbol{\Phi}_i(\boldsymbol{x},\boldsymbol{\omega}) = \overline{\boldsymbol{\Phi}_i}(\boldsymbol{x}) + \sum_{j=1}^{r_i}\sqrt{\lambda_i^j}\boldsymbol{\psi}_i^j(\boldsymbol{x})\boldsymbol{\phi}_i^j(\boldsymbol{\omega})$ avec $\overline{\boldsymbol{\Phi}_i} = \sum_{k=1}^{N}\boldsymbol{\Phi}_i^k$. La dimension du champ aléatoirement obtenu est : $r = \sum_{i=1}^{d} r_i$ avec r_i le nombre de modes sélectionnés pour rendre compte du i mode obtenu lors de la première décomposition KLE bi-orthogonale. Afin d'évaluer la pertinence du modèle, plusieurs microstructures sont élaborées puis recontruites dans la base. Si la microstructure reconstruite est identique à la microstruture initiale, le test est positif et la validité du modèle réduit est établie. Une telle décomposition eût pu être mise en oeuvre dans le cas de la tentative de couplage EF^2/hyperréduction. A la place, la décomposition retenue est de la forme : $\boldsymbol{u}(\boldsymbol{X},\boldsymbol{x},\boldsymbol{t}) = \sum_{i=0}^{N}\sqrt{\rho_i}\boldsymbol{\Psi}_i(t)\boldsymbol{\Phi}_i(\boldsymbol{X},\boldsymbol{x})$. Cette décomposition a été effectivement réalisée. On note que les états micro ne sont pas décorélés de l'état macro.

Méthode POD minimisant l'erreur sur la sortie Dans [16] et [15] est proposée une méthode de construction de base réduite qui améliore la construction opérée par une simple POD. Le schéma 1.4 illustre cette approche. Constatant que la POD ne prend pas en compte les sorties du système, l'idée est alors d'introduire une telle contrainte dans la construction de l'espace réduit $\boldsymbol{\Phi}$ (étape 2, 1.4). Ainsi pour chaque valeur de paramètre $\boldsymbol{\mu}$, l'écart entre le système complet d'état $\boldsymbol{x}$ et le système réduit d'état $\hat{\boldsymbol{x}}$ est évalué. Une procédure de type Newton-Raphson permet de faire converger le paramètre du système $\boldsymbol{\mu}$ vers une valeur $\boldsymbol{\mu}^*$ telle que l'écart évalué $\boldsymbol{G}$ défini sur la figure 1.4 est maximum. Plus précisément, Les auteurs se sont appuyés, pour résoudre le problème, sur une *trust-region inexact-Newton-conjugate-gradient method* ([68], p. 156). L'état du système complet convergé $\boldsymbol{x}(\boldsymbol{\mu}^*,t)$ est utilisé pour enrichir l'espace réduit $\boldsymbol{\Phi}$ en utilisant une POD par exemple. Dans [16], à la différence de [15], l'espace réduit $\boldsymbol{\Phi}$ est pris comme paramètre $\boldsymbol{\mu}$ et la base réduite est solution du problème d'optimisation. En pratique, le nombre de variables à optimiser devient très important. C'est pourquoi, on utilise une SVD sur des *snapshots* déjà calculés puis on cherche l'espace réduit à optimiser comme combinaison linéaire des $\boldsymbol{\phi}$ préalablement calculés, $\boldsymbol{\Phi} = \boldsymbol{\phi}\cdot\boldsymbol{\gamma}$, ce qui permet de réduire le nombre de variables, au prix d'une perte de la convexité du problème qu'il faut compenser par une initialisation adéquate. Le problème est alors de choisir une base orthonormale optimale $\boldsymbol{\Phi}$ au sens de la maximisation des erreurs sur les sorties :

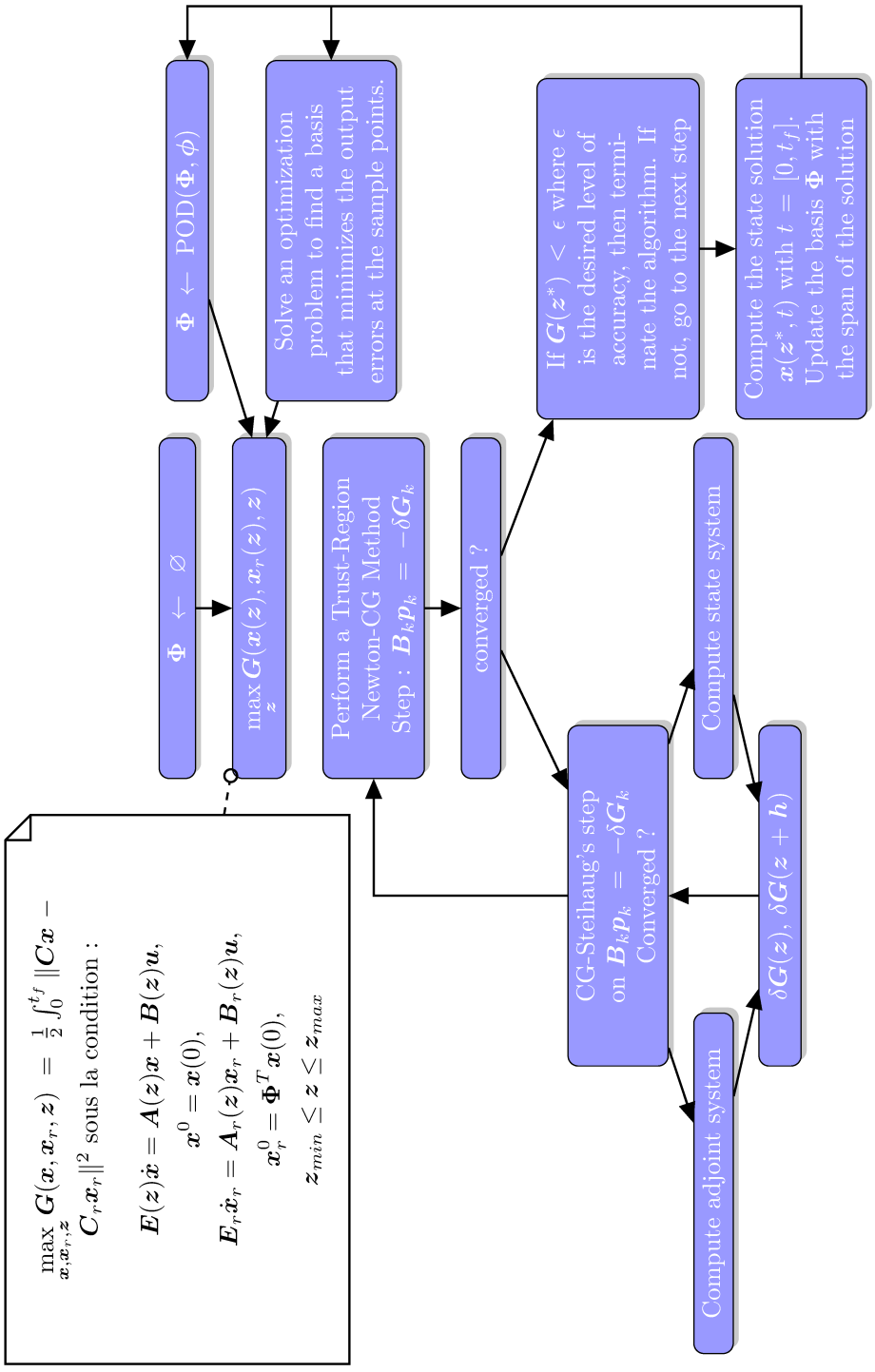


Figure 1.4 – Méthode de construction d'un espace réduit

Figure 1.5 – Images propres créées par snapshot-POD à partir d'une base de données de visages humains. Tiré de [42]. Reconstitution d'un visage humain en bas à droite en utilisant 50 images propres.

$$\min_{\boldsymbol{\phi},\boldsymbol{\alpha}} \boldsymbol{G} = \underbrace{\frac{1}{2}\sum_{k=1}^{S}\int_0^{t_f}(\boldsymbol{g}^k-\hat{\boldsymbol{g}}^k)^T(\boldsymbol{g}^k-\hat{\boldsymbol{g}}^k)\,\mathrm{d}t}_{\text{écart entre les outputs de référence et réduits}} + \underbrace{\frac{\beta}{2}\sum_{j=1}^{m}(1-\boldsymbol{\phi}_j^T\boldsymbol{\phi}_j)^2}_{\text{normalisation}} + \underbrace{\frac{\beta}{2}\sum_{\substack{i,j=1\\ i\neq j}}(\boldsymbol{\phi}_i^T\boldsymbol{\phi}_j)^2}_{\text{orthonorgonalité}} \quad (1.8)$$

sous les contraintes :

$$\begin{cases} \boldsymbol{\Phi}^T\boldsymbol{M}^k\boldsymbol{\Phi}\dot{\boldsymbol{\alpha}}^k + \boldsymbol{\Phi}^T\boldsymbol{K}^k\boldsymbol{\Phi}\boldsymbol{\alpha}^k = \boldsymbol{\Phi}^T\boldsymbol{f}^k,\ k=1,\dots,S \\ \boldsymbol{\Phi}^T\boldsymbol{M}^k\boldsymbol{\Phi}\boldsymbol{\alpha}_0^k = \boldsymbol{\Phi}^T\boldsymbol{M}^k\boldsymbol{u}_0^k,\ k=1,\dots,S \\ \hat{\boldsymbol{g}}^k = \boldsymbol{C}^k\boldsymbol{\Phi}\boldsymbol{\alpha}^k,\ k=1,\dots,S \end{cases} \quad (1.9)$$

La procédure de résolution utilisée est encore celle illustrée sur la figure 1.4 où l'on effectue la substitution : $\boldsymbol{\mu} \leftarrow \boldsymbol{\Phi}$

1.4 La Gappy POD

Dans [25], une méthode dite *Gappy POD* est proposée. Le but de la méthode est de pouvoir reconstruire un champ complet à partir de données lacunaires de ce champ ayant préalablement formé un modèle réduit. Cela consiste simplement à minimiser l'erreur : $\varepsilon^2 = \int_{\Omega_\Pi} \|\tilde{\boldsymbol{\phi}}(\boldsymbol{x}) - \boldsymbol{\Psi}\cdot\boldsymbol{u}\|^2 \mathrm{d}\boldsymbol{x}$. Cette méthode est appliquée à la reconstruction faciale dans [42]. Sur la figure 1.5, chaque image de visage peut être décomposée en un nombre limité d'images propres. Les images utilisées dans cette application comptent 2^{14} pixels et il apparaît que seulement 40 images propres permettent une reconstruction précise à 3 % d'erreur près. Cette technique trouve des applications dans la compression de données et est directement applicable à la reconstruction des champs en mécanique.

1.5 L'EIM (Empirical Interpolation Method)

La méthode EIM est présentée dans [10]. Cette méthode permet d'approximer une fonction non-linéaire d'un paramètre typiquement $\boldsymbol{u}(\boldsymbol{\mu})$ en construisant d'abord une suite d'espaces emboîtés $\hat{\Omega}_1 \subset \cdots \subset \hat{\Omega}_i$ puis en en déduisant des ensembles emboîtés de points d'interpolation $\tilde{\Omega}_1 \subset \cdots \subset \tilde{\Omega}_i$ de la fonction $\boldsymbol{u}$ initiale sur la suite d'espace ainsi construite. Cette méthode d'approximation permet également d'avoir une erreur d'approximation, i. e. de majorer la norme $\|\boldsymbol{u} - \hat{\tilde{\boldsymbol{u}}}\|$ entre la fonction initiale et son interpolée en quelques points par la distance de $\boldsymbol{u}$ à l'espace d'interpolation. Dans les faits, la méthode EIM est un type d'algorithme *greedy*. La construction de l'espace d'interpolation dépend largement en temps et en qualité de la pertinence de l'espace d'échantillonage pour le paramètre $\boldsymbol{\mu}$. En effet, le terme $\boldsymbol{\mu}_i$ retenu est celui pour lequel la distance $\boldsymbol{u}$ à l'espace d'interpolation $\hat{\Omega}$ est minimale. Evaluer une telle distance peut se réléver coûteux. Aussi on réduit généralement a priori l'espace des paramètres en sélectionnant un espace d'échantillonage représentatif. La sélection de l'espace d'échantillonage semble être un problème largement ouvert. Même si la constante de majoration est assez grande, les auteurs arguent de la convergence très rapide de l'écart de $\boldsymbol{u}$ à l'espace d'interpolation pour justifier la pertinence de la majoration. La méthode permet ainsi de ramener les équations différentielles paramétriques non-affines à des équations affines. Elle permet une séparation des paramètres des variables de l'équation différentielle.

Une version discrète de l'EIM a été développée. La méthode en question s'appelle la DEIM [19] (Discrete Empirical Interpolation Method). L'algorithme proposé n'est vraiment qu'une transcription de l'EIM au cas discret. Dans l'article, les considérations de [19] sur l'estimation d'erreur sont analysées. Il faut cependant remarquer que si l'on parvient à majorer rigoureusement l'écart en norme $\|\cdot\|_2$ entre $\boldsymbol{u}$, le champ réel et le champ reconstruit $\hat{\boldsymbol{u}}$ par la méthode DEIM, la majoration de ce majorant reste une question ouverte et difficile selon les auteurs de la DEIM. La difficulté est de trouver une estimation *a posteriori* du majorant sans évaluer $\boldsymbol{u}$ partout, i. e. une approximation à faible coût. Les auteurs tentent de justifier leur heuristique de majoration dans [19] p. 2749 - p. 2750.

1.6 La MPE (Missing Point Estimation)

Motivation Cette méthode est fondée sur la méthode de la *Gappy POD* présentée dans 1.4 développée dans [25] et rappelée à la section 1.4. Le but de la méthode est de pouvoir reconstruire un champ complet à partir de données lacunaires de ce champ ayant préalablement formé un modèle réduit. Cela consiste simplement à minimiser l'erreur : $\varepsilon^2 = \int_{\Omega_\Pi} \|\widetilde{\boldsymbol{\phi}(\boldsymbol{x})} - \boldsymbol{\Psi} \cdot \boldsymbol{u}\|^2 \mathrm{d}\boldsymbol{x}$. Cependant l'étude est menée dans le cas d'une fonctionnelle $\boldsymbol{F}$ linéaire et les résultats ne sont pas généralisés au non-linéaire. Pour fixer les idées, on notera $(\cdot, \cdot)_\Omega$ un produit scalaire sur l'espace Ω. On considère les deux systèmes qui émergent alors

du système de départ :

$$(\partial_t \hat{\boldsymbol{u}}, \hat{\boldsymbol{\phi}})_\Omega = \boldsymbol{F}((\hat{\boldsymbol{u}}, \hat{\boldsymbol{\phi}})_\Omega, ..., (\partial_x \hat{\boldsymbol{u}}, \hat{\boldsymbol{\phi}})_\Omega) \tag{1.10}$$

et

$$(\partial_t \hat{\boldsymbol{u}}, \hat{\boldsymbol{\phi}})_\Omega = (\boldsymbol{F}(\boldsymbol{u}, ..., \partial_x \hat{\boldsymbol{u}}), \hat{\boldsymbol{\phi}})_\Omega \tag{1.11}$$

Comme la fonctionnelle est linéaire, les deux systèmes éq. 1.10 et 1.11 coïncident. Dans le cas de système non-linéaire la projection de l'état du système et du résidu joue un rôle central, notamment dans l'étude de l'erreur menée en particulier dans [20]. Et on remarque qu'effectuer les produits scalaires des équations 1.10 et 1.11 est coûteux en temps de calcul.

Théorie L'objet de la MPE est de diminuer ce coût. Dans [8] est décrite l'approche *MPE* ou *Missing Point Estimation*. L'article présenté prolonge les travaux illustrés par [7] où une méthode de réduction de modèle, très proche de l'hyperréduction décrite dans [58], est appliquée à un cas thermique. Une heuristique pour construire un domaine réduit est proposée. Elle consiste à s'intéresser à la perte d'orthonormalité de la base des déplacements $\boldsymbol{\Phi}$. Lorsque l'on fait de la réduction en utilisant la POD, on construit une base des déplacements orthonormale : ${}^t\boldsymbol{\Phi}\boldsymbol{\Phi} = \boldsymbol{I}$. Dans [8], une justification de sélection des points d'échantillonage est donnée, une théorie est invoquée pour justifier une telle construction et un algorithme de type *greedy* est fourni. Il repose sur l'évaluation d'une quantité appelée *alias sensitivity*. Dans le cas simple où l'espace ambiant est équipé du produit scalaire euclidien canonique, *alias sensitivity* $\boldsymbol{A}$ s'écrit :

$$\boldsymbol{A} = ({}^t\tilde{\boldsymbol{\Phi}}\tilde{\boldsymbol{\Phi}})^{-1} - \boldsymbol{I} \tag{1.12}$$

L'algorithme *greedy* en question consiste à minimiser *alias sensitivity* ce qui revient à minimiser le conditionnement $c({}^t\tilde{\boldsymbol{\Phi}}\tilde{\boldsymbol{\Phi}}) = \frac{\lambda_{max}({}^t\tilde{\boldsymbol{\Phi}}\tilde{\boldsymbol{\Phi}})}{\lambda_{min}({}^t\tilde{\boldsymbol{\Phi}}\tilde{\boldsymbol{\Phi}})}$. Cet algorithme *greedy* est néanmoins très coûteux dans le cas de systèmes dynamiques de grande taille et un critère de préselection s'impose.

Algorithme Ainsi la MPE est une méthode de construction de domaine réduit ou d'échantillonage spatial. Le domaine spatial est échantillonné de telle façon que l'orthonormalité des fonctions de base soit préservée, ce qui a pour effet de rendre la méthode plus performante. Cette méthode est illustrée sur la figure 1.6. La sélection des points appartenant au domaine réduit est faite en vue de respecter l'orthonormalité de l'espace réduit $\boldsymbol{\Phi}$ préalablement constitué. Un algorithme *greedy* est proposé pour calculer effectivement le domaine réduit $\boldsymbol{Z}$. On projette alors à moindre coût sur l'espace réduit ce qui permet d'intégrer les équations différentielles du modèle sur un faible nombre de points et de diminuer le coût de calcul. La figure 1.6 montre que différentes projections peuvent être envisagées. Pour plus de précision, nous renvoyons à [8].

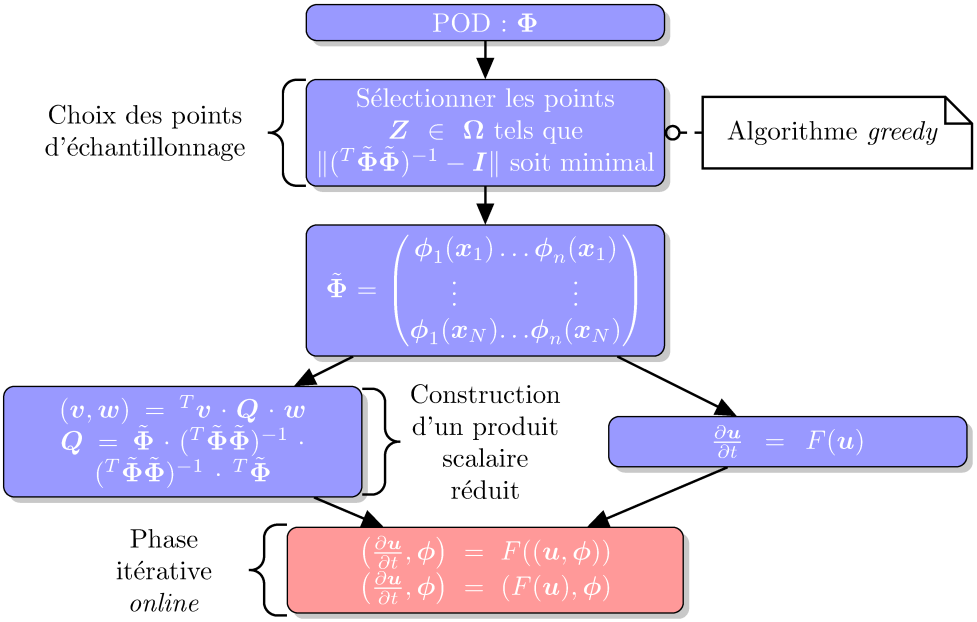


Figure 1.6 – Principe de l'algorithme MPE

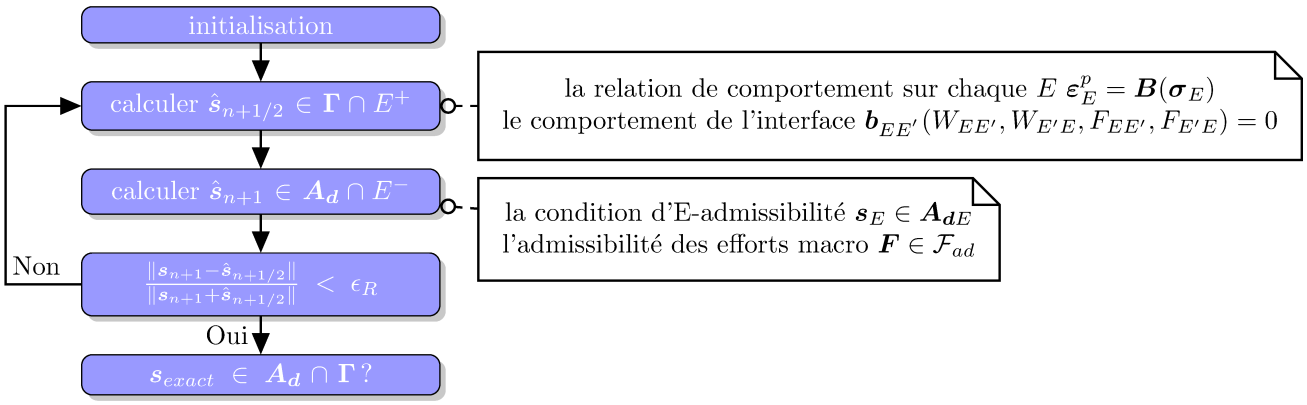


Figure 1.7 – La méthode LATIN. Schéma simplifié.

1.7 La méthode LATIN multiéchelle, la PGD

De la LATIN à la PGD L'objectif de la méthode LATIN est de trouver la solution $\boldsymbol{s}_E = (\dot{\boldsymbol{\varepsilon}}_{pE}, \dot{\boldsymbol{W}}_E, \boldsymbol{\sigma}_E, \boldsymbol{F}_E)$ du problème mécanique pour chaque domaine E où $\dot{\boldsymbol{\varepsilon}}_{pE}$ est le champ de déformation plastique, $\dot{\boldsymbol{W}}_E$ est le champ de déplacement à l'interface, $\boldsymbol{\sigma}_E$ est le champ de contrainte et $\boldsymbol{F}_E$ est le champ de force à l'interface. Après l'initialisation, sur la figure 1.7, l'étape 2 de la méthode LATIN permet de déterminer un point $\hat{\boldsymbol{s}}_{n+1/2}$ pour lequel les relations de comportement du matériau et de l'interface sont vérifiées. L'équilibre mécanique est assuré lors de l'étape 3. Des itérations entre ces deux étapes permettent d'assurer la convergence vers une solution du problème vérifiant à la fois la loi de comportement et l'équilibre mécanique. L'étape 2 de l'algorithme suppose la résolution de l'équilibre mécanique sur un domaine à la fois spatial et temporel $\Omega_E \times I_i^G$. Dans le cas d'un calcul multiéchelle, l'étape 2 se subdivise en plusieurs sous-étapes. Parmi elles, l'une consiste consiste à résoudre N problèmes micro définis sur $\Omega_E \times I_i^G$ [52] p. 45 et p. 58. Une façon classique de procéder est de résoudre ce problème de manière incrémental. Le problème est ramené alors à un problème éléments finis classique sur un intervalle temporel et sur les N sous-structures. Ce problème présente une grande parenté avec les problèmes obtenus par le couplage de l'hyperréduction avec les EF^2, objet de cette thèse.

Afin de résoudre les problèmes micro de la LATIN multidimensionnelle, la PGD (Proper Generalized Decomposition) en mode solveur se présente comme une technique de résolution de problèmes alternative aux méthodes incrémentales classiques qui nécessitent l'inversion d'un système à chaque élément de temps. Elle est issue de la famille des méthodes d'approximation radiale sur l'espace-temps. Elle a été utilisée dans la méthode LATIN lors de l'étape linéaire dans [44] auquel on renvoit pour la mise en place détaillée. Elle permet de diminuer le temps de calcul par une approche non-incrémentale en utilisant des opérateurs de faible dimension.

Algorithme de la PGD Dans [21] est décrite la méthode PGD et son application à des cas de problèmes linéaires de transport-diffusion est présentée. Contrairement aux méthodes de type POD, les méthodes de type PGD ne nécessitent pas la connaissance *a priori* de *snaphots* pour synthétiser le modèle réduit. Sur la figure 1.8 est présenté l'algorithme de la PGD appliqué à un problème linéaire. De prime abord, on s'aperçoit que la résolution d'un problème linéaire se fait en plusieurs itérations contrairement aux algorithmes de résolution éléments finis classiques. $\boldsymbol{a}$ représente classiquement une forme bilinéaire coercive et $\boldsymbol{b}$ une forme linéaire. On cherche la solution du problème d'évolution $\boldsymbol{u}$ dans un espace tensoriel $\boldsymbol{R} \otimes \boldsymbol{U}$. Il existe néanmoins certaines similarités entre la PGD et la POD. Ainsi, dans la PGD le déplacement est cherché sous la forme : $\boldsymbol{u}(\boldsymbol{x}, t) = \sum_{i=1} \boldsymbol{R}_i(\boldsymbol{x}) \cdot \boldsymbol{U}_i(t)$ tandis que la POD construit à partir de *snapshots* de la forme

$$\boldsymbol{Q} = \begin{pmatrix} u_1^1 & u_1^2 & \dots & u_1^p \\ u_2^1 & u_2^2 & \dots & u_2^p \\ \vdots & \vdots & \ddots & \vdots \\ u_{N_n}^1 & u_{N_n}^2 & \dots & u_{N_n}^P \end{pmatrix} \tag{1.13}$$

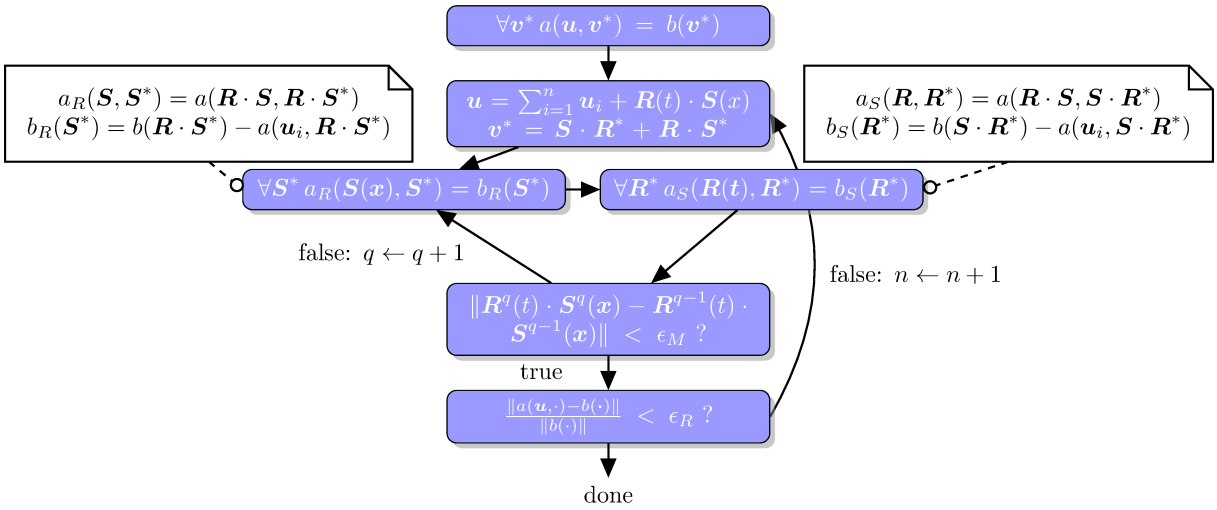


Figure 1.8 – L'algorithme PGD.

et en résolvant un problème aux valeurs propres

$$(\boldsymbol{Q}\boldsymbol{Q}^T)\boldsymbol{\phi} = \lambda \boldsymbol{\phi} \tag{1.14}$$

une décomposition de $\boldsymbol{u}$ de la forme :

$$\boldsymbol{u}(\boldsymbol{x}, t) = \sum_{i=1} \boldsymbol{\phi}_i(\boldsymbol{x}) \cdot \boldsymbol{U}_i(t) \tag{1.15}$$

qui ne va pas sans rappeler la décomposition de la PGD.

Application de la PGD Dans [22], la PGD est appliquée à l'équation de Fokker-Planck. Les paramètres d'entrées (conditions aux limites, paramètres de loi de comportement matériaux) du modèle sont traités de manière analogue au temps et à la position dans l'équation 1.15 et insérés ainsi comme terme multiplicatif à déterminer. L'approche par PGD consiste ainsi à recenser l'ensemble des paramètres d'un modèle, à créer un problème de très grande dimension sur lequel on fait levier et que cette technique permet *in fine* de réduire. Une implémentation de la PGD est décrite dans [3] et dans [22] des exemples d'applications de la PGD sont proposés, notamment autour de l'équation de Fokker-Planck.

En guise d'exemple, on cherche à résoudre le problème suivant :

$$\frac{\partial \boldsymbol{u}}{\partial t} - a\Delta \boldsymbol{u} = \boldsymbol{f}(\boldsymbol{x}, t) \tag{1.16}$$

et des conditions de bord appropriées. De plus,

$$\boldsymbol{f}(\boldsymbol{x}, t) = \sum_{i=1}^{i=r} \boldsymbol{\gamma}_i \cdot \boldsymbol{C}_i(x, t) \tag{1.17}$$

régi par l'équation :

$$\frac{d\boldsymbol{C}_i(x,t)}{dt} = \sum_{j=1}^{j=r} \boldsymbol{\alpha}_{ij} \boldsymbol{C}_j(x,t) \tag{1.18}$$

On peut chercher alors à globaliser la résolution du modèle en écrivant :

$$\boldsymbol{C}_i(\boldsymbol{x},t) = \sum_{q=1}^{q=m} X_q^{C,i}(\boldsymbol{x}) \cdot T_q^{C,i}(t) \tag{1.19}$$

ou en intégrant l'indice i :

$$\boldsymbol{C}(\boldsymbol{x},t,c) = \sum_{q=1}^{q=n} X_q(\boldsymbol{x}) \cdot T_q(t) A_q(c) + R(x) \cdot S(t) \cdot W(c) \tag{1.20}$$

C'est une idée similaire qui a donné l'hyperréduction multidimensionnelle : l'idée de regrouper dans un calcul de grande dimension.

1.8 Méthodes de réduction tensorielles

Motivation La PGD est une méthode d'approximation de type glouton tensoriel. L'étude d'un algorithme de type PGD est proposée dans [27]. Les méthodes de type PGD appartiennent à une catégorie plus large de méthodes : les méthodes d'approximation tensorielles. L'objet de ces méthodes est de trouver une approximation optimale dans un espace tensoriel. Dans ce champ, on s'intéresse généralement à la résolution de deux types de problème : $\min_{v \in \mathcal{S}_X} \|u - v\|$ et $\min_{v \in \mathcal{S}_X} \|Av - b\|$. Elles reposent sur l'emploi d'algorithme *greedy*. Les problématiques sous-tendant ce champ de recherche consistent d'une part à définir des algorithmes *greedy* bien posés et d'autre part à développer des algorithmes capables de résoudre ces problèmes.

Exemple d'application Les modèles réduits trouvent une application assez naturelle dans le domaine du calcul stochastique. Les paramètres des EDP sont aléatoires. Un grand nombre de simulations pour ces paramètres alétoires est requis ce qui implique un grand nombre de calculs éléments finis et une explosion du coût de calcul. Les modèles d'ordre réduit ont sûrement un rôle à jouer dans ce domaine. Dans la suite, nous examinons les avancées récentes dans ce domaine.

Algorithme 1 : Algorithme greedy

1 Initialize $\boldsymbol{u} = 0$
2 Find $(\boldsymbol{u}^{(1)}, ..., \boldsymbol{u}^{(d)}) \in \underset{(\boldsymbol{u}^{(1)},...,\boldsymbol{u}^{(d)}) \in V_1 \times \cdots \times V_d}{\arg\min} \mathcal{E}(\boldsymbol{u} + \boldsymbol{u}^{(1)} \otimes ... \otimes \boldsymbol{u}^{(d)})$
3 Update $\boldsymbol{u} \leftarrow \boldsymbol{u} + \boldsymbol{u}^{(1)} \otimes ... \otimes \boldsymbol{u}^{(d)}$

Algorithme greedy pour les fonctionnelles linéaires symétriques L'algorithme 1 présente est un algorithme *greedy* générique. Après l'initialisation vient la recherche de la minimisation d'une fonctionnelle d'énergie $\mathcal{E}$. Le choix de la fonctionnelle donne lieu à différents algorithmes *greedy*. La recherche du minimum est effectuée en dérivant l'expression de la fonctionnelle d'énergie : on obtient les équations d'Euler. Si dans le cas d'une fonctionnelle d'énergie de la forme $\mathcal{E}(\boldsymbol{u}) = \frac{1}{2}\boldsymbol{a}(\boldsymbol{u},\boldsymbol{u}) - \boldsymbol{b}(\boldsymbol{u})$ où $\boldsymbol{a}$ est symétrique coercive les équations d'Euler permettent de déterminer un minimum, ce n'est plus vérifié dans le cas où $\boldsymbol{a}$ n'est plus symétrique. On trouve alors dans la littérature un vaste éventail d'algorithmes pour traiter ce cas comme en atteste le tableau 1.2. Or en mode hyperréduit, un calcul éléments finis possède une forme non symétrique. Dès lors que l'on dispose d'algorithmes capables de traiter des formes nonsymétriques, l'application d'une méthode d'hyperréduction de modèle devient possible et constitue un axe de recherche intéressant.

Algorithme greedy pour les fonctionnelles linéaires non-symétriques Pour une forme $\boldsymbol{a}$ symétrique, l'algorithme de minimisation en norme L^2 revient à un algorithme de Galerkin-PGD exposé sur 1.8 appliqué à $\boldsymbol{A}^*\boldsymbol{A}$ et $\boldsymbol{A}^*\boldsymbol{l}$. Cet algorithme ne converge pas rapidement. Dans l'algorithme de minimax, à $\boldsymbol{u}_1$, $\tilde{\boldsymbol{u}}_1$ fixés, $\tilde{\boldsymbol{u}}_2$ apparaît comme un multiplicateur de Lagrange permettant d'assurer pour $\boldsymbol{u}_2$: $\boldsymbol{a}(\boldsymbol{u} + \boldsymbol{u}_1 \otimes \boldsymbol{u}_2, \tilde{\boldsymbol{u}}_1) - \boldsymbol{l}(\tilde{\boldsymbol{u}}_1) = 0$. L'objectif est de minimiser $\frac{1}{2}\|\boldsymbol{u}_1 \otimes \boldsymbol{u}_2\|$ sous cette contrainte. Même raisonnement en échangeant $1 \leftrightarrow 2$. Maintenant à $\tilde{\boldsymbol{u}}_1$ et $\tilde{\boldsymbol{u}}_2$ fixé, le problème se ramène à un problème avec une fonctionnelle $\boldsymbol{a}$ symétrique : $\min_{(\boldsymbol{u}_1 \otimes \boldsymbol{u}_2) \in \boldsymbol{V}_1 \times \boldsymbol{V}_2} \frac{1}{2}\|\boldsymbol{u}_1 \otimes \boldsymbol{u}_2\|_V^2 - \boldsymbol{a}(\boldsymbol{u} + \boldsymbol{u}_1 \otimes \boldsymbol{u}_2, \tilde{\boldsymbol{u}}_1 \otimes \tilde{\boldsymbol{u}}_2)$. Ainsi, en ajoutant une contrainte, l'algorithme du minimax permet de ramener le problème de fonctionnelle non symétrique à un problème symétrique. Dans le cas de l'algorithme *X-greedy*, il n'existe pas à notre connaissance d'implémentation permettant de calculer $(\boldsymbol{u}_1, \boldsymbol{u}_2)$ à chaque itération.

Méthodes de résolution Ces méthodes s'inspirent d'algorithmes de point fixe. Dans [50] afin de résoudre le problème 1.8 trois algorithmes sont décrits : $PGD - P$ p. 16, $PGD - S$ p. 17 et $PGD - P^*$ p. 18. Le $PGD - P$ correspond à celui présenté 1.8, le $PGD - S$ est celui de la figure 1.8 où $\boldsymbol{u} = \sum_{i=1}^{n} \boldsymbol{R}_i(t) \cdot \boldsymbol{S}_i(x) + \boldsymbol{R}(t) \cdot \boldsymbol{S}(x)$, autrement dit les couples d'itérations $(\boldsymbol{S}_i(x), \boldsymbol{R}_i(t))$ calculés précédemment sont réévalués ce qui permet une meilleure convergence. L'algorithme $PGD - P^*$ est intermédiaire entre $PGD - P$ et $PGD - S$ au sens où seulement une réévaluation des itérés $\boldsymbol{R}_i(t)$ est effectuée après un calcul de $\boldsymbol{S}(x)$. Les $\boldsymbol{S}_i(x)$ ne sont réévalués. Ainsi tandis que l'algorithme de type $PGD - P$ dans [50] s'assimile à un *Pure Greedy Algorithm (PGA)* décrit dans [63] p. 80, les deux autres sont du type *Orthogonal Greedy Algorithm* ([63] p. 80)

1.9 La méthode GNAT

Construction de l'espace réduit La méthode GNAT [18] est une méthode de réduction de modèle, assez proche de l'hyperéduction de modèle telle que formulée dans [59] et exposée à la section 1.10. Elle repose sur l'utilisation de base réduite construite par POD à partir de *snapshots* (cf. 1.3) et sur le choix d'un domaine réduit (cf. 1.5). Elle est utilisée dans un

Minimisation en norme L^2	
$(\boldsymbol{u}_1, \boldsymbol{u}_2) \in \underset{(\boldsymbol{u}_1,\boldsymbol{u}_2)\in V_1\times V_2}{\arg\min} \ \|\boldsymbol{A}(\boldsymbol{u}_1\otimes\boldsymbol{u}_2) - \boldsymbol{L}\|_V^2$	
Minimisation duale	
$(\boldsymbol{u}_1, \boldsymbol{u}_2) \in \underset{(\boldsymbol{u}_1,\boldsymbol{u}_2)\in V_1\times V_2}{\arg\min} \ \|(\boldsymbol{R}_V)^{-1}(\boldsymbol{A}(\boldsymbol{u}_1\otimes\boldsymbol{u}_2) - \boldsymbol{L})\|_V^2$	
Minimax	
$(\boldsymbol{u}_1, \tilde{\boldsymbol{u}}_1, \boldsymbol{u}_2, \tilde{\boldsymbol{u}}_2) \in \underset{(\tilde{\boldsymbol{u}}_1,\tilde{\boldsymbol{u}}_2)\in V_1\times V_2}{\arg\max} \ \underset{(\boldsymbol{u}_1,\boldsymbol{u}_2)\in V_1\times V_2}{\min} \ \frac{1}{2}\|\boldsymbol{u}_1\otimes\boldsymbol{u}_2\|_V^2 - \boldsymbol{a}(\boldsymbol{u}+\boldsymbol{u}_1\otimes\boldsymbol{u}_2, \tilde{\boldsymbol{u}}_1\otimes\tilde{\boldsymbol{u}}_2) + \boldsymbol{l}(\tilde{\boldsymbol{u}}_1\otimes\tilde{\boldsymbol{u}}_2)$	
Dual Greedy	
$(\tilde{\boldsymbol{u}}_1, \tilde{\boldsymbol{u}}_2) \in \underset{(\boldsymbol{u}_1,\boldsymbol{u}_2)\in V_1\times V_2,\|\boldsymbol{u}_1\otimes\boldsymbol{u}_2\|_V}{\arg\max} \ \boldsymbol{l}(\tilde{\boldsymbol{u}}_1\otimes\tilde{\boldsymbol{u}}_2) - \boldsymbol{a}(\boldsymbol{u}, \tilde{\boldsymbol{u}}_1\otimes\tilde{\boldsymbol{u}}_2)$	
$(\boldsymbol{u}_1, \boldsymbol{u}_2) \in \underset{(\boldsymbol{u}_1,\boldsymbol{u}_2)\in V_1\times V_2,\|\boldsymbol{u}_1\otimes\boldsymbol{u}_2\|_{iA}=1}{\arg\max} \ \boldsymbol{a}(\boldsymbol{u}_1\otimes\boldsymbol{u}_2, \tilde{\boldsymbol{u}}_1\otimes\tilde{\boldsymbol{u}}_2)$	
X-Greedy	
$(\boldsymbol{u}_1, \boldsymbol{u}_2) \in \underset{(\boldsymbol{u}_1,\boldsymbol{u}_2)\in V_1\times V_2\,(\tilde{\boldsymbol{u}}_1,\tilde{\boldsymbol{u}}_2),\|\tilde{\boldsymbol{u}}_1\otimes\tilde{\boldsymbol{u}}_2\|_V=1}{\arg\min} \ \underset{}{\sup} \ \boldsymbol{l}(\tilde{\boldsymbol{u}}_1\otimes\tilde{\boldsymbol{u}}_2) - \boldsymbol{a}(\boldsymbol{u}+\boldsymbol{u}_1\otimes\boldsymbol{u}_2, \tilde{\boldsymbol{u}}_1\otimes\tilde{\boldsymbol{u}}_2)$	
Décomposition	
$(\boldsymbol{u}_1, \boldsymbol{u}_2) \in \underset{(\boldsymbol{u}_1,\boldsymbol{u}_2)\in V_1\times V_2}{\arg\min} \ \frac{1}{2}\boldsymbol{b}_s(\boldsymbol{u}_{n-1}+\boldsymbol{u}_1\otimes\boldsymbol{u}_2, \boldsymbol{u}_{n-1}+\boldsymbol{u}_1\otimes\boldsymbol{u}_2) - \boldsymbol{l}(\boldsymbol{u}_1\otimes\boldsymbol{u}_2) - \boldsymbol{b}(\boldsymbol{u}_{n-1}, \boldsymbol{u}_1\otimes\boldsymbol{u}_2)$	

Table 1.2 – Algorithmes *greedy* pour la résolution de problèmes associés à des formes nonsymétriques

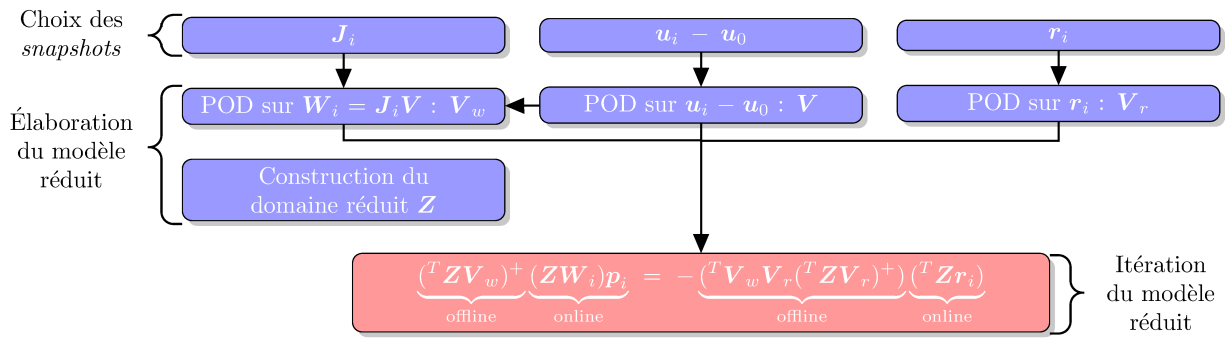


Figure 1.9 – La méthode GNAT

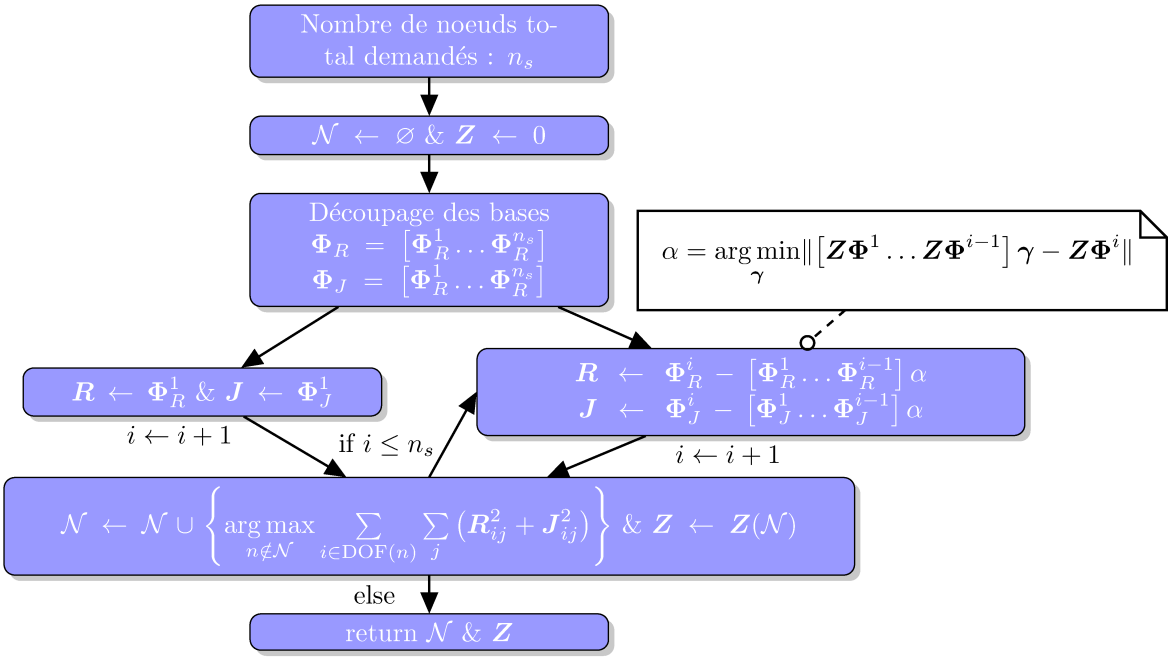


Figure 1.10 – Construction du domaine réduit dans la méthode GNAT

cas non-linéaire. La méthode GNAT prévoit une base pour les réactions et une base pour la matrice de rigidité du Newton-Raphson ce que n'inclut pas l'hyperréduction où seules les quantités d'intérêt - celles dont on veut connaître la valeur sur l'ensemble du domaine - possèdent une base associée. Les systèmes que l'on résout dans le cas de chacune des deux méthodes [60] ou [18] ne sont pas les mêmes. Dans le cas de la méthode GNAT, la matrice de rigidité et les réactions sont d'abord projetées sur l'espace réduit puis échantillonnées ainsi que le montre la figure 1.9. Les *snapshots* de la méthode GNAT sont construits à partir des itérés du système de Newton-Raphson tandis que ceux de l'hyperréduction sont choisis parmi les états convergés. Plus précisément dans le cas de la GNAT, les snapshots sont constitués des itérés du Newton-Raphson auxquels on a enlevé la condition initiale $\boldsymbol{u}_0$ correspondant à la valeur de l'état à la fin de l'incrément précédant. Ce ne sont ni les incréments convergés que l'on choisit directement ni les valeurs brutes des états convergés.

Construction du domaine réduit (RID) En construisant une base pour les états itérés, les auteurs de la méthode GNAT réussissent, en s'appuyant sur la procédure d'estimation de l'erreur de la DEIM à donner une estimation de l'erreur pour cet algorithme de Newton-Raphson itératif ce qui n'existe pas dans l'hyperréduction. Alors que dans la méthode d'hyperréduction le choix du domaine d'échantillonage est purement heuristique, dans la méthode GNAT, il répond à l'objectif de minimiser la projection orthogonale associée à la méthode *Gappy Pod*, ce qui a pour effet de minimiser la majoration de l'erreur. L'algorithme proposé pour la construction du domaine réduit dans le cas de la GNAT (fig. 1.10) ne sélectionne pas les degrés de liberté mais ajoute l'ensemble des degrés de liberté associés à chaque noeud ce qui permet d'assurer l'équilibre aux noeuds du domaine réduit. L'hyperréduction, quant à elle, agit sur des degrés de liberté uniquement. Certains noeuds jugés importants peuvent être ajoutés par l'algorithme de construction de la méthode GNAT. Les bases des déplacements pour la méthode GNAT sont construites à partir des *incréments* de déplacements $\Delta\boldsymbol{u}$ et non

pas des déplacements eux-mêmes comme par ailleurs on fait en hyperréduction. Les bases contruites pour approximer la matrice de rigidité et le vecteur des réactions internes sont élaborées à partir d'états non convergés dans le cas de la méthode GNAT. Plus précisément, la construction du domaine d'intégration réduit est exposée dans [17] p. 173 et [18]. Sur la figure 1.10 est présentée une version simplifiée de cet algorithme. Définissant le nombre de noeuds n_s que doit compter le domaine réduit, cet algorithme effectue la sélection des noeuds et construit l'opérateur de projection $\boldsymbol{Z}$. Pour cela, les bases sont découpées en n_s sous-ensembles de vecteurs $\boldsymbol{\Phi}^i$. A la manière de la DEIM (cf. 1.5) et inspiré par la *Gappy POD* (cf. 1.4), l'orthogonal du sous-espace $\boldsymbol{Z}\boldsymbol{\Phi}^i$ par rapport aux sous-espaces $(\boldsymbol{Z}\boldsymbol{\Phi}^1, ..., \boldsymbol{Z}\boldsymbol{\Phi}^{i-1})$ est utilisé pour ajouter le noeud n au domaine d'intégration réduit. Le noeud ajouté est celui qui correspond au degré de liberté d'intensité maximum sur ce sous-espace.

Exemples d'application Dans [17] puis dans [18] l'exemple applicatif donné appartient au champ de la mécanique des fluides avec des équations faiblement non-linéaires sans variable interne. Dans [5] est présentée une méthode de réduction de systèmes dynamiques nonlinéaires $\frac{d\boldsymbol{u}}{dt} = f(\boldsymbol{u}(t), t)$ à modèle variable dépendant de l'état $\boldsymbol{u}$ dans lequel se trouve le système dynamique. L'espace des états $\boldsymbol{u}$ est fractionné en sous-domaines. Dans chaque sous-domaine, un modèle réduit GNAT est construit conformément à la procédure 1.9. Un algorithme est proposé pour déterminer quel est celui qui, parmi l'ensemble des centres des domaines, est le plus proche de $\hat{\boldsymbol{u}}$. Ainsi l'état estimé $\hat{\boldsymbol{u}}$ est de la forme : $\hat{\boldsymbol{u}} = \hat{\boldsymbol{u}}_0 + (\boldsymbol{V}_1 \cdot \boldsymbol{q}_1, ..., \boldsymbol{V}_n \cdot \boldsymbol{q}_n)$. [4] traite d'un problème de stabilisation d'un système dynamique réduit construit à partir d'un système dynamique de grande dimension.

1.10 L'hyperréduction de modèle

Un processus adaptatif L'hyperréduction de modèle est une méthode de réduction de modèle adaptative. Ainsi que le présente la figure 1.12, des phases de calcul *offline* où le modèle réduit est élaboré alternent avec des phases de calcul *online* où ce dernier est exploité. L'alternance dépend de la pertinence de la solution réduite convergée obtenue. Lors d'une phase de transition *online-offline*, les champs de variable interne sont extrapolés du domaine d'intégration réduit à l'ensemble de la structure en utilisant une *Gappy POD*. Le critère de transition exposé dans [59] consiste à s'assurer de la réalisation de l'équilibre des forces $\boldsymbol{F}^{int}(\hat{\boldsymbol{u}}) = \boldsymbol{F}^{ext}(\hat{\boldsymbol{u}})$ au sens éléments finis sur le domaine d'intégration réduit $\boldsymbol{\Omega}_\Pi$. Une variante de cet indicateur consiste à évaluer $\|\hat{\boldsymbol{\sigma}} - \boldsymbol{\sigma}\|$, i. e. l'erreur sur le champ de contrainte sur le domaine d'intégration réduit. Dans les deux cas, il s'agit d'indicateurs d'erreur qui ne sont en rien des estimateurs d'erreur. On pourrait penser à les améliorer en utilisant une technique statistique de validation croisée sur différents RID. L'erreur entre le modèle complet et le modèle réduit demeure indéterminée. Pour une méthode d'estimation d'erreur, on renvoie à la section 1.11.

Construction d'un espace réduit Dans [59] est présenté une méthode d'enrichissement de l'espace réduit à partir de *snapshots*. Cette méthode de construction est appliquée sur

l'ensemble des bases de déplacements et de variables internes composantes par composantes, i. e. on ne traite que des bases scalaires. Comme illustré sur la figure 1.11, on détermine l'orthogonal du *snapshot* $\boldsymbol{u}$ sur l'espace réduit $\boldsymbol{\Phi}$ existant. Puis on adjoint cette projection à $\boldsymbol{\Phi}$ et on adjoint également les coordonnées de $\boldsymbol{u}$ dans $\begin{pmatrix}\boldsymbol{\Phi} & \boldsymbol{u}_n\end{pmatrix}$ à l'espace des coordonnées réduites $\boldsymbol{\Psi}$. C'est sur cet espace $\boldsymbol{\Psi}$ augmenté que l'on pratique une POD (cf. 1.3). Ainsi dans [60], l'algorithme de construction de base est appliqué composante par composante, autrement dit, il n'est ni vectoriel, ni tensoriel. Ce point soulève de nombreuses questions quand il s'agit de constituer une base du champ tensoriel de contrainte. Construit par combinaisons linéaires à partir de champs qui vérifient l'équilibre mécanique (aux erreurs d'approximation éléments finis près), les champs de contraintes générés sur la base ne sont plus statiquement admissibles. Développer un algorithme tensoriel pour générer la base des contraintes permettrait de s'assurer d'un champ statiquement admissible.

Construction du domaine réduit (RID) Cette construction est un problème crucial plus général que son application à l'hyperréduction puisqu'il se ramène à la question de savoir où et quels capteurs disposer dans l'instrumentalisation d'un essai expérimental. Trouver les mesures pertinentes à obtenir pour maximiser la précision sur une quantité d'intérêt, tel est le problème de la construction du domaine réduit ; les éléments du domaine réduit trouvent leur analogie en expérimental avec la notion de capteur. Dans [59], la construction du domaine d'intégration réduit a lieu une fois que les bases réduites $(\boldsymbol{\Phi}, \boldsymbol{\Psi}, ...)$ des champs de déplacements et de variables internes ont été déterminées. C'est à partir des bases réduites que l'on construit le RID. L'heuristique choisie consiste, pour chaque vecteur de base, à sélectionner les coordonnées maximum, à remonter aux degrés de liberté éléments finis et à choisir les éléments finis qui comportent ces degrés de liberté. L'ensemble de ces éléments finis auxquels on ajoute ceux sur lesquelles agissent des conditions aux limites constitue le RID. La pertinence de cette heuristique se mesure à l'aune de la précision des solutions obtenues par rapport à un calcul éléments finis complet. Il est possible d'améliorer cet algorithme en utilisant à la place une méthode de type DEIM (cf. 1.5) comme dans la méthode GNAT (cf. 1.9)

Comparaison de l'hyperréduction avec une méthode de réduction de modèle adaptative Dans [40] on applique et on compare des méthodes de réduction de modèle adaptative pour un matériau endommageable. La résolution de ce problème s'appuie sur l'utilisation d'un algorithme de gradient conjugué modifié dont l'espace de Krylov est initialisé par une base réduite constituée d'un ensemble $\boldsymbol{\phi}_0$ de *snapshots* initiaux et de l'ensemble $\boldsymbol{\psi}$. $\boldsymbol{\psi}$ correspondant aux solutions convergées des valeurs des déplacements $\boldsymbol{u}$ obtenues au cours des pas de temps précédents et orthogonalisées par un procédé de Gram-Schmidt en prenant en compte l'espace engendré par la base réduite initiale $\boldsymbol{\phi}_0$ et $\boldsymbol{\psi}$. Au cours de la simulation, $\boldsymbol{\psi}$ grandit. Afin de réduire la dimension de cet espace et d'augmenter l'efficacité de l'algorithme, une SVD est effectuée sur l'ensemble des *snapshots* $(\boldsymbol{\phi}_n \quad \boldsymbol{\psi}_n)$ ce qui donne un nouvel espace $(\boldsymbol{\phi}_{n+1} \quad \boldsymbol{\psi}_{n+1})$. Par rapport à la méthode d'hyperréduction, cette approche présente l'avantage de tirer partie des $\delta\boldsymbol{u}$ au cours des itérations ; la méthode d'hyperréduction se cantonnant à la construction d'une base réduite à partir de la valeur des $\Delta\boldsymbol{u}$ convergés. L'enrichissement de la base réduite à partir des espaces de Krylov ne se fait que si la norme du résidu réduit relative à la force externe réduite est très supérieure à la norme

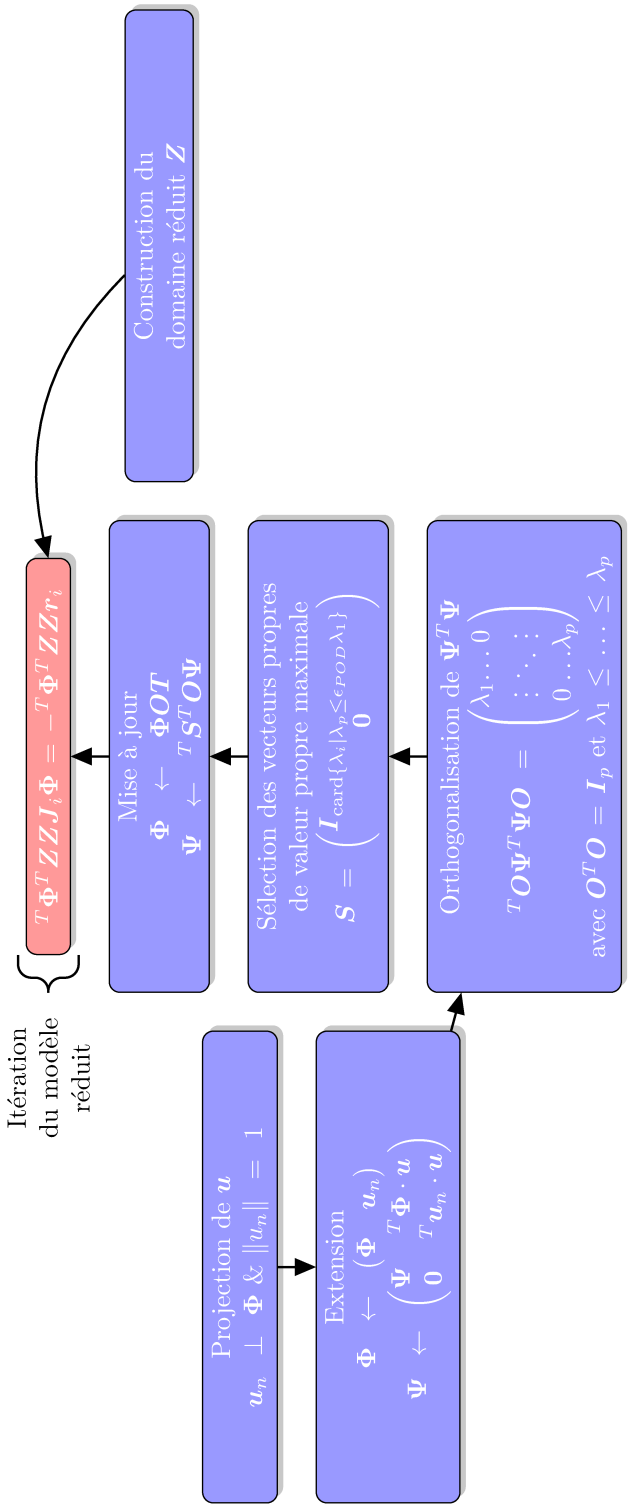


Figure 1.11 – Construction des bases réduites en hyperréduction de modèle

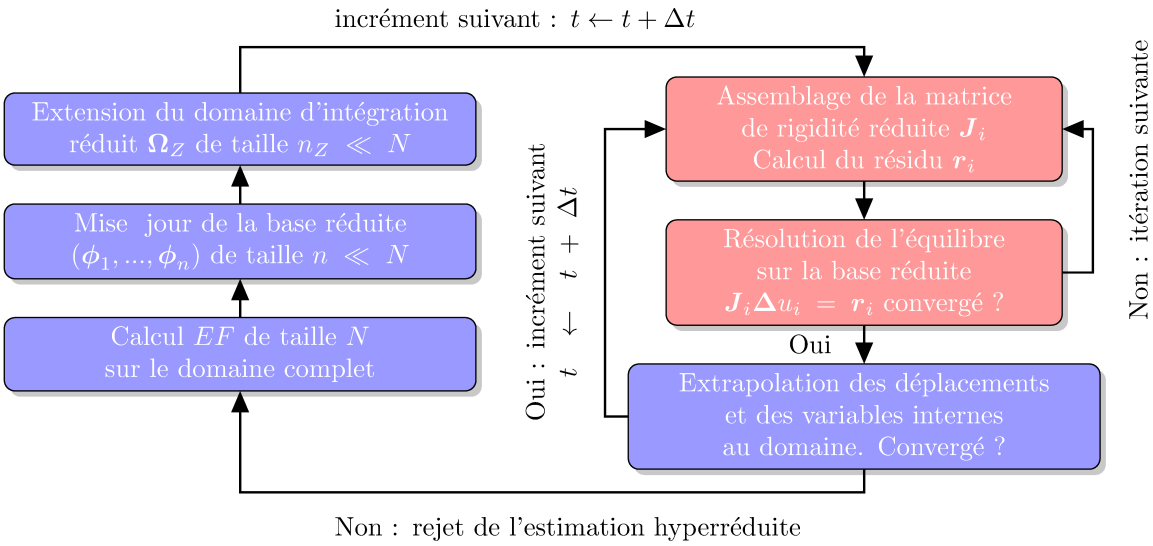


Figure 1.12 – Méthode d'hyperréduction

du résidu relative à la force extérieure. La méthode proposée implique le calcul du résidu et du résidu réduit, ce qui est en pratique fort coûteux en temps. Dans [40], une comparaison de l'algorithme mise en place avec l'hyperréduction de modèle montre l'incapacité de la méthode de réduction à prendre en compte la dissipation d'énergie associée à l'endommagement de la structure.

1.11 Méthode de réduction avec estimateur d'erreur

Objectif : estimation d'erreur au service de la construction du modèle réduit. Cadre général des algorithmes gloutons faibles Dans [63] sont recensés les différents types d'algorithmes *greedy* ou glouton, des preuves de convergence sont fournies, des applications notamment dans le domaine de la reconstruction de signal sont décrites. Un algorithme *greedy* construit une suite d'approximation d'une fonction. Les éléments de cette suite sont choisis dans un espace d'approximation appelé *dictionnaire* Ω. Ces algorithmes peuvent être classifiés en plusieurs grandes familles. Sans entrer dans les détails, cette classification repose sur deux critères : la manière dont on choisit l'élément du dictionnaire qui permet de construire le $i+1$-ème terme de la suite d'approximation $\hat{\Omega}_1 \subset \cdots \subset \hat{\Omega}_i$ et la manière dont on construit effectivement le $i+1$-ème terme de la suite à partir des termes précédents et de l'élément du dictionnaire sélectionné. Dans l'article [13], les auteurs font le lien entre les concepts développés dans le livre [63] p. 80 - p. 81 et l'estimation d'erreur en réduction de modèle. Nous nous intéressons à l'algorithme de type glouton faible qui permet d'approximer une fonction de manière efficace lorsque le coût de calcul associé à ses évaluations en des points particuliers est élevé. C'est typiquement le cas dans les applications en mécanique des matériaux où la connaissance de la valeur de la fonction pour une valeur de paramètre $\boldsymbol{\mu}$

nécessite de faire un calcul éléments finis. Ainsi au lieu de chercher le minimum de la norme de la projection sur le dictionnaire $\|\boldsymbol{u} - \hat{\boldsymbol{u}}\|$, on tente de l'encadrer par une borne dont le calcul est peu coûteux :

$$a\hat{\Delta}_i(\boldsymbol{\mu}) \leq \|\boldsymbol{u}(\boldsymbol{\mu}) - \hat{\boldsymbol{u}}_i(\boldsymbol{\mu})\| \leq b\hat{\Delta}_i(\boldsymbol{\mu}) \tag{1.21}$$

où $\hat{\boldsymbol{u}}_i \in \hat{\Omega}_i$. L'idée est ici d'essayer de développer un algorithme glouton pour estimer l'erreur à faible coût. En effet, une méthode de réduction de modèle fournit généralement à faible coût $\hat{\boldsymbol{u}}$. Mais la plupart du temps, une estimation d'erreur à faible coût est hors de portée. Par exemple, l'hyperréduction ne dispose que d'un indicateur d'erreur. A ce jour, à notre connaissance, aucun argument théorique n'a été avancé pour justifier que l'estimateur d'erreur mené par un calcul d'hyperréduction vérifie l'équation 1.21.

Pour fixer les idées, le *dictionnaire* Ω en analyse paramétrique en mécanique des matériaux peut être l'ensemble des déplacements à l'équilibre mécanique $\boldsymbol{u}(\boldsymbol{\mu})$ paramétrés par une variable que l'on note souvent $\boldsymbol{\mu}$ dans la littérature et qui correspond soit à un éventail de paramètres matériaux, soit à une modification de conditions aux limites en bord de structure. De même que les itérées de l'algorithme de Newton-Raphson lors de la résolution d'un problème non linéaire en mécanique des matériaux sont pilotées par la matrice tangente $\boldsymbol{K}$, de même l'algorithme glouton est piloté par une estimation à faible coût $\hat{\Delta}_i(\boldsymbol{\mu})$ de la valeur $\|\boldsymbol{u}(\boldsymbol{\mu}) - \hat{\boldsymbol{u}}(\boldsymbol{\mu})\|$. L'enrichissement de l'espace $\hat{\Omega}$ est effectué à partir d'une simulation éléments finis à fort coût pour le paramètre $\boldsymbol{\mu}_i$ qui réalise : $\sup_{\boldsymbol{\mu}} \hat{\Delta}_i(\boldsymbol{\mu})$. On en déduit un champ de déplacement $\boldsymbol{u}_i = \boldsymbol{u}(\boldsymbol{\mu}_i)$. A la succession des snapshots des déplacements $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_i$ évalués en différents paramètres répond ainsi une succession d'espaces réduits emboîtés : $\hat{\Omega}_1 \subset \cdots \subset \hat{\Omega}_i$ à condition de se donner une méthode d'élaboration de base réduite.

Position du problème L'objet de [57] est la construction d'un modèle réduit et sa certification. En sollicitant le modèle réduit *préalablement élaboré*, on obtient *à faible coût* une réponse *et* une borne de l'écart entre la réponse du modèle réduit et celle du modèle réel qu'on aurait obtenue si l'on avait fait le calcul complet avec les mêmes entrées. Le type de problème complet abordé consiste en la détermination d'une sortie $\boldsymbol{s}(\boldsymbol{\mu}) = \boldsymbol{l}(\boldsymbol{u}(\boldsymbol{\mu}))$ où $\boldsymbol{u}(\boldsymbol{\mu})$ est solution de :

$$\boldsymbol{a}(\boldsymbol{u}(\boldsymbol{\mu}), \boldsymbol{v}; \boldsymbol{\mu}) = \boldsymbol{b}(\boldsymbol{v}; \boldsymbol{\mu}),\ \forall \boldsymbol{v} \in \boldsymbol{X} \tag{1.22}$$

On souhaite construire un modèle réduit de ce problème. Classiquement, il faut se donner un certain nombre de *snapshots* $\boldsymbol{u}(\boldsymbol{\mu})$ à partir desquels on construit un espace réduit en utilisant une POD ou une interpolation de Taylor, Lagrange, Hermite, etc. En faisant une projection de Galerkin sur cet espace, on en déduit les quantités réduites correspondantes : $\hat{\boldsymbol{s}} = \boldsymbol{l}(\boldsymbol{u}(\boldsymbol{\mu}))$ et $\hat{\boldsymbol{a}}(\boldsymbol{u}, \boldsymbol{v}; \boldsymbol{\mu}) = \hat{\boldsymbol{l}}(\boldsymbol{v}, \boldsymbol{\mu})$. La question qui vient alors est celle de savoir quel est l'écart entre la sortie réduite $\hat{\boldsymbol{u}}$ et la sortie réelle $\boldsymbol{u}$ et comment choisir les *snapshots* pour minimiser cette erreur, i. e. quelles valeurs de $\boldsymbol{\mu}$ sélectionner.

Techniques d'estimation d'erreur : la recherche d'un minorant L'objectif est d'encadrer l'erreur commise entre la sortie réduite $\hat{\boldsymbol{s}}$ et la sortie $\boldsymbol{s}$ du calcul complet. $\boldsymbol{u}$ et $\hat{\boldsymbol{u}}$ représente

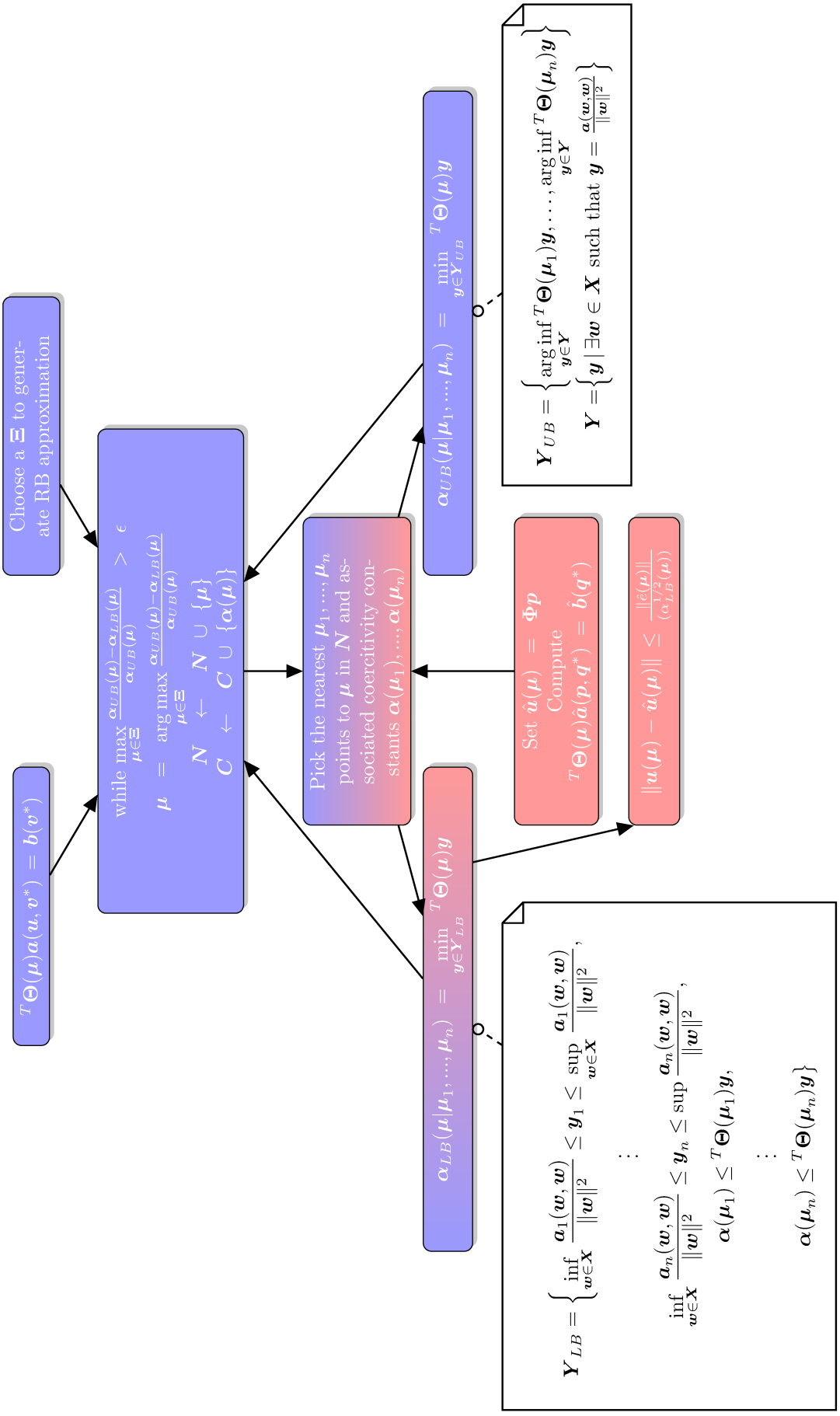


Figure 1.13 – Estimation d'erreur

respectivement l'état du système complet et du système réduit. $\hat{\boldsymbol{s}}(\boldsymbol{\mu}) = \hat{\boldsymbol{s}}^{-}(\boldsymbol{\mu}) \leq \boldsymbol{s}(\boldsymbol{\mu}) \leq \hat{\boldsymbol{s}}^{+}(\boldsymbol{\mu}) = \hat{\boldsymbol{s}}(\boldsymbol{\mu}) + \boldsymbol{\Delta}(\boldsymbol{\mu})$. Produire une estimation inférieure de la valeur de la sortie $\hat{\boldsymbol{u}}$ découle immédiatement de $\boldsymbol{s}(\boldsymbol{\mu}) - \hat{\boldsymbol{s}}(\boldsymbol{\mu}) = \boldsymbol{S}(\boldsymbol{u}(\boldsymbol{\mu}) - \hat{\boldsymbol{u}}(\boldsymbol{\mu})) = \boldsymbol{A}(\hat{\boldsymbol{u}}(\boldsymbol{\mu}) - \boldsymbol{u}(\boldsymbol{\mu}), \hat{\boldsymbol{u}}(\boldsymbol{\mu}) - \boldsymbol{u}(\boldsymbol{\mu})) \geq 0$. Mais il est plus difficile de trouver une borne supérieure. L'idée générale consiste à trouver une forme bilinéaire $\hat{\boldsymbol{a}}(\boldsymbol{u}, \boldsymbol{v}; \boldsymbol{\mu})$ qui est inférieure à la forme $\boldsymbol{a}(\boldsymbol{u}, \boldsymbol{v}; \boldsymbol{\mu})$ pour tout $\boldsymbol{u}$, $\boldsymbol{v}$ et $\boldsymbol{\mu}$. Dans [54] les auteurs proposent deux méthodes d'estimation de borne supérieure. La première est obtenue en adjoignant l'hypothèse $\boldsymbol{\alpha}\|\boldsymbol{v}\|^2 \leq \boldsymbol{g}(\boldsymbol{\mu})\boldsymbol{a}(\boldsymbol{v}, \boldsymbol{v}) \leq \boldsymbol{a}(\boldsymbol{v}, \boldsymbol{v}; \boldsymbol{\mu})$ ce qui revient à choisir $\hat{\boldsymbol{a}}(\boldsymbol{u}, \boldsymbol{v}; \boldsymbol{\mu}) = \boldsymbol{g}(\boldsymbol{\mu})\boldsymbol{a}(\boldsymbol{u}, \boldsymbol{v}; \boldsymbol{\mu})$. Une deuxième piste revient à affiner l'espace de résolution, i. e. raffiner le maillage. L'estimation d'une borne supérieure est alors possible en comparant la solution sur le maillage affiné à celle du maillage grossier. La forme $\hat{\boldsymbol{a}}(\boldsymbol{u}, \boldsymbol{v}; \boldsymbol{\mu})$ est celle correspondant au maillage grossier. Dans [65], on définit le résidu de la manière suivante : $\hat{\boldsymbol{e}}(\boldsymbol{\mu}) = \boldsymbol{b} - \boldsymbol{A}(\boldsymbol{\Theta}(\boldsymbol{\mu}))\hat{\boldsymbol{u}}(\boldsymbol{\mu})$. La sortie $\boldsymbol{s}(\boldsymbol{\mu})$ s'écrit : $\boldsymbol{s}(\boldsymbol{\mu}) = \boldsymbol{b} \cdot \boldsymbol{A}^{-}(\boldsymbol{\Theta}(\boldsymbol{\mu}))\boldsymbol{b}$. Ici la forme bilinéaire retenue est donnée par : $\hat{\boldsymbol{A}}(\boldsymbol{\mu}) = \left(\boldsymbol{\alpha}(\boldsymbol{\mu}) \cdot \boldsymbol{A}^{-1}(\boldsymbol{\theta})\right)^{-1}$ avec $\boldsymbol{\theta} = \sum \boldsymbol{\alpha}_j(\boldsymbol{\mu})\boldsymbol{\theta}_j$, $0 \leq \boldsymbol{\alpha}_j \leq 1$, $\sum \boldsymbol{\alpha}_j = 1$ et $\boldsymbol{\theta}(\boldsymbol{\mu}) \leq \boldsymbol{\Theta}(\boldsymbol{\mu})$. Un tel choix fournit une forme bilinaire minorante à $\boldsymbol{A}$ et permet donc de déduire une borne supérieure de l'écart entre la sortie réduite et complète.

Dans [57], au lieu de considérer de nouvelles formes bilinéaires minorant la forme étudiée pour donner une borne supérieure, on propose de donner une estimation de la constante de coercivité :

$$\boldsymbol{\alpha}(\boldsymbol{\mu}) = \inf_{w \in X} \frac{\boldsymbol{a}(\boldsymbol{w}, \boldsymbol{w}; \boldsymbol{\mu})}{\|\boldsymbol{w}\|_X^2} \tag{1.23}$$

Majoration de l'erreur La recherche d'une forme bilinéaire minorante se ramène à la recherche d'une borne inférieure à la constante de coercivité $\boldsymbol{\alpha}^{LB}(\boldsymbol{\mu})$ ainsi que le démontrent les auteurs dans [57] p. 262 où l'écart entre le modèle complet et le modèle réduit est donné par $\|\boldsymbol{u}(\boldsymbol{\mu}) - \hat{\boldsymbol{u}}(\boldsymbol{\mu})\| \leq \frac{\|\hat{\boldsymbol{e}}(\boldsymbol{\mu})\|^2}{\boldsymbol{\alpha}^{LB}(\boldsymbol{\mu})}$ L'avantage d'une telle méthode provient de ce qu'elle est systématique. Une méthode dite *successive Constraint Method* est présentée par les auteurs comme préférable à l'utilisation des bornes de Gershgorin ou aux méthodes d'approximation de Rayleigh Ritz pour calculer une borne inférieure à la constante de coercivité. Un algorithme de type Greedy est mis en place pour sélectionner les valeurs des paramètres $\boldsymbol{\mu}$ pour lesquelles on évalue la constante de coercivité α.

Choix des snapshots Dans [57] est exposée une méthode de construction de choix des *snapshots* fondée sur un calcul d'estimation d'erreur. La figure 1.13 en décrit le principe. Partant d'une fonctionnelle affine par rapport au paramètre $\boldsymbol{\mu}$ et en choisissant un ensemble $\boldsymbol{\Xi}$ de valeurs du paramètre $\boldsymbol{\mu}$, l'idée est de sélectionner les valeurs des paramètres $\boldsymbol{\mu}$ permettant de construire un modèle réduit minimisant l'erreur avec le modèle réel. L'erreur entre l'estimée réduite $\hat{\boldsymbol{u}}$ et le champ réel $\boldsymbol{u}$ peut être majorée en contrôlant la constante de coercivité $\boldsymbol{\alpha}$ de l'équation réduite. C'est pourquoi un algorithme de type *greedy* est mis en oeuvre pour choisir à chaque fois le $\boldsymbol{\mu}$ qui minimise l'erreur d'estimation. Une stratégie d'estimation d'un majorant $\boldsymbol{\alpha}_{UB}$ et d'un minorant $\boldsymbol{\alpha}_{LB}$ de $\boldsymbol{\alpha}$ connaissant les valeurs de la constante de coercivité aux points $\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_n$ est réalisée. Elle permet de déterminer à faible coût, i. e. indépendamment des dimensions du problème réel, une estimation de la valeur de la constante de coercivité. A chaque fois, l'algorithme *greedy* ajoute le point $\boldsymbol{\mu}$ pour lequel l'estimation est la moins pertinente.

Hypothèses restrictives et améliorations Néanmoins, la méthode exposée dans [57] impose des hypothèses restrictives. En particulier, afin de pouvoir distinguer une phase de calcul *offline* pendant laquelle sont élaborés le modèle réduit et l'estimateur d'erreur et une phase *online* efficace, la forme bilinéaire $\boldsymbol{a}(\cdot\,,\cdot\,;\boldsymbol{\mu})$ doit être affine en $\boldsymbol{\mu}$: $\boldsymbol{a}(\cdot\,,\cdot\,;\boldsymbol{\mu}) = {}^T\boldsymbol{\Theta}(\boldsymbol{\mu})\boldsymbol{a}(\cdot\,,\cdot\,)$ ainsi que $\boldsymbol{b}$: $\boldsymbol{b}(\cdot\,;\boldsymbol{\mu}) = {}^T\boldsymbol{\Theta}(\boldsymbol{\mu})\boldsymbol{b}(\cdot\,)$. Des recherches ont été conduites pour se ramener à cette forme de fonctionnelles. Par exemple dans [43], les auteurs appliquent ces techniques de réduction de modèle à une équation parabolique de convection-diffusion, dans un domaine Ω parallélépipédique :

$$\frac{\partial}{\partial t}y(x,t;\boldsymbol{\mu}) + \underbrace{\boldsymbol{v}\nabla y(x,t;\boldsymbol{\mu})}_{\text{convection}} - \underbrace{\kappa\nabla^2 y(x,t;\boldsymbol{\mu})}_{\text{diffusion}} = \underbrace{q(x,\boldsymbol{\mu})u(t)}_{\text{terme source}} \tag{1.24}$$

où $\boldsymbol{v}$ est la vitesse, u l'intensité de la source, κ le coefficient de diffusivité thermique. Le champ de vitesse est imposé constant dans l'ensemble de la structure : $\boldsymbol{v} = (v_x, 0, 0)$. Le terme source est décrit par $\boldsymbol{q}(\boldsymbol{x},\boldsymbol{\mu}) = e^{-\frac{x_1^2}{\sigma_1^2}} e^{-\frac{x_2^2}{\sigma_2^2}} e^{-\frac{x_3^2}{\sigma_3^2}}$. Ainsi, dans ce cas, le paramètre d'intérêt vaut : $\boldsymbol{\mu} = (\sigma_1, \sigma_2, \sigma_3, \kappa)$. La sortie d'intérêt est la moyenne, à un instant donné t, pour un paramètre donné $\boldsymbol{\mu}$, du champ scalaire y sur $\Omega_m \subset \Omega$: $l(y) = \frac{1}{|\Omega_m|}\int_{\Omega_m} y(t,\boldsymbol{\mu})\,\mathrm{d}x$. Le terme source de l'équation 1.24 n'est pas affine en $\boldsymbol{\mu}$ ce qui empêche d'appliquer directement la méthode de la figure 1.13. La solution imaginée par les auteurs est d'utiliser l'EIM pour fournir une approximation affine de $\boldsymbol{q}$, le terme source. Un autre exemple d'application est donné dans [64] où la méthode développée est appliquée dans le champ de la mécanique des fluides à l'équation de Von Karman non-linéaire pour un fluide visqueux incompressible : $\boldsymbol{g}(\boldsymbol{w},\boldsymbol{v};Gr) = \boldsymbol{a}_0(\boldsymbol{w},\boldsymbol{v}) + \frac{1}{2}\boldsymbol{a}_1(\boldsymbol{w},\boldsymbol{w},\boldsymbol{v}) - GrF(\boldsymbol{v})$

1.12 Méthodes de réduction appliquée à l'analyse de VER

Mise en place d'un estimateur d'erreur Dans [41] sont présentés la construction d'un modèle réduit et d'un estimateur d'erreur. L'erreur $\boldsymbol{e}(\boldsymbol{\mu}) = \boldsymbol{u}^h(\boldsymbol{\mu}) - \boldsymbol{u}^r(\boldsymbol{\mu})$ est encadrée en norme par une borne sup $\nu^{sup}(\boldsymbol{\mu})$ et une borne inf $\nu^{inf}(\boldsymbol{\mu})$. Il est démontré que :

- $\nu^{sup}(\boldsymbol{\mu}) = \|\boldsymbol{\sigma}^r(\boldsymbol{\mu}) - \hat{\boldsymbol{\sigma}}(\boldsymbol{\mu})\|_{K(\boldsymbol{\mu})} \leq \|\boldsymbol{e}^r(\boldsymbol{\mu})\|_{K(\boldsymbol{\mu})}$ où $\boldsymbol{\sigma}^r(\boldsymbol{\mu}) = \boldsymbol{D}(\boldsymbol{\mu}) : \boldsymbol{\varepsilon}(\boldsymbol{u}^r)(\boldsymbol{\mu})$ (qui ne vérifie pas l'équilibre). Afin d'obtenir la plus petite borne supérieure possible, on choisit $\hat{\boldsymbol{\sigma}}(\boldsymbol{\mu}) = \arg\min_{\boldsymbol{\sigma}^* \in \mathcal{S}_r} \|\boldsymbol{\sigma}^* - \boldsymbol{\sigma}^h(\boldsymbol{\mu})\|_{K(\boldsymbol{\mu})}$. $\mathcal{S}_r$ désigne l'espace réduit des contraintes statiquement admissibles. Ce problème se ramène à un problème variationnel et peut être résolu en deux temps. Lors d'une phase de calcul *offline*, la matrice de rigidité réduite est pré-calculée en projetant la matrice de rigidité complète sur l'espace réduit des contraintes ainsi que le vecteur des réactions internes. Ensuite, le problème réduit assemblé peut être résolu à faible coût lors d'un phase de calcul *online*.
- $\nu^{inf}(\boldsymbol{\mu}) = \frac{R(\boldsymbol{e}^r,\boldsymbol{\mu})}{\|\boldsymbol{u}^{2r} - \boldsymbol{u}^r\|_{\boldsymbol{D}(\boldsymbol{\mu})}} \leq \|\boldsymbol{e}^r(\boldsymbol{\mu})\|_{\boldsymbol{D}(\boldsymbol{\mu})}$ où $\boldsymbol{R}(\boldsymbol{e}^r,\boldsymbol{\mu}) = \boldsymbol{l}(\boldsymbol{e}^r,\boldsymbol{u}) - \boldsymbol{a}(\boldsymbol{u}^r,\boldsymbol{e}^r,\boldsymbol{\mu})$ avec $\boldsymbol{e}^r = \boldsymbol{u}^{2r} - \boldsymbol{u}^r$. $\boldsymbol{u}^{2r}$ est évalué à partir d'un second modèle réduit de l'espace des déplacements plus riche que le premier modèle réduit. Ainsi le champ de déplacement réduit obtenu $\boldsymbol{u}^{2r}$ est plus proche du champ de déplacement éléments finis réel $\boldsymbol{u}$ que le champ $\boldsymbol{u}^r$. C'est de cette proximité que l'on arrive à déduire une borne expression de la borne inférieure. Là encore, il est possible de séparer une étape de calcul *online*

d'une étape de calcul *offline* sous l'hypothèse d'une variation affine des équations du modèles par rapport aux paramètres.

Application L'application à une étude paramétrique portant sur un composite fibre-matrice 2D où la position et le diamètre sont fixés aléatoirement est ensuite abordée. Les équations du modèle dépendent de manière affine de 4 paramètres qui représentent la fraction de phase de fibre, les deux tractions et le cisaillement. La dépendance affine des équations du modèle en les paramètres permet de distinguer une phase de calcul *online* peu coûteuse d'une phase de calcul *offline* beaucoup plus coûteuse pendant laquelle le modèle réduit est construit. L'absence de dépendence affine en les paramètres obligerait à se tourner vers d'autres formes de réduction de modèle, notamment l'hyperréduction, afin de diminuer le coût d'assemblage de la matrice de rigidité et le calcul des réactions internes. La loi de comportement retenue est élastique linéaire aussi bien pour la fibre que pour la matrice. L'application de cette méthode d'estimation à des lois non-linéaires n'a rien de trivial et est à ce jour un problème largement ouvert. Les conditions aux limites choisies pour solliciter le volume élémentaire représentatif sont les conditions homogènes au contour en déplacement. De telles conditions permettent d'éviter les difficultés d'application des méthodes d'hyperréduction de modèle, même si le comportement homogène équivalent macroscopique obtenu est généralement systématiquement trop rigide et si des conditions aux limites périodiques au contour permettraient de modéliser un comportement homogène équivalent moins rigide. Lors de l'étape de calcul *offline*, pour chaque valeur de paramètre $\boldsymbol{\mu}$, le VER est sollicité. Ces tests permettent d'obtenir :

- un espace réduit de déplacements correspondant à l'application de la POD aux champs de déplacement obtenus à chaque fois privé de leur valeur au bord
- un espace réduit de contraintes correspondant à l'application de la POD sur les champs de contraintes prélevés dans chacun des cas. Cet espace est constitué de champs de contraintes statiquement admissibles.

Dans [51], les auteurs appliquent à l'échelle micro d'un volume élémentaire représentatif une méthode de réduction de modèle. La cellule est sollicitée par des chargements macroscopiques en conditions dites homogènes au contour. Un des constituants micro est endommageable. Les bases réduites sont construites par *snapshot POD*. On opère une SVD sur la matrice des *snapshots*. Le modèle est constitué d'une base réduite pour le terme de fluctuations microscopiques $\boldsymbol{\Phi}$. Les auteurs utilisent une méthode *gappy* consistant à évaluer les forces intérieures sur une partie du domaine d'intégration. Une base réduite $\boldsymbol{\Psi}$ leur est dédiée. $\boldsymbol{\Phi}$ et $\boldsymbol{\Psi}$ sont construites à partir des mêmes snapshots. La sélection des éléments du domaine réduit est opérée par la méthode DEIM. Les bases des déplacements et des forces intérieures sont calculées *offline* et sont réutilisées dans l'assemblage et la résolution du modèle réduit *online*. Ainsi on distingue deux degrés d'approximation : l'approximation des fluctuations et l'approximation des forces intérieures. Les facteurs d'accélérations affichés sont de l'ordre de dix. Le nombre de vecteurs engendrant les espaces réduits est également de l'ordre d'une dizaine. Ces caractéristiques sont très proches de celles que l'on peut relever en hyperréduction de modèle.

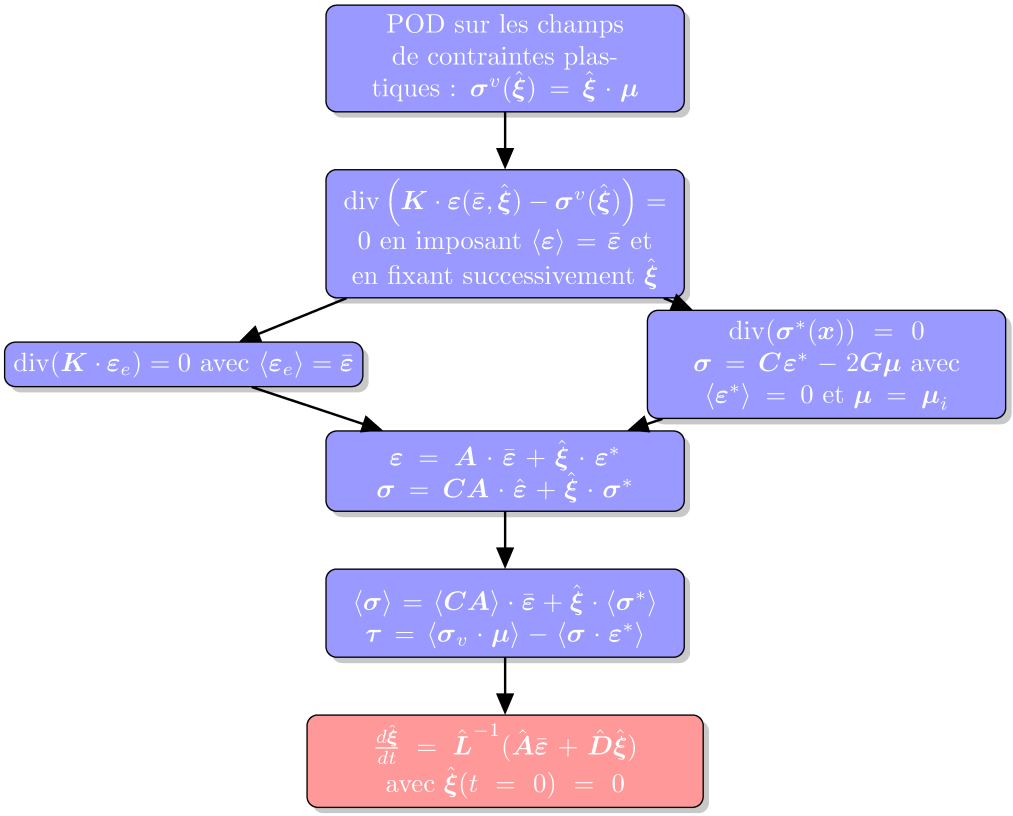


Figure 1.14 – Description de la NTFA mise en oeuvre dans [32].

1.13 La famille des méthodes NTFA

Dans [49] est développée une technique d'homogénéisation des composites à matériau standard appelée NTFA, i. e. *Non uniform transformation field analysis*. L'objectif de la méthode, tel que présentée dans [49], est de pouvoir représenter un comportement homogène équivalent standard par une loi matériau possédant seulement un faible nombre de variables internes. La NTFA s'intéresse à la résolution du problème de type suivant pour un champ fixé $\boldsymbol{\varepsilon}^{an}$ de contraintes internes : $\boldsymbol{\sigma} = \boldsymbol{L} : (\boldsymbol{\varepsilon}(\boldsymbol{x}) - \boldsymbol{\varepsilon}^{an}(\boldsymbol{x}))$, $\text{div}(\boldsymbol{\sigma}) = 0$, $\langle \boldsymbol{\varepsilon} \rangle = \hat{\boldsymbol{\varepsilon}}$. Après quelques développements [49], on trouve que ce problème admet la solution suivante : $\boldsymbol{\varepsilon} = \boldsymbol{A} \cdot \hat{\boldsymbol{\varepsilon}} + \boldsymbol{D} \cdot \boldsymbol{\varepsilon}^{an}(\boldsymbol{x})$. C'est cette expression que l'on cherche à évaluer pour mettre en oeuvre la méthode. On la retrouve sur la figure 1.14 à l'étape 5. Le calcul de l'opérateur de localisation linéaire $\boldsymbol{A}$, qui permet de passer d'une déformation macroscopique $\hat{\boldsymbol{\varepsilon}}$ à un champ de déformation microscopique, est simple : il se ramène à sa connaissance sur une base de sollicitation macroscopique (fig. 1.14 4ème étape gauche). L'évaluation de l'opérateur $\boldsymbol{D}$ est quant à elle plus délicate. Dans [24], l'opérateur est calculé sur un ensemble de base constante par phase $\boldsymbol{\varepsilon} = \hat{\boldsymbol{\xi}} \cdot \boldsymbol{\chi}$ où $\boldsymbol{\chi}$ est constant par sous-domaine.

Dans [49], l'opérateur est calculé sur un ensemble de mode non nécessairement constant par phase, $\boldsymbol{\varepsilon} = \hat{\boldsymbol{\xi}} \cdot \boldsymbol{\mu}(\boldsymbol{x})$, ainsi qu'illustré figure 1.14 à l'étape 2. Les variables internes du modèle sont alors constituées des $\hat{\boldsymbol{\xi}}$, coefficients d'activité de chaque mode plastique. Dans

[56], la POD est utilisée pour synthétiser les modes à partir des déformations plastiques $\boldsymbol{\varepsilon}^p$. Afin d'écrire la loi homogène équivalente macroscopique, un choix de variables internes s'impose. Dans [49], les variables internes sont choisies comme les projetés des variables internes micro sur les modes de déformation. Dans [32], les seules variables internes retenues sont les composantes de $\hat{\boldsymbol{\xi}}$, i. e. les coefficients d'activité eux-mêmes. Les variables internes duales de $\hat{\boldsymbol{\xi}}$ sont les $\boldsymbol{\tau}$. Une loi macroscopique homogène équivalente est alors déduite dans la dernière étape de la figure 1.14.

Dans [33], les auteurs proposent une implémentation 3D de la NTFA (exploitée jusqu'alors dans le cas 2D). Le passage de la 2D à la 3D pose des problèmes de nature algorithme et théorique, en particulier en ce qui concerne le calcul des modes propres et la constitution des bases réduites. C'est pourquoi les auteurs s'interrogent sur une procédure d'identification des modes propres, essentielle pour construire un modèle de comportement NTFA 3D de la microstructure. Ils rappellent la méthode classique de construction de snapshots, celle de Karhunen-Loève dont ils relèvent quelques limites concernant leur cas d'application :

- *son coût.* Elle nécessite le calcul de nombreux produits scalaires de vecteurs Ω. La formation de nouveau champ représenté par les vecteurs de la base réduite s'effectue par des combinaisons linéaires des champs existant. Quand le nombre de points d'intégration et de snapshots s'accroissent, le temps de calcul devient considérable ;
- *sa précision.* La matrice $\boldsymbol{S}$ peut être mal conditionnée. La recombinaison d'un grand nombre de snapshots en base réduite peut donner naissance à des problèmes de troncature ;
- *sa capacité à représenter les états de la microstructure.* Appliquée au champ de déformation plastique, elle conduit à une surreprésentation des derniers états temporels, là où on enregistre la plascité la plus forte.

Ils présentent une méthode alternative. Cette méthode ne repose pas sur une SVD mais sur un procédé d'orthogonalisation de Schmidt.

1.14 Formulation du problème mécanique EF^2

Les méthodes EF^2 appartiennent à la catégorie des méthodes d'homogénéisation numérique. Présentées dans [29], appliquées à l'analyse des effets de taille dans [28] et à l'étude de l'endommagement de structures composites dans [31], elles permettent de prendre en compte les effets microsctruturaux à l'échelle macroscopique. Le principe d'un calcul EF^2 est décrit sur la figure 1.15. A l'échelle macroscopique, la structure est sollicitée. Dans un calcul EF classique, en chaque point de Gauss, la loi de comportement est intégrée afin de pouvoir assembler la matrice de rigidité et déterminer la réponse de la macro-structure. Dans le cas d'un calcul EF^2, en chaque point de Gauss, on sollicite un volume élémentaire représentatif (VER). Différentes conditions aux limites peuvent être utilisées : des conditions homogènes au contour, des conditions périodiques dont on fera usage. Le déplacement des cellules périodiques est décomposé en $\boldsymbol{u} = \boldsymbol{E} \cdot \boldsymbol{x} + \boldsymbol{v}$ où $\boldsymbol{E}$ est la déformation macroscopique et $\boldsymbol{v}$ le terme de fluctuation périodique. On impose que ce dernier soit périodique : $\boldsymbol{v}(\boldsymbol{x}) = \boldsymbol{v}(\boldsymbol{y})$ sur les points frontière correspondant du VER périodique. Sur le champ de contraintes

normales $\boldsymbol{\sigma} \cdot \boldsymbol{n}$ en bord de VER doivent être imposées des conditions d'anti-périodicité : $\boldsymbol{\sigma}(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) = -\boldsymbol{\sigma}(\boldsymbol{y}) \cdot \boldsymbol{n}(\boldsymbol{y})$ (fig. 1.15).

L'intégration de la loi de comportement est ainsi rejetée à l'échelle microscopique. Au lieu d'une simple intégration de la loi de comportement comme le requiert la méthode NTFA (cf. 1.13) une fois la loi de comportement élaborée (fig. 1.14) ou les méthodes de calcul de loi macroscopique homogène équivalente standard, un calcul éléments finis sur cellule élémentaire est ainsi réalisé (fig. 1.15). Cela conduit à une multiplication des variables internes et à des modèles numériques très coûteux. L'avantage résidant dans le fait qu'il n'est pas besoin de définir à l'échelle macroscopique une loi de comportement. En effet, choisir une loi macroscopique homogène équivalente représentative de phénomènes complexes d'endommagement de matériaux ou de décohésion peut s'avérer délicat. La contrepartie est un modèle très coûteux à résoudre mais facilement parallélisable. Les problèmes microscopiques peuvent se traiter de manière tout à fait indépendante. Néanmoins, les efforts de parallélisation ne sont pas suffisants pour traiter des cas industriels [29] p. 224.

1.15 Conclusion

Dans ce chapitre, une série de méthodes de réduction de modèle a été présentée. Elles ont été comparées. On peut les classifier en deux niveaux : des méthodes élémentaires comme la POD (cf. 1.3) ou la DEIM (cf. 1.5) ou encore la MPE (cf. 1.6) sont impliquées dans l'élaboration de méthodes de réduction plus complexes comme la GNAT (cf. 1.9) ou l'hyperréduction (cf. 1.10).

On peut aussi classifier ces approches de réduction de modèle selon qu'elles s'intéressent à un système de grande taille comme la PGD (cf. 1.7) où l'ensemble des paramètres est intégré dans les équations et résolu ; ou, au contraire, la NTFA (cf. 1.13) qui s'intéresse à une seule cellule élémentaire pour élaborer une loi numérique homogène équivalente qui sera utilisée dans un calcul de macrostructure.

Enfin on peut répartir ces méthodes en deux dernières catégories : les méthodes où la phase de calcul offline et online se mélangent, par exemple, la PGD utilisée en mode solveur ou l'hyperréduction adaptative de celles où les phases de calcul sont nettement séparées comme la NTFA.

Une approche type hyperréduction de modèle a été choisie dans cette thèse afin de réduire les calculs EF^2. L'inconvénient d'une telle méthode est qu'elle nécessite un calcul complet de snapshots ce qui impose de résoudre des calculs EF^2 massifs contrairement à la PGD qui permet de construire un snapshot à la volée. L'avantage est qu'elle ne nécessite en mode online que de résoudre l'équilibre mécanique sur une partie de la structure. Enfin, une implémentation de la méthode d'hyperréduction avait déjà été réalisée dans le code de calcul ZSET.

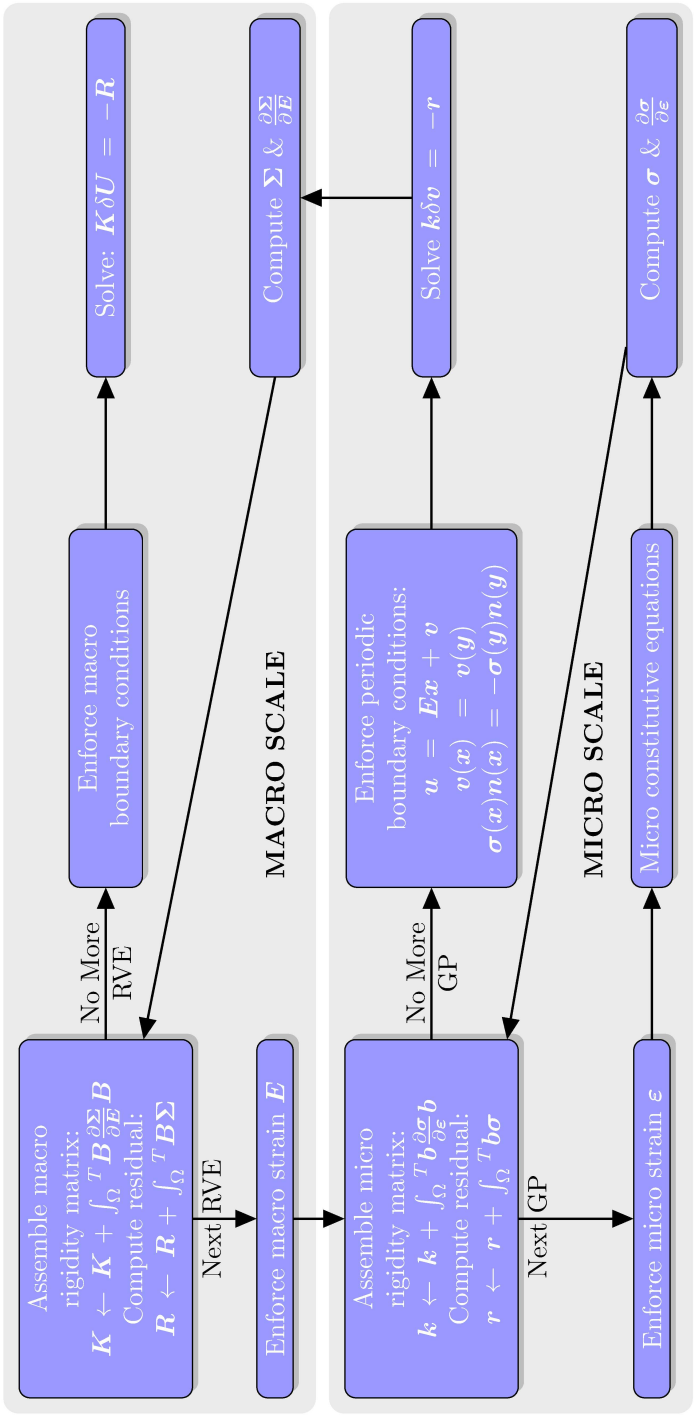


Figure 1.15 – Formulation d'un problème EF^2

2

Eléments d'architecture logicielle

L'objectif que nous poursuivons est la conception d'un plugin modulaire d'hyperréduction de modèle pour un code éléments finis. Programmer n'est pas une tâche simple. L'objectif de ce chapitre n'est certainement pas d'expliquer profondément comment bien programmer. De nombreux auteurs se sont attelés à cette matière difficile qui requiert habileté et expérience. Ce chapitre a pour vocation de présenter les difficultés qui surgissent quand on est confronté à un tel objectif et de sensibiliser le lecteur aux moyens d'y faire face, à certains concepts de programmation qui seront fréquemment utilisés dans la suite de ce travail et qui sont trop souvent négligés, si bien que leur présentation apparaît comme un prérequis indispensable à la compréhension. Ce chapitre peut être vu comme une boîte à outils à laquelle nous nous réfèrerons et comme une brève introduction aux techniques qui ont été utilisées ici et qu'il est nécessaire d'acquérir pour se lancer dans l'amélioration et dans la poursuite du développement de ce plugin.

Ce chapitre se développe en trois axes :

- *Une présentation de certains schémas de conception. Les schémas de conception ne sont qu'un recensement des façons de penser l'architecture d'un code. En effet les mêmes besoins reviennent sans cesse lors de l'activité de développement. L'expérience d'un développeur consiste en sa capacité à localiser et isoler ces schémas de conception, à rattacher une situation en apparence nouvelle à de l'existant. Bien assimiler ces schémas pour savoir les appliquer à bon escient est très certainement une pierre angulaire du succès du développement. Le champ d'application ne se limite ni à celui de développement de logiciels pour la mécanique numérique, ni à celui d'ajout de module à un code éléments finis, ni même à un langage de programmation. La connaissance de ces schémas est utile dès que l'on souhaite programmer quoi que ce soit. C'est pourquoi, il est nécessaire d'aborder ce point. Ainsi, après une présentation très générale de l'ensemble de ces schémas, nous proposons une description détaillée des schémas que nous avons employés dans le plugin d'hyperréduction. Ces techniques permettent de définir des objets, d'isoler des tâches, de structurer le développement à un haut niveau de granularité. Cela constitue une première étape certes nécessaire mais insuffisante.*
- *Il convient également de prêter une attention particulière à un niveau de granularité inférieure. Les méthodes de réduction de modèle, en particulier l'hyperréduction de modèle, requiert l'utilisation de modèles informatiques très importants. Dès lors factoriser les fonctionnalités de bas niveau devient primordial pour assurer la maintenabilité et les développements futures d'une architecture logicielle. Faire fi d'une telle problématique*

revient à se priver d'un développement collaboratif et, à terme, à se laisser envahir par une multitude de lignes de code qui échappent à toute compréhension, ainsi que souligné dans [9]. Pour répondre à ce besoin, des techniques de métaprogrammation vont être mises en oeuvre dans ce travail. Le langage cible, le C++, a développé depuis maintenant près d'une vingtaine d'années une série de moyens de métaprogrammation avec des patrons dont nous ferons abondamment usage. Même si ces techniques ne sont utilisées dans le code hôte ZSET que de façon embryonnaire, nous estimons que ce n'est plus sacrifier à la portabilité que d'introduire aujourd'hui ces techniques matures dans ce logiciel commercial.

- *Enfin, avant même de se lancer dans la conception, il est primordial de bien connaître l'architecture d'un code éléments finis. Cette architecture fait globalement consensus. Nous la présenterons dans ces grandes lignes afin de préciser l'endroit où l'on interviendra pour accrocher le plugin. L'idée est d'utiliser cette architecture, sans en rien modifier, pour faire de l'hyperréduction.*

Sommaire

2.1 Définitions

Objets La programmation orientée objet permet d'isoler les données à l'intérieur d'espaces de nommage. Ces espaces constituent des classes qui représentent des objets et les données rendues inaccessibles - on dit aussi encapsulées - sont appelées attributs. La seule manière de modifier un objet est d'agir grâce à des fonctions qui lui appartiennent - les méthodes. Les objets sont créés en instanciant une classe. Chaque objet a un cycle de vie : création, initialisation, utilisation et destruction. En particulier il possède un état interne qui doit être contrôlé par l'encapsulation la plus stricte.

Interface De la même façon que les données sont systématiquement enfouies au sein d'objets, la manipulation des objets s'effectue au travers d'interfaces. Un effort particulier doit être consenti au début pour conceptualiser une interface suffisamment abstraite qui doit conférer au projet de développement sa stabilité.

C++ Le langage C++ permet de faire de la programmation orientée objet. Mais il permet bien plus. En effet le C++ est le regroupement des quatre sous-langages suivants :

- C
- C++ orienté objet (*Object-Oriented C++*)
- C++ des patrons et de la metaprogrammation (*Template C++*)
- l'approche par la librairie standard (*STL*)

Liens entre objets Sur la figure 2.1 la flèche pointillée représente l'instantiation de `class INSTANTIATEE` par `class INSTANTIATOR`. L'héritage est le mécanisme que l'on utilise quand on souhaite faire partager les propriétés d'une classe par ces différentes spécialisations. On le représente couramment par une flèche ouverte pointant vers la classe mère (fig. 2.1). Ainsi on dit que `class SUBCLASS` hérite de `class PARENT_CLASS` ou encore que `class SUBCLASS` est une spécialisation de `class PARENT_CLASS`. Il est possible d'implémenter une redéfinition d'une méthode dans les classes spécialisées par le biais de méthodes virtuelles : on appelle cela l'*overriding* qui ne doit pas être confondu avec l'*overloading*, la surcharge, qui offre la possibilité de choisir entre de multiples implémentations d'une même fonction ou méthode selon le nombre et le type d'arguments. L'héritage associé à l'utilisation de méthodes virtuelles permet de contrôler l'état d'une instance de la classe spécialisée, à travers les méthodes de la classe mère, de manière tout à fait transparente pour le client. A ce titre, il constitue une forme de polymorphisme. Par certains aspects extrêmement pratique - notamment comme moyen de factorisation -, l'héritage n'est pas une panacée et conduit à fragiliser le logiciel car il ne respecte pas l'encapsulation. Pour une discussion plus détaillée sur les limites de cette notion, nous renvoyons à [14], Item 16, p. 81 - p. 86. L'auteur de [14] fait preuve de la plus grande circonspection vis-à-vis de cette technique, intitulant l'Item 17 (p. 87 - p. 92) de son livre : "Design and abstractument for inheritance or else prohibit it". Dépassant le cadre de cette introduction, nous n'en recommandons pas moins la consultation et l'étude de ces deux derniers points qui illustrent la manière dont on doit procéder avec l'héritage et des techniques qui permettent de contourner ses limites. La composition représentée par une flèche pleine

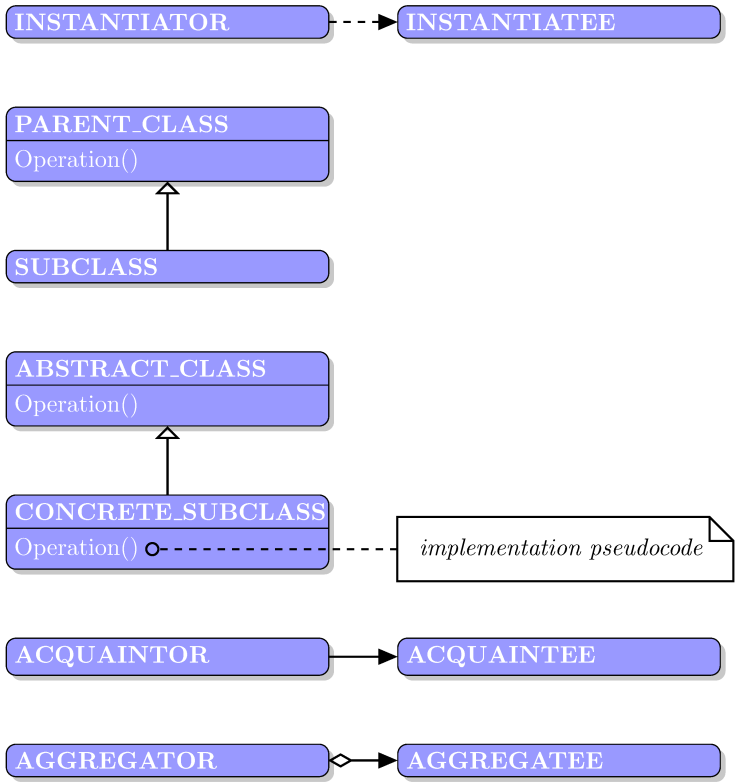


Figure 2.1 – Notations

à base losangique (fig. 2.1) indique une relation d'appartenance tandis que l'agrégation permet de définir une liaison entre `class ACQUAINTOR` et `class ACQUAINTEE` sans appartenance. Ainsi, non seulement l'instance de `class AGGREGATOR` possède un pointeur vers une instance de `class AGGREGATEE` comme `class ACQUAINTOR` vers `class ACQUAINTEE`, mais de surcroît le cycle de vie de l'objet pointé ne peut excéder celui de l'instance de `class AGGREGATOR`, à la différence du couple `class ACQUAINTOR-class ACQUAINTEE`. Enfin, il n'est pas rare d'adopter dans la littérature une flèche suivie d'un rond pour exprimer l'agrégation ou la composition avec de multiples instances du même type.

2.2 Haut niveau : les schémas de programmation

Les relations que les objets peuvent entretenir entre eux sont en nombre limité et constituent des schémas de conception ou de programmation. On a extrait de [34] les schémas de programmation que l'on a utilisés dans la suite de ce travail. Pour la consistance, nous commentons brièvement chacun d'entre eux, en notant les points saillants qui nous sont apparus lors de la programmation.

Il est courant de subdiviser les schémas de conception en trois familles caractérisées par leur fonctionnalité :

- les schémas de création. Ils fournissent des fonctionnalités permettant d'isoler la

création d'objet sans en connaître le type exact ou, plus généralement, de contrôler la création. C'est pourquoi ils conduisent à un gain en flexibilité d'usage. Ainsi en est-il de l'*Abstract Factory* dont le rôle est de créer une famille d'instance d'objets compatibles, du *Builder* qui permet de diviser la construction d'un objet complexe en menus morceaux, de la *Factory Method* qui permet de créer un objet sans avoir accès à son type exact, du *Prototype* qui permet de fournir, quand invoqué, une copie d'une instance de référence et du *Singleton* qui permet de s'assurer de la création d'une unique instance de l'objet considéré.

- les schémas de structure. Ils offrent des moyens de gérer des agrégats d'objets. Parmi eux, le plus utilisé, est sans doute le schéma *Composite*. Une multitude d'autres schémas présentés dans 2.2 en découle moyennant menues modifications.
- les schémas de comportement. Ils définissent la manière dont les objets interagissent. Un des plus simples et des plus utiles est peut-être le schéma de conception appelé *Strategy* qui permet d'infléchir le comportement d'un objet en le définissant tout ou partie dans un autre nouvel objet.

Composite L'objet composite 2.3 permet de manipuler des objets ou des agrégats d'objet uniformément, i. e. de façon invisible pour le client. Les opérations que l'on souhaite effectuer sur une catégorie d'objets, les objets élémentaires ou *leaf* sont commandées en appelant simplement la méthode `Operation()` de *composite*. Le *component* fait office d'interface client. Par son intermédiaire, le client peut ajouter ou soustraire d'autres éléments élémentaires ou composés.

Singleton Le singleton (fig. 2.4) permet de garantir l'existence d'une unique instance d'une classe. Il est fréquemment utilisé pour fournir une représentation virtuelle de ressources ou de matériel comme le clavier, l'écran... il existe de nombreuses implémentations du singleton. Comme le fait remarquer [2] p. 129, peu sont pleinement satisfaisantes en dépit de l'apparente simplicité de l'objet. Le singleton est responsable de son cycle de vie. Il doit être capable de se créer et de se détruire à la fin de l'exécution du programme. Une des façons les plus courantes d'implémenter un singleton consiste à utiliser une méthode statique comme en témoigne le programme 2.1. Le constructeur par défaut et le constructeur de copie sont marqués privés ce qui empêche toute création en dehors de la méthode `Instance`

```
class Singleton                                                    1
{                                                                  2
public:                                                            3
 static Singleton* Instance() // Unique point of access            4
 {                                                                 5
    if (!pInstance)                                                6
      pInstance_ = new Singleton;                                  7
    return pInstance_;                                             8
 }                                                                 9
                                                                   10
private:                                                           11
                                                                   12
 Singleton(); // Prevent client from creating a new Singleton      13
 Singleton(const Singleton&) // Prevent client from creating       14
                             // a copy of Singleton                15
```

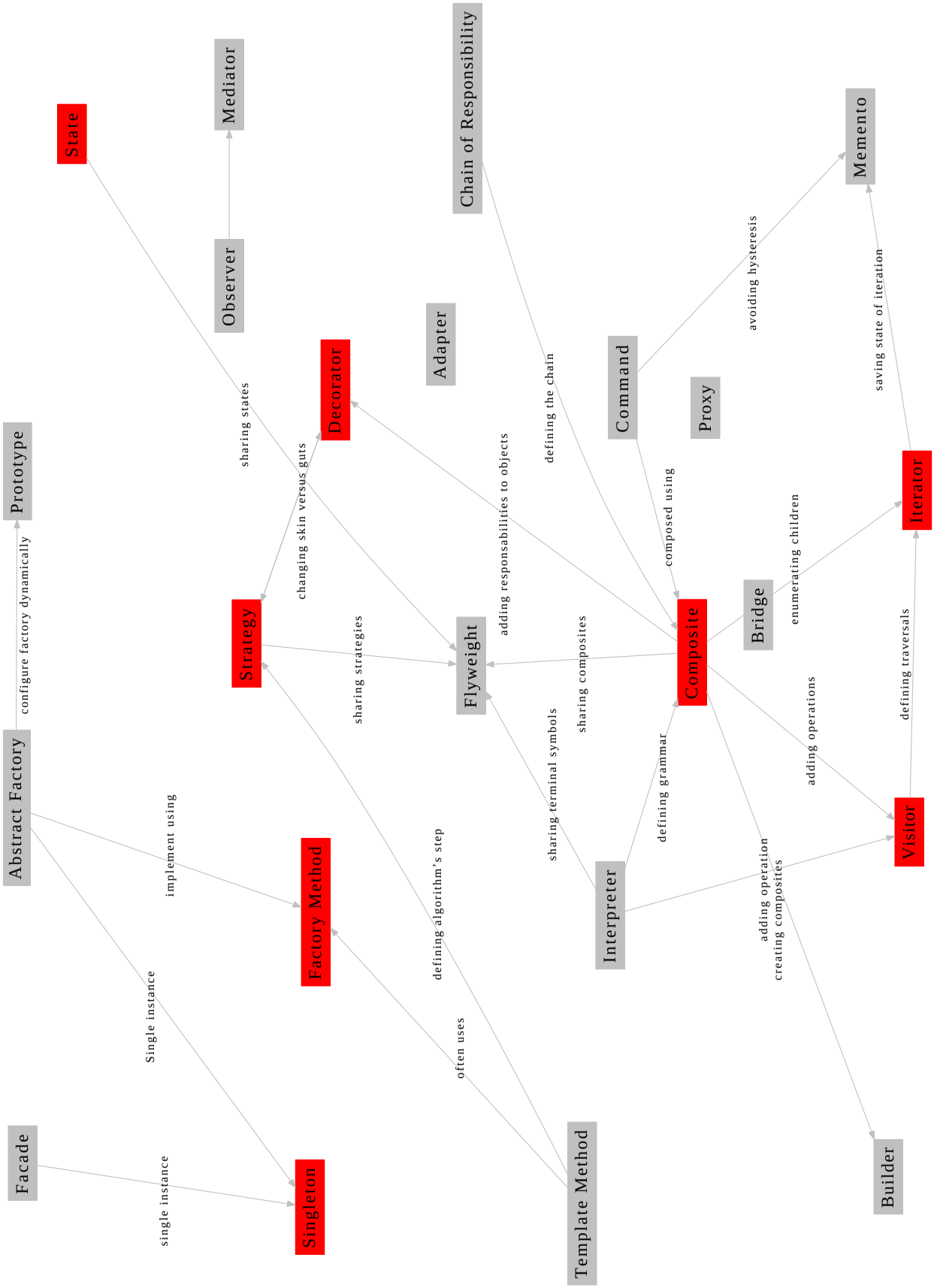


Figure 2.2 – Liens entre les schémas de conception.

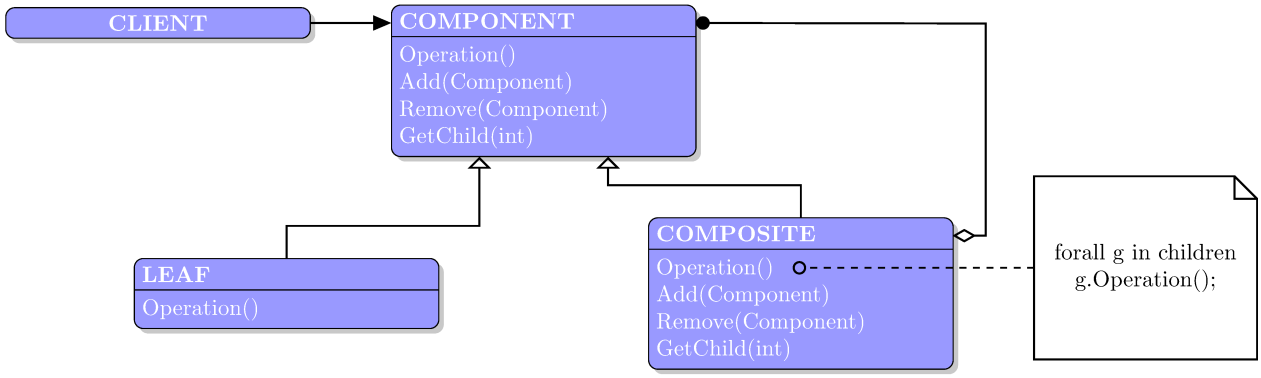


Figure 2.3 – Une structure composite

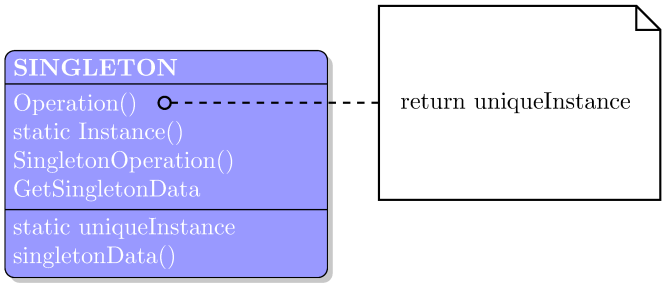


Figure 2.4 – Schéma d'une classe Singleton

```
  static Singleton* pInstance_; // The one and only instance
                                                                16
                                                                17
                                                                18
  };                                                            19
                                                                20
  // Implementation file Singleton.cpp                          21
  Singleton* Singleton::pInstance_ = 0;                         22
                                                                23
};                                                              24
```

Listing 2.1 – Mise en oeuvre du Singleton

Ainsi que le fait remarquer [2], [34] explique comment s'assurer de l'unicité du singleton mais jamais comment procéder à sa destruction. Certes la mémoire allouée par le singleton est automatiquement délivrée au moment de la fin d'exécution de l'application par les systèmes d'exploitation modernes. C'est pourquoi, on ne peut parler de fuite mémoire. Mais si le singleton se met à acquérir des ressources système comme des connexions réseaux, moyens de communication entre processus, ceux-là même qui ne sont pas automatiquement délivrés à la fin de l'exécution, il y a fuite de ressources. Différentes propositions d'amélioration de mise en oeuvre du singleton ont été proposées afin de remédier à ce problème : le singleton de Meyers, le singleton phénix, etc... En ce qui concerne ce travail, même si nous nous sommes reposés sur l'implémentation classique du singleton, nous nous sommes assurés de l'absence de fuite de cette nature.

State Le schéma de conception d'état dont le diagramme de classe est donné figure 2.5 permet de mimer le fonctionnement d'une machine d'état. Les objets se comportent

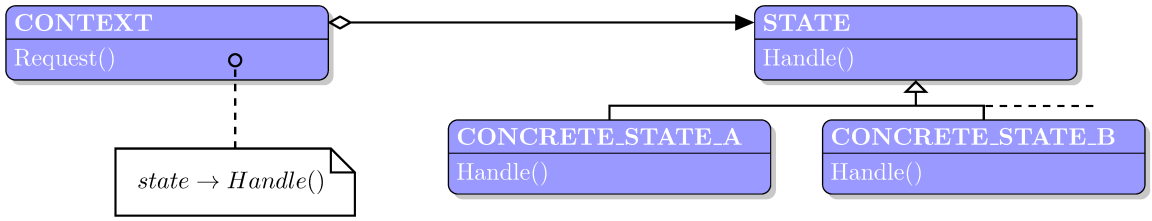


Figure 2.5 – Schéma de conception d'état

différemment en fonction de l'état du système. Chaque état est un singleton (fig. 2.4). Le `CONTEXT` possède un pointeur sur une des spécialisations de `STATE`. En fonction de l'état imposé par le client, le comportement varie. A ce titre, ce schéma de conception offre une parenté avec le schéma de conception *strategy*.

Decorator Parfois, les schémas de conception sont tant utilisés que les concepteurs de langage les intégrent dans la grammaire. Ainsi, le schéma de conception "decorator" a été intégré dans la syntaxe de python qui lui réserve le symbole @ précédent le nom d'une fonction à invoquer et donne lieu à de nombreuses déclinaisons : `@staticmethod` pour appeler les méthodes statiques d'une classe, `@my_decorator` pour appeler un décorateur développé pour l'occasion... pour une rapide description, nous référons à [45] p. 1034 - p. 1040. Cependant une telle syntaxe manque en C++. Il faut l'émuler. Ce diagramme de classe décrit son fonctionnement et montre comment l'émuler. Le décorator permet d'entourer l'exécution d'une méthode d'un objet d'une phase de préparation et d'une phase de post-traitement 2.8.

```
class tracer:                                                          1
  def __init__(self, func):  # Remember original, init counter        2
    self.calls = 0                                                     3
    self.func = func                                                   4
  def __call__(self, *args): # On later calls: add logic, run original 5
    self.calls += 1                                                    6
    print('call %s to %s' % (self.calls, self.func.__name__))         7
    return self.func(*args)                                            8
                                                                       9
@tracer                      # Same as spam = tracer(spam)             10
def spam(a, b, c):           # Wrap spam in a decorator object         11
 return a + b + c                                                      12
                                                                       13
print(spam(1,2,3))           # Really calls the tracer wrapper object  14
print(spam('a','b','c'))     # Invokes __call__ in class               15
```

```
c:\code> python tracer.py                                              1
call 1 to spam                                                         2
6                                                                      3
call 2 to spam                                                         4
abc                                                                    5
```

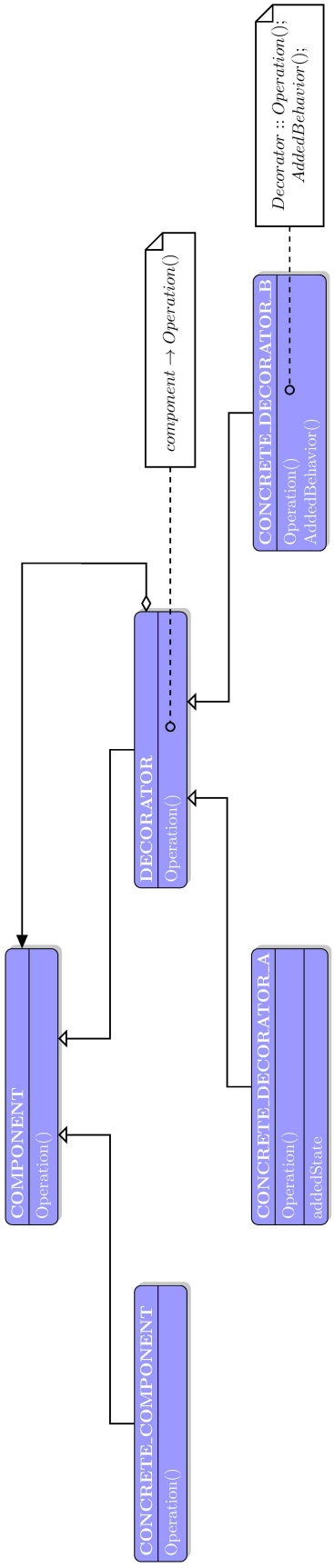


Figure 2.6 – Schéma de conception décorateur

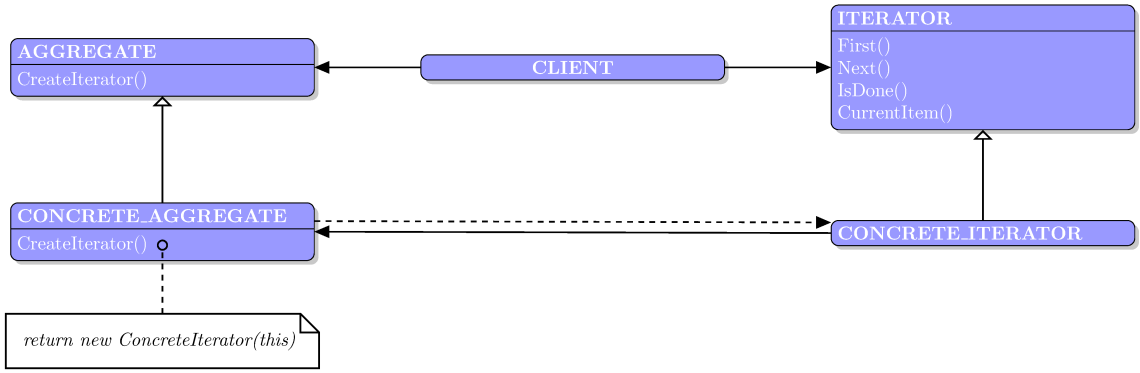


Figure 2.7 – Diagramme de classe d'un itérateur

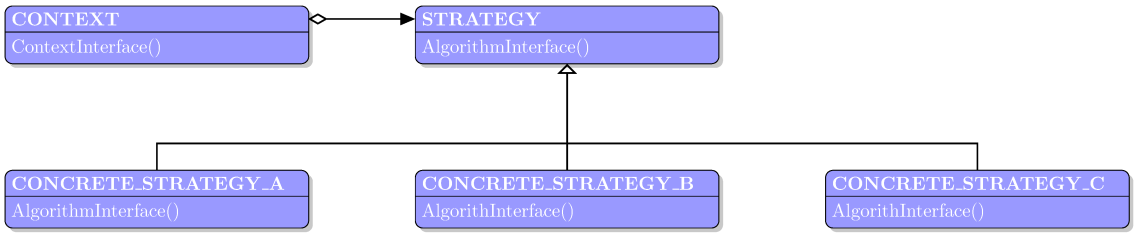


Figure 2.8 – Schéma de conception stratégie

Stratégie Le schéma de conception appelé *strategy* permet de modifier le comportement d'un objet à l'exécution. Il est préférable de l'utiliser plutôt que dériver systématiquement la classe. En effet, l'héritage n'est pas si souvent que cela une solution appropriée. Si l'on utilise l'héritage, il est préférable de ne pas surcharger directement les méthodes publiques, mais de créer des méthodes privées, appelées par les méthodes publiques, que l'on surcharge. Cela permet un contrôle plus étroit de ce qui est exécuté et conduit à éviter les mauvaises surprises. Notamment, en l'absence de documentation, dériver une méthode est dangereux car on ne sait pas où cette méthode est appelée. Il faut indiquer, par un `@override` que la dite méthode est susceptible d'être dérivée. C'est pourquoi, il mieux vaut utiliser la composition. Le schéma de conception *strategy* exploite en particulier cette idée.

De même que le langage Python propose une syntaxe pour le décorateur, de même le C++11 met en oeuvre des syntaxes pour la stratégie : il s'agit des expressions lambda. Bénéficiant d'une syntaxe très ramassée, elles clarifient le développement. Une expression lambda typique est : `[&os,m](int x)! if (x\%m)os << x << '\n'; !` ([62] p. 291). Elles sont composées d'une liste d'objets capturés `[]` qui jouent le rôle d'attributs de classe et d'une série d'arguments `()`, arguments passés à l'opérateur `operator()` de la classe. Elles sont très souvent utilisées comme *callback*. L'emploi des `tr1::function` permet d'étendre l'usage du pointeur vers une fonction à un pointeur vers un objet qui se comporte comme une fonction comme expliqué dans [47], Item 35, p. 173. L'usage d'une telle signature combinée aux fonctions standard de lien comme `std::bind` introduit une très grande flexibilité.

Visiteur Le schéma de conception, dit *visitor*, permet d'ajouter des comportements à une classe sans la modifier au prix d'indirections qui peuvent s'avérer coûteuses ou d'exporter des fonctionnalités partagées par un ensemble de classes. Il est particulièrement efficace

quand il s'agit d'ausculter une hierarchie pour extraire des informations sur l'état et le comportement des instances des objets lors de débogage. Ainsi que présenté dans 2.9, le client a accès à deux interfaces, `class VISITOR` et `class ELEMENT`. C'est dans les spécialisations de l'interface `class VISITOR` que se niche l'implémentation des fonctionnalités additionnelles. Quant aux spécialisations de l'`ELEMENT`, elles implémentent la méthode virtuelle `Accept(Visitor v)`. L'implémentation garantit que la méthode correspondant à la spécialisation de l'élément considéré sera exécutée.

2.3 Bas niveau : types génériques et template metaprogramming

Introduction Nous tentons de justifier l'emploi de techniques de métaprogrammation dans une communauté où leur utilisation n'est pas courante. La metaprogrammation permet par ailleurs de programmer le compilateur lui-même pour lui faire générer du code spécifique.

Le code ZSET ne comporte pas de types génériques exceptés ceux utilisés classiquement pour les listes, les piles, les files... i. e. ce que l'on appelle en Java les collections [14]. La STL [47] est aussi malheureusement volontairement écartée du code, à notre connaissance pour des raisons historiques. Afin de respecter le style du programme coeur, pour que les futurs développeurs s'y retrouvent, nous n'avons pas utilisé la STL.

Définition Le *metaprogramming* permet de contrôler finement la compilation pour générer du code sur mesure à partir de patrons, les *templates*. On parvient en utilisant ces techniques à diminuer drastiquement le nombre de lignes de code en évitant les redondances. Un autre avantage, moins évident, mais non moins important : le travail sur les types permet de mettre en lumière les concepts utilisés et de s'affranchir du manque de clarté qui règne souvent lorsqu'on implémente des fonctionnalités de bas niveau. Ainsi, les types génériques et les *templates* permettent le réemploi des concepts et des idées jusque dans les plus basses couches logicielles. L'implémentation des fonctions génériques elle-même se fait en faisant usage de l'efficacité et de la brièveté du C, sous-ensemble du langage C++.

Motivation pour l'utilisation Cependant ces techniques sont loin d'être une panacée. Afin d'envisager leur mise en oeuvre, pour qu'elles revêtent quelque forme d'intérêt, il faut pouvoir répondre à l'affirmative à la question : "est-ce que le type générique développé est connu avant l'exécution ?" (fig. 2.11). Si cela n'est pas le cas, il convient de se tourner vers les fonctionnalités orientées objets fournis par le langage. Les types génériques et la métaprogrammation permettent donc de factoriser un certains nombres de routines utilisées dans le programme : les instructions données au compilateur sont suffisantes pour produire le code désiré.

Exemples d'application Les *templates* sont utilisés par exemple pour s'assurer de la cohérence des unités, optimiser les opérations matricielles - la librairie Eigen [37] développée à l'Inria fonctionne à partir de *template* -. Il existe des implémentations sous forme *template*

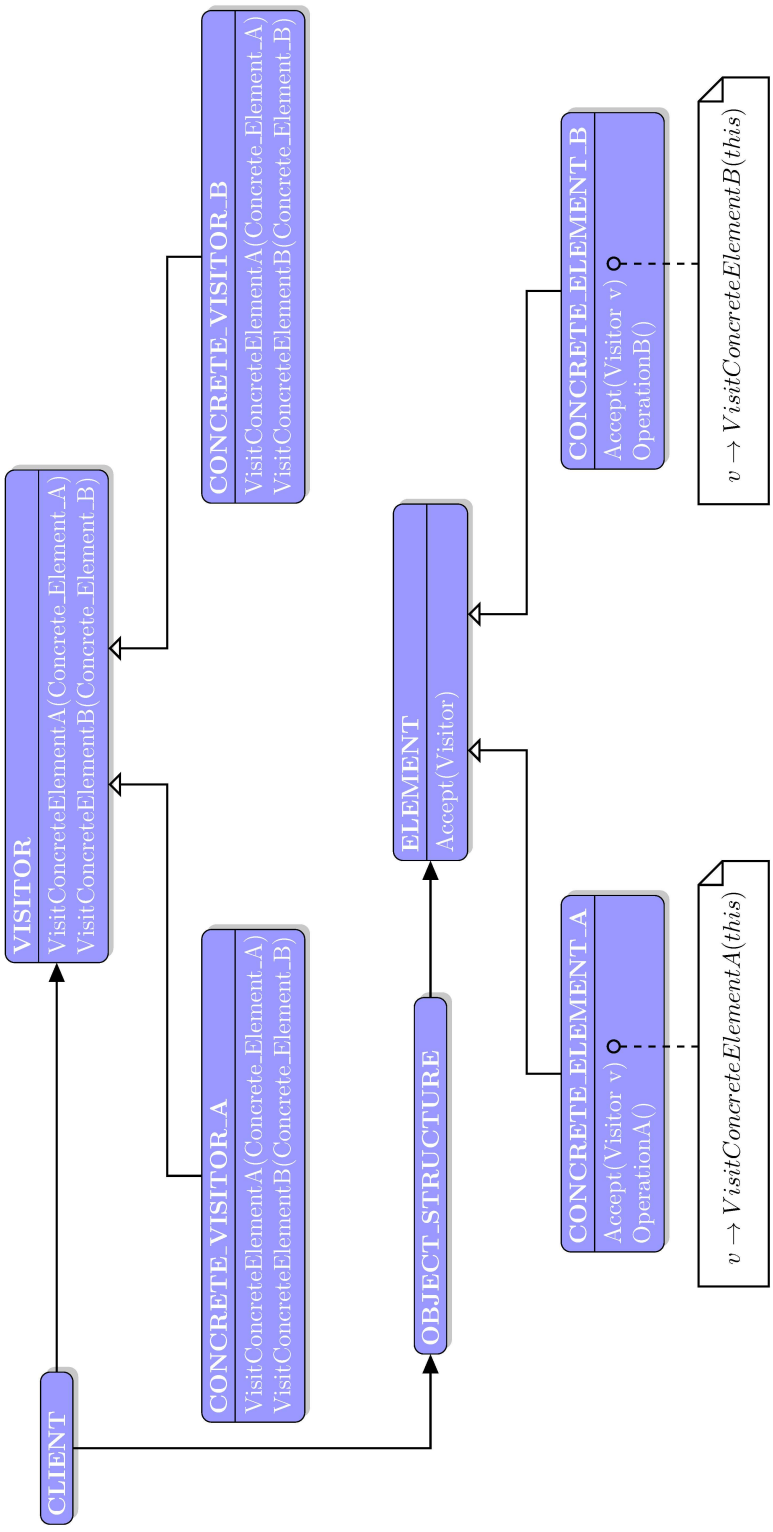


Figure 2.9 – Diagramme de classe d'un visiteur

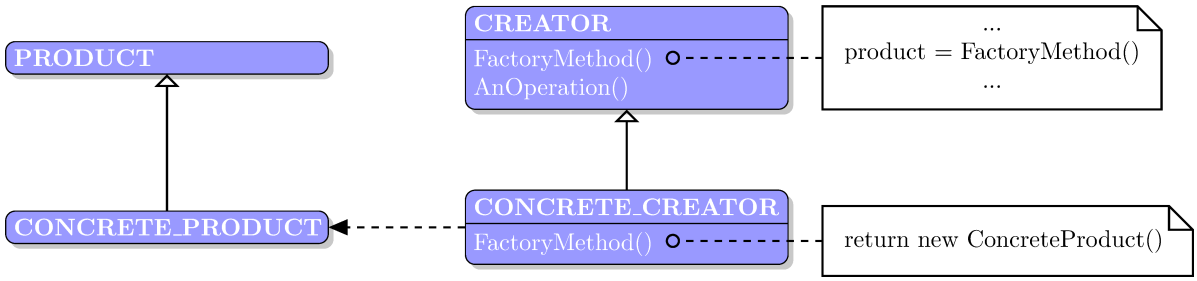


Figure 2.10 – Diagramme de classe d'une méthode de construction

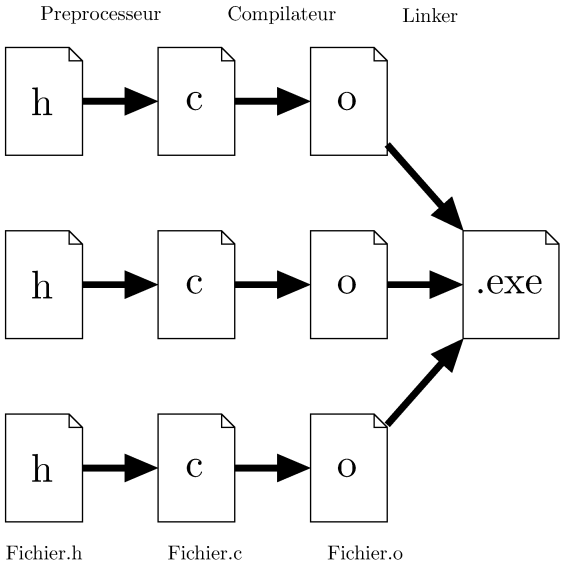


Figure 2.11 – Processus de fabrication des exécutables binaires

des schémas de conceptions usuels. Par exemple, dans [2] sont données des implémentations se servant des *templates* des *functors*, des *singletons*, des *smarts pointers*, d'une *object factory*, d'une *abstract factory* et d'un *visitor*. Nous recommandons ce livre très spécialisé seulement en seconde lecture.

2.3.1 Les classes de traits et de politiques

Définition Les classes hôtes sont construites grâce à un assemblage de petites classes, les politiques. La donnée d'un ensemble de politiques permet de paramétrer le patron de la classe hôte et de définir un comportement. On peut alors en assemblant une classe à partir d'un petit nombre de politiques, obtenir un très large éventail de comportements. Les classes de politiques permettent d'obtenir des effets similaires au schéma de conception *strategy* au moment de la compilation.

Règles d'utilisation La difficulté dans l'utilisation des classes hôtes est de correctement décomposer une classe en des classes de politiques. Quelques règles à suivre :

- Une classe ne doit jamais dépendre de plus de 4 paramètres *template* ;
- Les politiques doivent fournir des fonctionnalités orthogonales ;
- Les alias pour s'affranchir des longs noms issus des classes hôtes doivent être utilisés ;

Afin d'illustrer le concept de politique, nous présentons un exemple de création d'instance emprunté à [2] p. 8.

```
template <class T>
struct OpNewCreator
{
  static T* Create()
  {
    return new T;
  }
};

template <class T>
struct MallocCreator
{
  static T* Create()
  {
    void* buf = std::malloc(sizeof(T));
    if (!buf) return 0;
    return new(buf) T;
  }

};

template <class T>
struct PrototypeCreator
{
  PrototypeCreator(T* pObj = 0)
    : pPrototype_(pObj)
  {}

  T* Create()
  {
    return pPrototype_ ? pPrototype_->Clone() : 0;
  }

  T* GetPrototype() { return pPrototype_; }

  void SetPrototype(T* pObj) { pPrototype_ = pObj; }

  private:

    T* pPrototype_;

};
```

Listing 2.2 – Exemple de politiques

Sont présentés dans le programme 2.2 des exemples de politiques qui servent à allouer et initialiser des instances d'objets dans la mémoire. Il existe en effet plusieurs façons de procéder et on peut les rassembler sous un même concept en utilisant cette technique de conception. La première politique permet de créer de manière standard une instance, la

seconde alloue un espace mémoire grâce à la fonction `malloc` du langage C puis initialise en appelant le constructeur de position du C++. Pour une discussion sur le constructeur de position, nous référons à [46] p. 39. Quant à la troisième politique, elle peut être vue comme une mise en oeuvre élémentaire du schéma de conception prototype (fig. 2.2). Il s'agit de classes patrons. A ce titre, elles peuvent opérer sur un large éventail de types ; une restriction sur les types admis, comme illustrée dans [62] p. 709-715, rapprocherait ces exemples de politiques réelles. Ainsi, il est à noter qu'on ne trouvera jamais une telle mise en oeuvre en C++ sans plus de détails. Ces politiques valent seulement pour présentation de concept.

Le pendant des politiques est la classe hôte. Il s'agit simplement d'un patron dépendant d'une ou de plusieurs politiques. Ainsi `WidgetManager` du programme 2.3 est une classe hôte dont la politique de créaction est régie par l'une des trois politiques présentées 2.2. Afin de ne pas alourdir la notation pour le client, il est d'usage de définir un alias, ici `MyWidgetMgr` pour la classe créée de manière standard. Une chose est certaine, dans une des méthodes de `WidgetManager`, en particulier celle qui requiert la création d'une instance de cette classe, la fonction `CreationPolicy::Create()` est appelée. Même si le C++ ne permet pas de définir de concepts, il permet néanmoins de s'en approcher à l'aide des patrons. Ainsi, pour définir une politique de création, il faut imposer que la classe ou le patron en question définisse une fonction `Create`. Il est possible d'exprimer à l'aide de patrons une telle contrainte ce qui permet d'approcher le concept de politique de création [62] p. 801.

```
1
// Library code                                                    2
template <class CreationPolicy>                                    3
class WidgetManager : public CreationPolicy                        4
{                                                                  5
  ...                                                              6
};                                                                 7
                                                                   8
typedef WidgetManager< OpNewCreator<Widget> > MyWidgetMgr;         9
```

Listing 2.3 – Exemple d'une classe de politique

Afin de contrôler à la compilation la fonction à exécuter, on utilise la surchage d'opérateur. Le but est de sélectionner parmi une liste de candidats qui portent le même nom mais possède des arguments différents, celui susceptible de correspondre au mieux à la situation. Le standard du C++ définit ainsi des règles de sélection portant sur les arguments et le contexte [62] p. 327-p. 330. Se rallier aux règles de surcharge permet de contrôler, dans une certaine mesure, le code à compiler. Mais souvent un contrôle plus étroit est nécessaire. On en vient à pratiquer ce que l'on appelle la surcharge aggressive. Cette technique consiste à réduire la liste des candidats potentiels avant que le compilateur opère automatiquement le choix. Se référant à la littérature, il existe essentiellement deux telles façon de procéder :

- celle qui consiste à utiliser des classes de traits pour définir des informations sur les types comme l'illustre la figure 2.13. Les informations supplémentaires sur les types permettent d'orienter la sélection du compilateur. Une telle technique est couramment utilisée dans les mises en oeuvre des itérateurs de la librairie standard C++. Une présentation très pédagogique est donnée dans [48], Item 47, p. 227. Le principe est simple. Il s'agit de définir les caractéristiques entrant en jeu comme des classes à part entière. La figure 2.13 illustre une architecture typique de classe de trait. Les caractéristiques

en jeu sont ici le fait de posséder ou non l'opérateur `operator!=` ce qui se traduit par la définition de deux classes : `struct Has_not_equals` et `struct Hasnt_not_equals`. Il est important que les classes définissant ces caractéristiques soient constructibles et sans attributs afin de donner lieu à des optimisations. Chaque itérateur contient un alias vers une de ces deux caractéristiques appelée ici `not_equality_category`. La classe de trait, `struct iterator_traits` agit comme une fonction au moment de la compilation et permet de déterminer, à partir du type de l'itérateur, si ce-dernier possède ou non de l'opérateur `operator!=`. Puis, toujours au moment de la compilation, les règles de résolutions entrent en vigueur et, parmi les deux méthodes `doFind`, celle qui correspond au bon type d'itérateur est substituée à l'intérieur de la fonction `find`, seule fonction que le client connaît. Une fois la compilation terminée, et à l'exécution seulement, sera imprimé le message d'annonce en fonction du type d'itérateur compilé.

- celle qui consiste à utiliser la propriété baptisée SFINAE, qui est une règle du langage (iso. 14.8.2) et qui signifie textuellement "Substitution Failure Is Not An Error" et les spécialisations de la classe *template* standard `enable_if`, ce qui permet de contrôler la liste des candidats à la surcharge. Des informations concernant cette technique peuvent être trouvées dans [62] p. 692. Pour fixer les idées et comme il s'agit d'une technique que nous avons employée, nous en donnons un exemple d'application illustré par la figure 2.12. La librairie standard définit un patron `enable_if` de la façon suivante :

```
template <bool, class T = void>
struct enable_if_c
{
   typedef T type;
};

template <class T>
struct enable_if_c<false,T>
{};

template <class Cond, class T = void>
struct enable_if
 : enable_if_c<Cond::value,T>
{};
```

Listing 2.4 – Principe de fonctionnement de `enable_if`

Si l'on applique la construction du `enable_if` à l'exemple du programme 2.5 et que l'on essaie de compiler, on obtiendra une variable `i` de type entier et une erreur de compilation au niveau de `j`.

```
struct Cond { static bool value = true; };
typename enable_if<Cond,int>::type i;

struct Cond { static bool value = false; };
typename enable_if<Cond,int>::type j;
```

Listing 2.5 – Cas d'application

Ainsi, on remarque que quand la condition `C` est fausse, `enable_if_c<C,T>::type` n'existe pas. Maintenant, les standards du C++ de la surcharge des opérateurs spécifient que quand la déduction d'un argument d'une fonction *template* échoue, cela écarte simplement la fonction du processus de surcharge mais ne cause pas d'erreur (SFINAE) [1] p. 211. Par ce moyen,

en combinant avec `enable_if`, il est possible de sélectionner à la compilation la fonction à utiliser par une syntaxe bien plus naturelle que celle des classes de traits.

2.4 Les codes éléments finis : très bref aperçu de ZSET

Le code hôte utilisé est le code éléments finis ZSET. Ce code propriétaire d'accès libre aux universitaires a été développé conjointement par l'Ecole des Mines au Centre des Matériaux, l'Onera et Northwest Numerics. Ses atouts majeurs sont :

- la modularité favorisant grandement la maintenabilité du code, sa facilité d'utilisation en tant que développeur, son extensibilité qui permet d'appliquer le code existant directement à de nouveaux éléments, de nouveaux schémas de discrétisation, de nouveaux solveurs et d'autres problèmes - sans compter le debogage facilité car il est désormais possible d'isoler clairement les fonctionnalités à tester ;
- l'usine à objet étendue à l'ensemble des composants du code qui, associée à une interface fédératrice pour les composants - `class PROBLEM_COMPONENT` - permet de créer facilement des plugins et de les appeler dans la mise en données.

Nous allons aborder successivement ces deux points. L'organisation de ce code éléments finis orienté objet est donnée dans les articles [12] et [30] en ce qui concerne les aspects calculs parallèles haute performance.

Au cours de nos développements, nous nous sommes efforcés d'être le moins intrusif possible et nous avons privilégié des fonctionnalités qui font concensus entre les codes. C'est pourquoi nous procédons ici, pour la description du premier point, à une succincte étude bibliographique qui s'intéresse à l'architecture d'autres logiciels de calcul éléments finis ; l'objectif étant de relever les plus petits communs dénominateurs entre les codes présentés et de décrire en filigrane la structure de ZSET. Ce premier point se subdivise ainsi naturellement en deux niveaux :

- l'étude de la structure logicielle nécessaire au traitement de la non-linéarité ;
- l'étude de l'organisation requise pour effectuer les procédures d'assemblage et de résolution d'un problème éléments finis linéaire.

2.4.1 Architecture d'un problème éléments finis non-linéaire

Fonctionnalités majeures Dans les années 90, de nombreux travaux ont été menés afin de proposer une structure modulaire pour le calcul éléments finis non-linéaire. Le problème incrémental à résoudre est présenté en figure 2.14. L'article [12] recense quatre fonctionnalités majeures dont on se sert pour résoudre un problème éléments finis et qui sont en partie résumées sur la figure 2.14 :

- parcourir la géométrie et charger les fichiers d'entrées
- définir la structure globale de l'équation

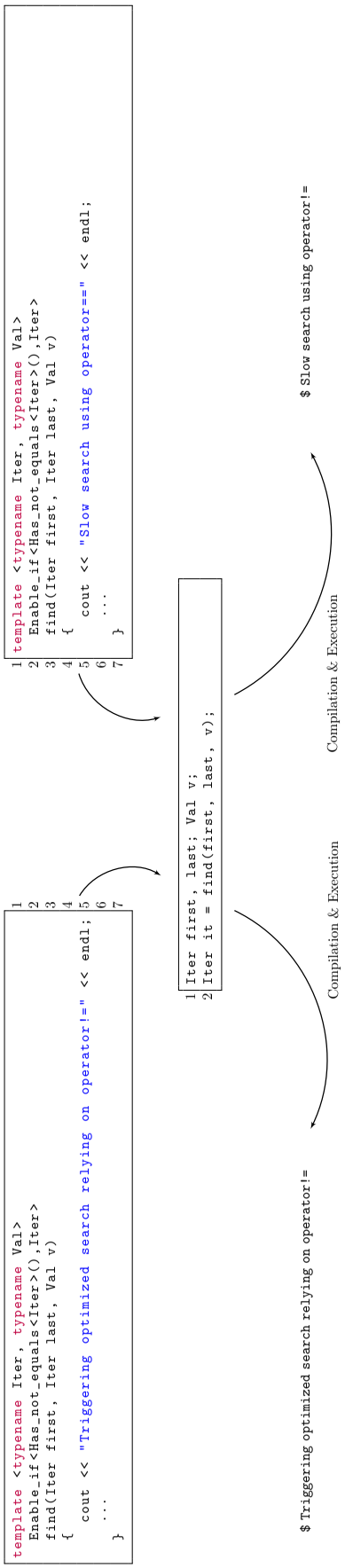


Figure 2.12 – Principe d'utilisation du standard SFINAE

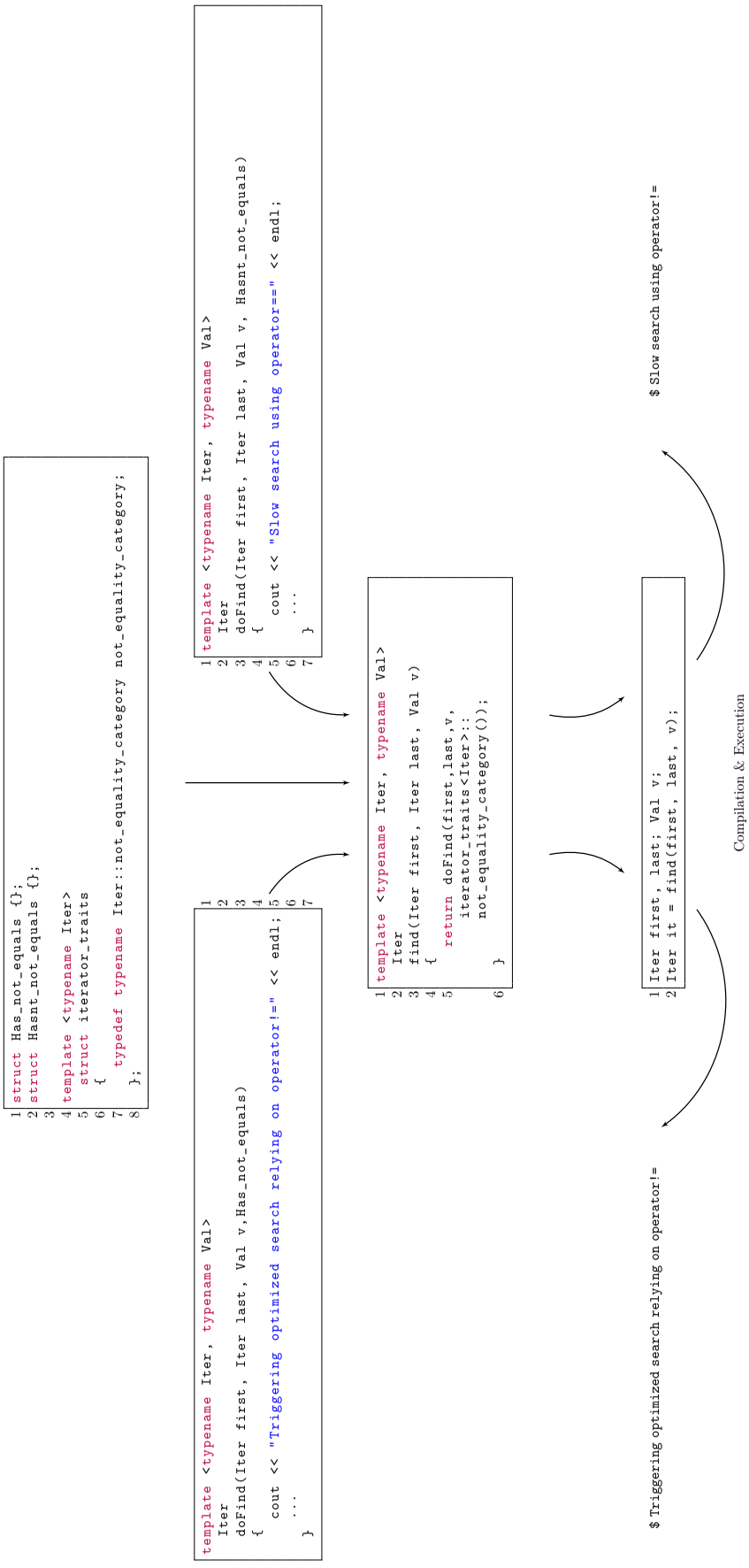


Figure 2.13 – Principe d'utilisation des politiques

- appliquer un algorithme pour résoudre l'équilibre global et évaluer le résidu
- générer les fichiers de sorties appropriés à la fin des incréments

Architecture couramment adoptée Dans [53], une architecture orientée objet pour résoudre ce problème, très proche de celle adoptée dans ZSET, est proposée. Elle repose essentiellement sur deux hiérarchies de classe. La première issue d'une interface abstraite représentée [53] par `class EngnModel` et dans ZSET par `class PROBLEM`. Les spécialisations de cette interface - comme `class NLStatic` ou `class PROBLEM_STATIC_MECHANICAL` pour les calculs de mécanique non-linéaire en statique - représentent le modèle physique sous-jacent. Le rôle de cette interface est de fournir un cadre au déroulement du calcul. Elle organise séquentiellement les étapes du calcul de la figure 2.14. Disposant désormais d'un cadre réglant la marche et l'enchaînement global du calcul, il faut ensuite proposer une interface pour définir les opérations à exécuter dans chaque bloc de 2.14. A cette fin, on propose aussi bien dans ZSET que dans [53] de construire une seconde hierarchie, respectivement celle des `class ALGORITHM` ou des `class NumericalMethod`, dont les spécialisations sont chargées de définir les opérations nécessaires à l'accomplissement de chaque bloc de 2.14. Cette double hiérarchie permet ainsi de séparer l'ensemble des tâches impliquées dans la résolution d'un problème physique du détail de leur réalisation numérique effective.

Traitement des non-linéarités Plus en détail, la méthode classique de résolution consiste d'abord en une discrétisation temporelle en pas de temps : les incréments. Puis dans la mise en oeuvre d'une procédure de Newton-Raphson pour trouver un champ de déplacement $\boldsymbol{u}$ qui, annulant le résidu, satisfasse l'équilibre mécanique ou thermique. La procédure de Newton-Raphson, pour converger, exécute une succession d'itérations. Au cours de chaque itération, on résout un problème éléments finis linéaire. C'est ainsi que l'on ramène la résolution d'un problème éléments finis non-linéaire à des résolutions successives de problèmes éléments finis linéaires, objets de la section 2.4.2.

2.4.2 Architecture d'un problème éléments finis linéaire

Comparaison ZSET-deal.II Nous présentons l'architecture à travers une comparaison entre deal.II et ZSET.

Le code éléments finis deal.II est présenté comme un succès de développement dans [9]. Il s'agit d'un code libre lancé en 1997, successeur de *deal*, dont l'architecture était devenue désuette suite à la progression du langage C++. C'est une librairie d'usage général, dont le but est d'aider à l'écriture de routines éléments finis. L'introduction de l'objet maillage `class Triangulation` de deal.II ou `class GMesh` de ZSET et d'éléments `class FiniteElement` dans deal.II ou `class Element` dans ZSET permet de gérer collectivement des éléments aux géométries différentes. Dans deal.II, à la différence de ZSET, la dimension du maillage est un paramètre de patron : `Triangulation<dim> triangulation` ce qui confère une opportunité d'optimisation au moment de la compilation. En outre, deal.II propose des itérateurs pour accéder aux éléments ce qui évite les confusions surgissant à cause des problèmes de renumérotations.

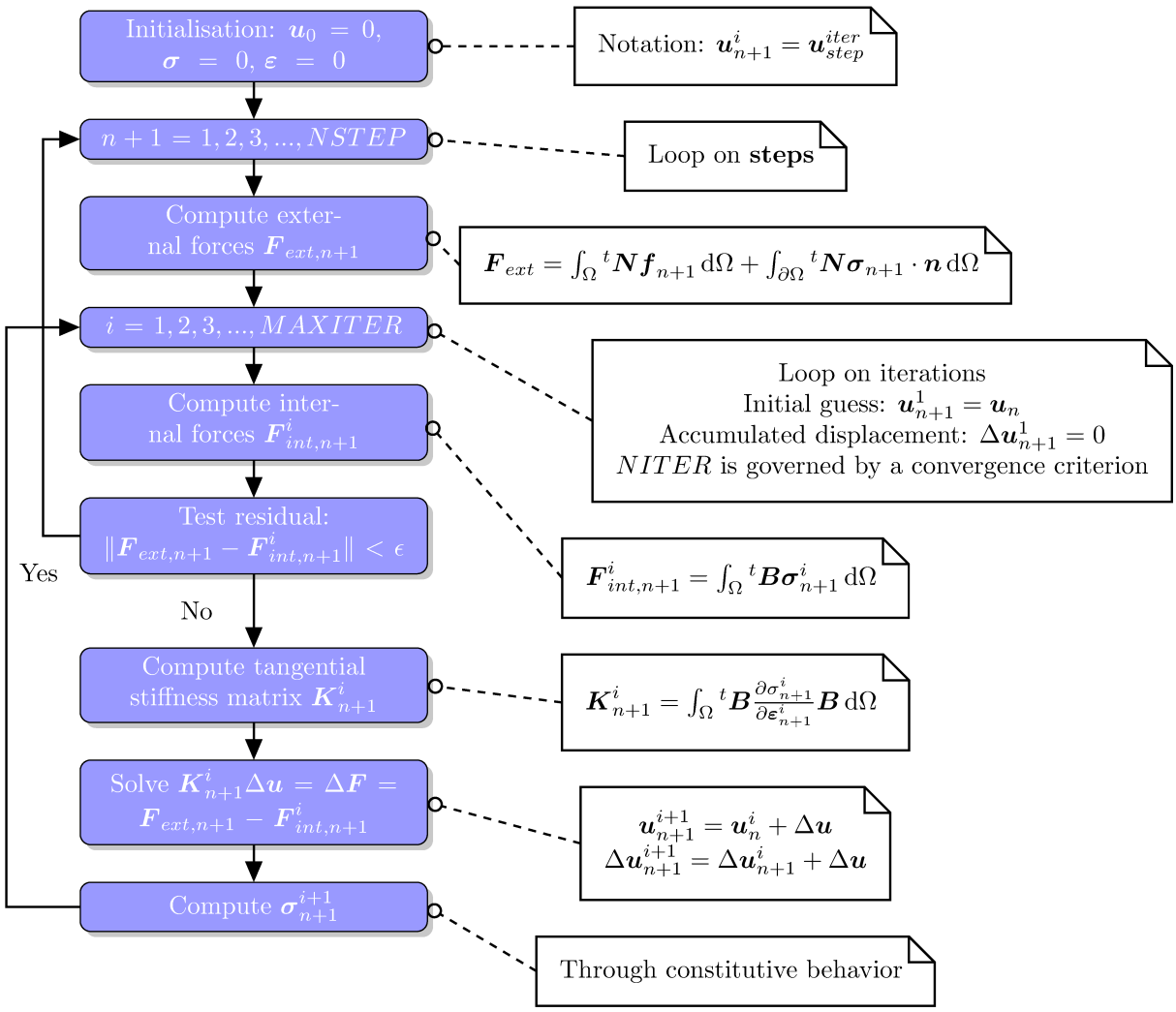


Figure 2.14 – Algorithme standard des éléments finis non-linéaire

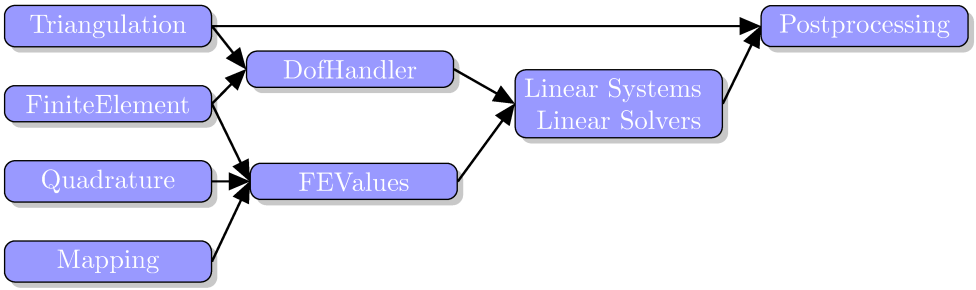


Figure 2.15 – Architecture globale de deal.II

Alors que dans ZSET les éléments eux-mêmes sont responsables d'établir la correspondance entre les informations géométriques et la description des degrés de liberté, au travers notamment de la méthode `D_ELEMENT::setup_dofs()` c'est le `DoFHandler::distribute_dofs` du deal.II à un niveau global qui a pour vocation de distribuer les degrés de liberté. Aux classes `FiniteElement`, `Quadrature`, `Mapping` et `FEValues` du deal.II (fig. 2.15) font écho les classes `SHAPE_FUNCTION`, `INTEGRATION`, `GEOMETRY` et `INTERPOLATION` de ZSET. En particulier, la proximité entre les architectures se mesurent également aux très grandes similitudes formelles entre les routines d'assemblage de la matrice de rigidité des deux codes :

```
const unsigned int dofs_per_cell = finite_element.dofs_per_cell;
std::vector<unsigned int> global_dof_indices(dofs_per_cell);

for (DoFHandler<dim>::active_cell_iterator
      cell = dof_handler.begin_active();
     cell != dof_handler.end(); ++cell)
{
  // compute local_a

  cell->get_dof_indices(global_dof_indices);

  for (unsigned int i=0; i != dofs_per_cell; ++i)
    for (unsigned int j=0; j != dofs_per_cell; ++j)
      global_a(global_dof_indices[i],global_dof_indices[j])
        += local_a(i,j);
}
```

```
INTEGRATION_RESULT* MESH::compute_internal_reac(GLOBAL_MATRIX*
    elementary_matrix,
                                                bool if_compute_stiffness,
                                                bool get_only_matrix)
{

  D_ELEMENT* elem = NULL;

  for (int ielem=0; ielem<element.size();ielem++)
  {
    elem->internal_reaction(if_compute_stiffness,internal,stiff,
        get_only_elementary_matrix);
    ...
    elementary_matrix->stor(*elem,stiff);
  }
  ...
}
```

Les auteurs présentent aussi l'architecture du code. Nous reprenons ici les points essentiels qui nous seront utiles pour la suite de l'étude. Sont présentés ici deux diagrammes de classe : un premier présente l'interaction de `class ELEMENT` avec les autres classes du code, un second la hiérarchie de `class ELEMENT`

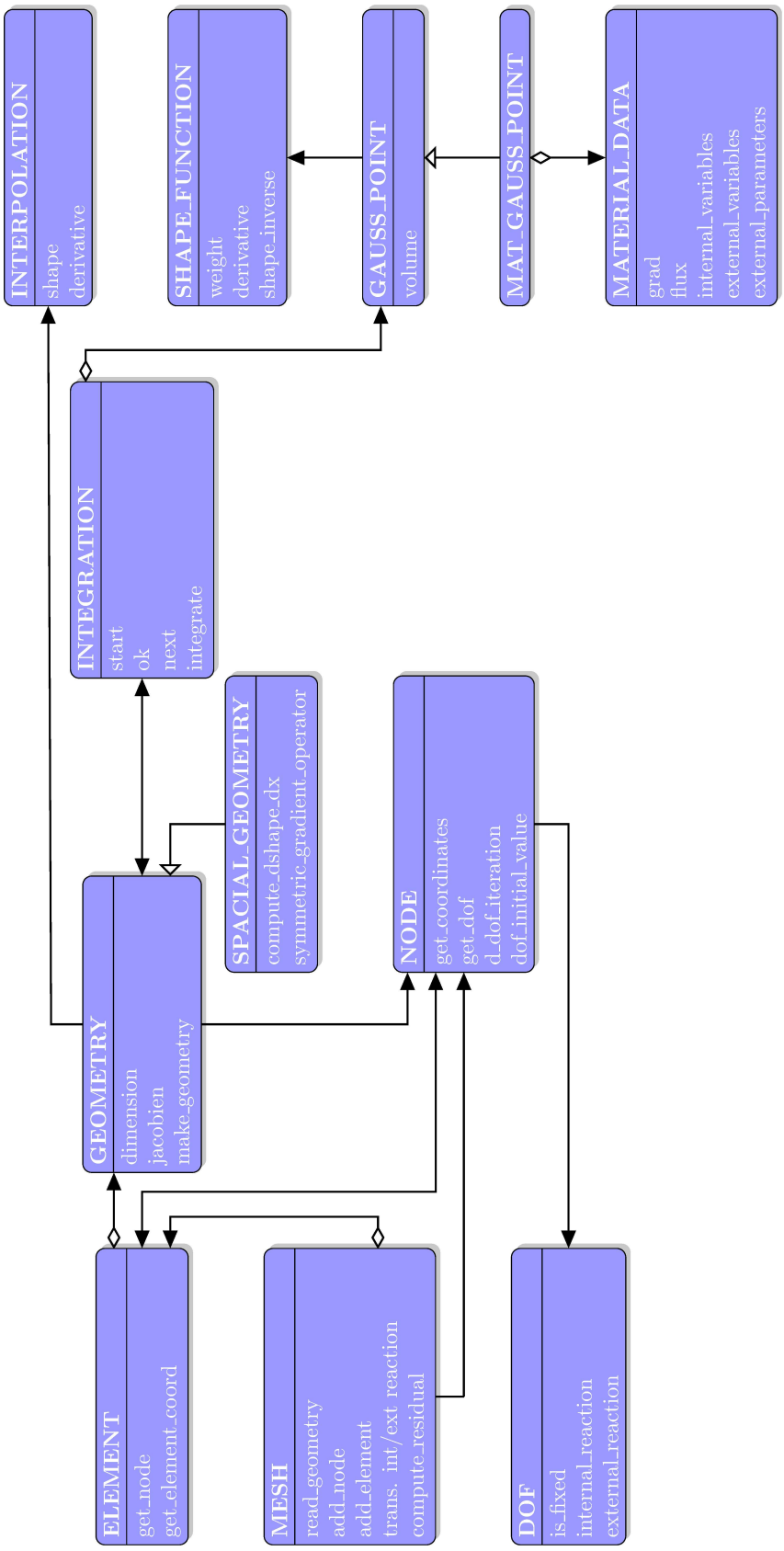
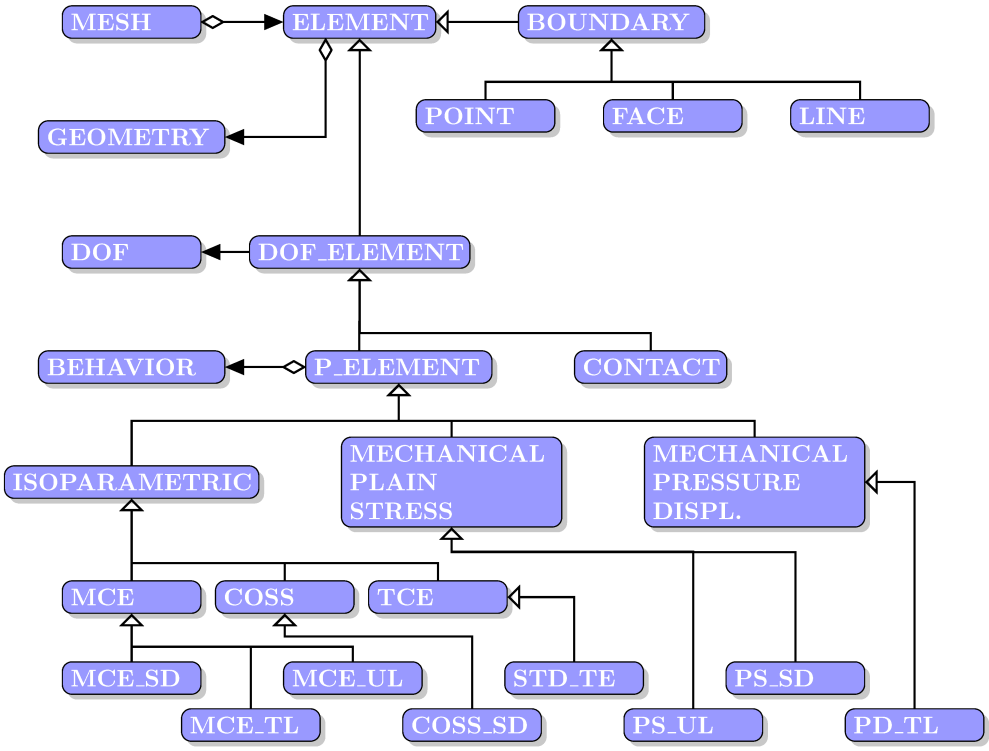


Figure 2.16 – Organisation du code ZSET



Les éléments Le maillage `class MESH` reccueille les éléments `class ELEMENT`. Chaque élément possède une géométrie `class GEOMETRY`. Le `class DOF_ELEMENT` ajoute à la géométrie la gestion des degrés de libertés et `class P_ELEMENT` une loi de comportement matériau. A ce niveau, il n'apparaît aucun détail d'implémentation. Ce sont des classes très générales dont on hérite pour construire des nouveaux éléments. Les équations de la physique sont implémentées dans les `class MCE`, `class COSS`, `TCE`, `class MCE_SD`, `class MCE_TL`, `class MCE_UL`, `class STD_TE`, `class PS_UL`, `class PS_SD` et `class PD_TL`.

A plus fin niveau de granularité, sur la figure 2.16, on distingue essentiellement deux fonctionnalités : celles qui tiennent à la gestion de la géométrie et de la topologie et celles qui concernent la gestion des espaces d'approximation. D'abord, la gestion de la géométrie est effectuée par les méthodes `ELEMENT::get_elem_coord` qui donnent la position des noeuds de l'élément, `MESH::read_geometry`, `MESH::add_node` qui permet d'ajouter un noeud au maillage, `NODE ::get_coordinates` qui renvoie la position du noeud. Puis la gestion des espaces d'approximation est assurée essentiellement par les autres méthodes de `class GEOMETRY`, `class SPACIAL_GEOMETRY`, `class INTEGRATION` qui permet de parcourir lors de l'intégration l'ensemble des points de Gauss de l'élément grâce à l'itérateur constitué par les méthodes `start`, `ok`, `next`, `class INTERPOLATION`, `class SHAPE_FUNCTION` et `class GAUSS_POINT`. La classe `MATERIAL_DATA` sert à stocker les valeurs des variables internes, contraintes et déformations aux points de Gauss dans les calculs de mécanique éléments finis non-linéaire.

2.5 Mécanisme d'extension de fonctionnalités

Nous passons maintenant au second point, objet de la présente section, qui concerne les mécanismes d'extension des fonctionnalités. Plus particulièrement nous proposons une description de l'usine à objet de ZSET et de son système de greffons, après une brève introduction.

Interpréteur Différents mécanismes d'extension peuvent être employés. Deal.II [9] se présente ainsi sous la forme d'un système de librairies que le client utilise pour se construire une application sur mesure. D'autres logiciels éléments finis disposent d'un interpréteur de langage offrant des fonctionnalités de haut niveau, possédant une syntaxe suffisamment expressive pour délester l'utilisateur de préoccupations informatiques sans pour autant le contraindre. Par exemple freefem++ [38] accepte des formulations variationnelles. Ainsi on saisit :

```
macro Grad(u) [dx(u),dy(u), dz(u)]
Xh p,q;
solve laplace(p,q,solver=CG) =
 int3d(Th) ( Grad(p)'*Grad(q) ) -
 int3d(Th) ( 1*q );
```

pour exprimer la forme faible :

$$\int_\Omega \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{v} = \int_\Omega 1 \cdot \boldsymbol{v} \tag{2.1}$$

[26] étudie les questions de programmation orientée objet avec application à la dérivation symbolique et à la programmation automatique.

ZSET a aussi son interpréteur implémenté en se servant de l'analyseur syntaxique Yacc [39]. Si son emploi se justifie pour l'implémentation de loi de comportement matériaux ou pour des scripts nécessitant l'emploi de fonctionnalités avancées de ZSET, il ne saurait convenir pour une implémentation de l'hyperréduction.

Usine à objet généralisée La nécessité d'une usine à objet se fait sentir quand le client a non seulement besoin de manipuler les objets mais aussi de les créer.

ZSET utilise ce que l'on appelle classiquement une usine à objet, très proche du modèle proposé par [2]. Une usine à objet permet de renvoyer un pointeur vers instance d'un objet identifiée par un mot-clé.

```
typedef BASE_CLASS* (*fc)();

LIST<STRING> keywords;
LIST<fc> creators;
```

```
BASE_CLASS* f1() { return(new DERIVATIVE); }

...

keywords.add("derivative"); creators.add(f1);
```

Listing 2.6 – tiré de [30]

L'usine à objet L'usine à objet implémentée en C++ s'appuie sur une propriété du langage qui veut que les objets statiques soient créés entre le lancement du programme et l'entrée dans le `main`. Au moment où l'on souhaite créer l'objet `keyword`, l'usine à objet est invoqué par un `Create_object("keyword")`, la table de correspondance entre l'étiquette et le pointeur de la fonction qui renvoie l'objet est parcourue, la fonction est invoquée, l'objet est créé. Ce mécanisme offre une très grande flexibilité : pour créer un objet il n'est plus nécessaire de connaître ses caractéristiques, un mot-clé suffit. Cette flexibilité a un coût.

Afin d'enregistrer une classe auprès de l'usine à objet, on utilise la macro : `DECLARE_OBJECT(A, B,keyword)` qui signifie que l'on enregistre auprès de l'usine à objet la `class B` qui descend de la `class A` sous l'étiquette "keyword".

Parallèlement, la mise en place d'une usine à objets combinée à une structure favorisant la génération de plugin permet une prise en main immédiate des nouveaux développements par l'utilisateur. On souhaite pouvoir ajouter des objets dérivés sans avoir à recompiler la classe de base.

La figure 2.17 présente le mécanisme d'extension réalisé par adjonction de greffon dans ZSET. L'interface `class PROBLEM_COMPONENT` permet de prendre le contrôle des opérations à différents niveaux d'exécution (fig. 2.14) : en début de problème, lors du chargement du problème, de la lecture du fichier de mise en données, avant ou après le chargement du maillage, avant ou après remaillage, avant à l'issue d'un incrément, d'une itération, en fin de calcul, etc. Ces possibilités correspondent à autant de méthodes de l'interface `class PROBLEM_COMPONENT`. La `class PROBLEM` contient une liste de `PROBLEM_COMPONENT` dont les méthodes sont appelées à chaque étape clé. Un greffon est simplement une spécialisation de l'interface `class PROBLEM_COMPONENT` compilé indépendamment du coeur du code sous forme de librairie dynamique qui est chargée à l'exécution. Joint à l'enregistrement auprès de l'usine à objet, ici sous le nom de `mon_extension` par la commande `DECLARE_OBJECT`, ce système permet très simplement à l'utilisateur d'invoquer le greffon directement dans le fichier de mise en données en y inscrivant un `***mon_extension`. Dans le journal d'exécution figure alors le nom du greffon.

C'est ainsi que ZSET, développé sur la base de telles approches, dispose des qualités requises pour organiser simplement un plugin d'hyperréduction.

2.6 L'enjeu au regard de l'état de l'art

Grandes similarités entre code FE orientés objet Ainsi que le souligne l'article [6] qui propose une architecture logicielle pour construire et utiliser des modèles réduits de type

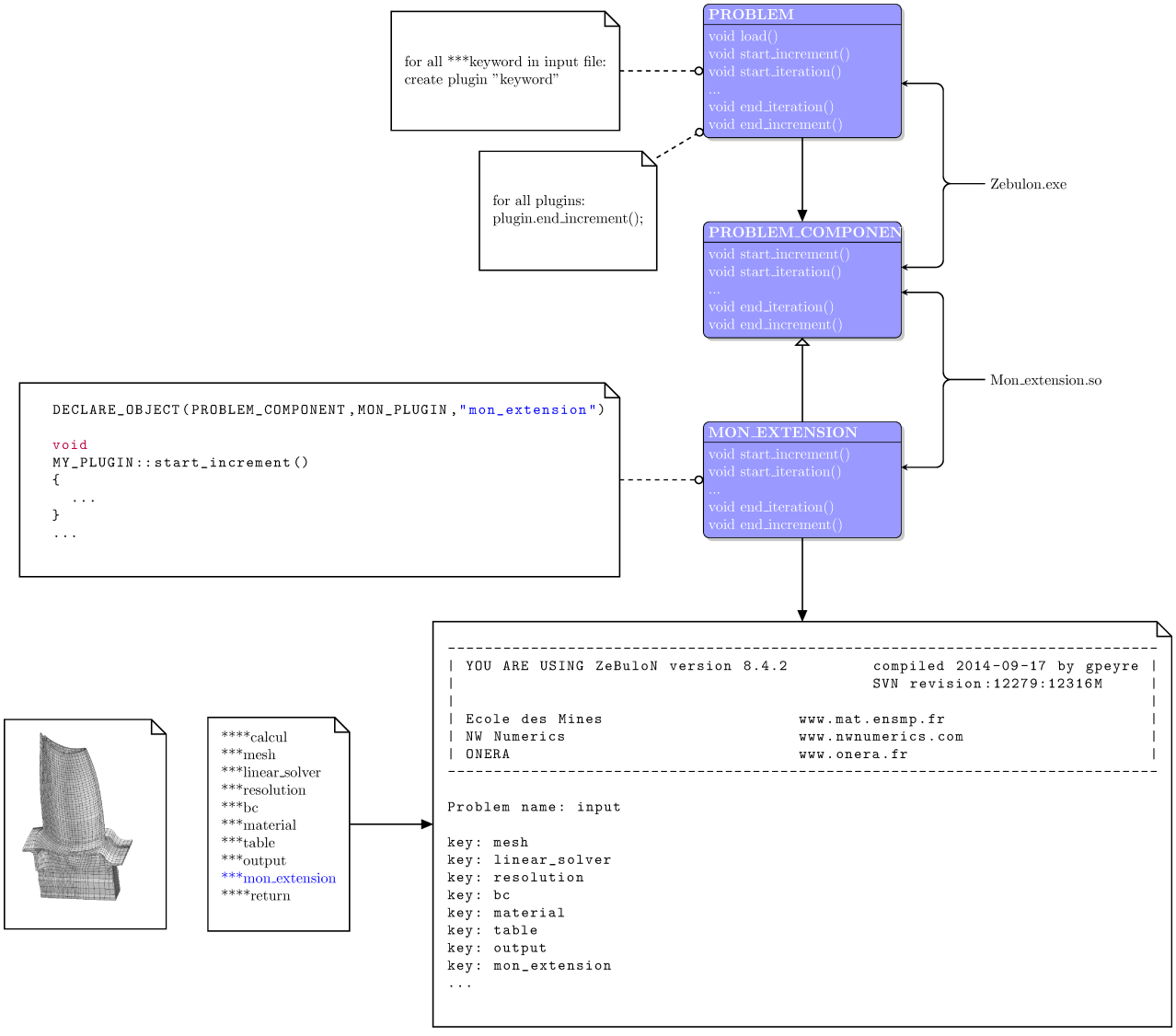


Figure 2.17 – Système de plugin dans ZSET

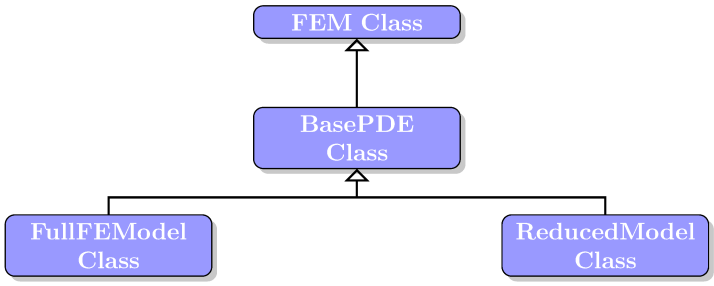


Figure 2.18 – Architecture de la réduction telle que formulée dans [6]

T1	Composite	T2	Singleton	T3	Factory Method
T4	Strategy	T5	Decorator	T6	State
T7	Visitor	T8	Iterator	T9	Cross-casting
T10	Classe de traits	T11	Politiques	T12	SFINAE
T13	Interdiction de modifier les classes standard PROBLEM et ALGORITHM				
T14	Interdiction de modifier routines standard d'assemblage et de résolution				
T15	ZSET Object factory	T16	ZSET problem component	T17	ZSET Plugin

Table 2.1 – Récapitulatifs des architectures et techniques

POD, les similarités des équations à résoudre entre un calcul éléments finis classique et un calcul effectué à l'aide d'un modèle réduit sont si importantes qu'il est très judicieux d'en tirer parti par une architecture logicielle adaptée. Un principe d'architecture proposé dans [6] consiste à dériver le calcul éléments finis et le calcul réduit d'une même classe.

Peu de développements dans la littérature Les classes `class FullFEModel` et `class ReducedModel` implémentent toutes deux une version de la méthode `makeSystem()` qui permet d'assembler la matrice de rigidité et les réactions internes. Adhérer à cette architecture, c'est considérer qu'il est nécessaire d'élaborer un nouvel algorithme éléments finis pour mettre en oeuvre l'hyperréduction. Or, nous verrons que ce n'est pas nécessaire.

De plus [6] présente néanmoins certaines limites :

- En particulier il ne permet de traiter que la réduction, i. e. des calculs où le coût d'assemblage est faible devant le coût de résolution, ainsi que les auteurs le rappellent dans le préambule.
- S'il sépare nettement les routines liées au calcul éléments finis de celle liées au calcul réduit, il ne semble pas désolidariser la méthode de construction du modèle réduit de son assemblage.
- L'analogie entre un calcul éléments finis classique et un calcul hyperréduit peut être développée encore plus avant, sans même reprogrammer la procédure d'assemblage, en agissant au niveau des éléments.
- L'application retenue est un problème thermique. De nouvelles difficultés surgissent lors de la prise en compte de problèmes mécaniques.

Une partie très importante de notre thèse peut être lue comme une proposition d'améliorations à effectuer pour remédier à ces limites.

2.7 Conclusion

Pour rappel, nous avons présenté les schémas de conception que nous allons utiliser, des techniques de metaprogrammation avec des patrons et certaines parties d'intérêts pour la suite de nos travaux de l'architecture orientée objet d'un code éléments finis, le code éléments finis hôte : ZSET.

Notre tâche sera simplement d'appliquer à la lettre les techniques relevées dans la littérature et exposées dans ce chapitre. La qualité du logiciel obtenu se mesure à l'aune de l'adéquation entre la façon dont sont définies les techniques dans la littérature et la façon dont on les a appliquées : toutes les parties du code sont justifiées bibliographiquement. Cela permet de désolidariser les développements futurs de l'auteur du plugin et de garantir la maintenabilité, par référence commune à la bibliographie.

Partie II

Mise en oeuvre et applications

3

Le plugin d'hyperréduction

Au début du projet, le code de l'hyperréduction dans ZSET comptait quelques 14000 lignes écrites en C++. Il n'était ni orienté objet, ni commenté ce qui rendait extrêmement difficile sa prise en main. Il n'existait pas non plus de notice explicative. Le code avait été développé au fil des ans, pour répondre au besoin de la recherche et à l'émergence de nouvelles idées, mais sa structure n'avait jamais vraiment été pensée. Il était très difficile pour un étudiant de l'utiliser ou de s'en servir pour vérifier ses idées. Dans la littérature, nulle trace à notre connaissance d'un article décrivant un couplage étroit entre les éléments finis classiques et la réduction de modèle.

Une part importante de ma thèse a été de le repenser entièrement. Il ne s'agit pas d'une simple reprogrammation. Nous avons essayé de mettre à jour ou de simplifier les algorithmes qui étaient jusqu'à présent implémentés en les remplaçant par ceux que l'on trouvait dans la littérature. La version du code que nous proposons est modulaire, orienté objet. Elle comporte en outre un faible nombre de lignes de C++, de l'ordre de 2000 si l'on omet les commentaires, ce qui signifie que la taille du code a été réduite d'un facteur 10. L'ensemble des concepts de la réduction de modèle a été repensé entièrement et réinterprété dans un moule élément fini.

Derrière cette réimplémentation s'est toujours nichée la volonté d'être pédagogue, l'idée qu'un code bien expliqué et pédagogiquement implémenté fait progresser la recherche sur le long terme, les bénéfices de court terme vu l'ampleur de la tâche à l'échelle d'une thèse ne pouvant rester que très modestes.

La nouvelle version tente de répondre à la problématique suivante : comment faire de l'hyperréduction avec un code éléments finis de façon transparente pour ce dernier ? Le code éléments finis croit exécuter un calcul éléments finis classique. Or, il n'en est rien : c'est un calcul hyperréduit qui est en fait exécuté. Quoi de plus naturel alors que d'essayer de développer l'analogie entre éléments finis classiques et hyperréduction en introduisant un élément d'un type particulier, un élément réduit ? Nous sommes certains que cette vision permettra de mieux appréhender la réduction et l'hyperréduction. Et là réside sans doute notre contribution.

Ce chapitre se développe en trois axes :

- *Dans un premier temps, nous effectuons une analyse des principales fonctionnalités requises par le plugin, nous mettons en relation ces fonctionnalités avec celles déjà présentes dans ZSET. De cette analyse découle naturellement notre proposition d'architecture pour le plugin d'hyperréduction. Puis à un niveau de granularité inférieure, nous montrons comment s'incrit cette architecture de plugin au sein de l'architecture*

du code cible ZSET. On peut distinguer essentiellement deux grands types de fonctionnalités : les fonctionnalités nécessaires à l'accomplissement des opérations du stade de calcul online et celles requises pour le stade offline.

- *Les composantes d'architecture online mises en exergue reposent sur les notions d'éléments réduits et de conditions aux limites réduites. Le lien entre les éléments réduits et les formulations variationnelles de l'équilibre discrétisé est souligné. Nous montrons quels peuvent être les atouts d'un élément hyperréduit en terme de coût de calcul par rapport à une formulation globale de l'hyperréduction. Enfin une implémentation dans ZSET est donnée. A chaque fois, on précise l'environnement ZSET pour en faciliter la compréhension, qui peut s'avérer difficile, ZSET ne disposant que de très peu de commentaires.*
- *La construction du modèle réduit lors de l'étape offline est abordée. On présente un atelier de construction de modèle, les bases réduites étant vues comme des objets sur lesquels sont appliqués des foncteurs afin de les tranformer ou de récupérer de l'information sur leur état. Nous expliquons en quoi ce modèle reposant sur l'utilisation d'interfaces est extensible. Nous précisons dans le détail comment interagit cet atelier avec ZSET.*

Dans toute cette partie, la couleur rouge des schémas est utilisée pour décrire l'existant tandis que le bleu correspond à ce que nous avons développé.

Sommaire

.inp

```
1  ****calcul
2   ***mesh ...
3    ***hyperreduction
4     **reduced_model
5      *magnitude U 1.e-5
6      *magnitude eto 1.e-5
7      *magnitude sig 1.
8      *magnitude eel 1.e-5
9      *widen 1.
10     *ratio 0.01
11  ***resolution ...
12  ***equation ...
13  ***material ...
14 ****return
```

Figure 3.1 – Mise en données

3.1 Architecture générale du plugin

Mise en données La mise en données pour un problème d'hyperréduction adaptative est représentée sur la figure 3.1. Nous l'avons voulue aussi simple que possible. Pour lancer l'hyperréduction, il suffit d'ajouter approximativement 8 lignes à un fichier de mise en données ZSET classique. Sur la figure 3.1, le `***mesh` permet d'appeler le maillage, les commandes `***resolution` de définir un processus de résolution du problème, `***equation` de définir les conditions aux limites, `***material` de spécifier une loi matériau, enfin l'objet d'intérêt ici est le `***hyperreduction` qui permet de lancer une procédure d'hyperréduction. L'utilisateur est invité à définir les caractéristiques de construction du modèle réduit, notamment en précisant les valeurs seuil pour les variables internes regroupées sous un `*magnitude`, valeurs pour lesquelles la contribution du *snapshot* à la base réduite est jugée négligeable ; un paramètre de contrôle de l'élargissement du domaine réduit, option `*widen`, longueur, permet de définir la taille moyenne du domaine réduit ; enfin, une tolérance à l'erreur sous l'étiquette `*ratio` permet de contrôler la qualité du modèle hyperréduit et de provoquer une adaptation de ce modèle, si nécessaire.

Fonctionnalités de haut niveau Sur la figure 3.3 est représenté en rouge l'algorithme de résolution d'un problème éléments finis non linéaire classique mis en oeuvre dans tous les codes éléments finis non linéaire et déjà décrit sur la figure 2.14. Afin de pratiquer l'hyperréduction, il est nécessaire de développer des fonctionnalités additionnelles aux fonctionnalités préexistantes représentées par les blocs bleus. Pour cela, nous utilisons l'architecture de plugin de ZSET de la section et nous proposons une spécialisation de `class PROBLEM_COMPONENT` appelée `class REDUCED_MODEL_MANAGER` (fig. 3.4) et enregistrée dans l'usine à objet sous le nom de *hyperreduction* ce qui permettra à l'utilisateur de l'appeler directement dans le fichier de mise en données comme en témoigne la figure 3.1. En outre, `class REDUCED_MODEL_MANAGER` dispose par composition de trois attributs : `class REDUCED_MESH` permet de mettre en oeuvre concrètement les états *online* ou *offline* en contrôlant le maillage sur lequel opère le code éléments finis, `class REDUCED_BASIS_MANAGER` permet à partir des données du code éléments

Fonctionnalité	Interface
enregistrer le plugin sous le nom *hyperreduction*	`class REDUCED_MODEL_MANAGER`
prendre le contrôle en début ou en fin d'incrément	`class REDUCED_MODEL_MANAGER`
organiser le calcul d'hyperréduction à un haut niveau	`class REDUCED_MODEL_MANAGER`
estimer l'erreur ε entre le modèle réduit et complet	`class REDUCED_MODEL_STATE`
organiser la bascule entre le modèle réduit et complet	`class REDUCED_MODEL_STATE`
charger les maillages réduits et complets à jour	`class REDUCED_MESH`
organiser la mise en place des conditions aux limites relatives au maillage chargé	`class REDUCED_MESH`
organiser la construction du modèle réduit	`class REDUCED_BASIS_MANAGER`

Table 3.1 – Spécifications pour l'architecture du plugin d'hyperréduction

finis de construire le modèle réduit et `class REDUCED_MODEL_STATE` joue le rôle de bascule entre modèle réduit et modèle complet. Enfin cette spécialisation de `class PROBLEM_COMPONENT` permet de prendre le contrôle en début et fin d'incrément ainsi que souligné dans la section 3.1.

Bascule pour un modèle adaptatif Afin de pratiquer l'hyperréduction, il est nécessaire de développer une bascule qui permet le passage d'un calcul éléments finis complet à un calcul éléments finis réduit et réciproquement. Dans les deux cas, calcul réduit ou calcul complet, le même algorithme non linéaire est utilisé ce qui signifie que le fait de pratiquer l'hyperréduction est complètement indépendant de l'algorithme de résolution utilisé. Un autre peut y être substitué. La bascule s'effectue après s'être assuré de la convergence de la solution complète ou réduite, i. e. à la fin d'un incrément, quand ce dernier a été validé. Dépendant de l'indicateur d'erreur qui permet d'évaluer ε, dépendant du degré de précision ε_r retenu, dépendant de l'état présent - réduit ou complet - du processus, l'état futur peut être déterminé comme l'indique le diagramme de la figure 3.2. Si la précision souhaitée n'est pas atteinte, un calcul éléments finis complet est effectué afin d'enrichir à la volée le modèle réduit. En terme de fonctionnalités, la machine d'état est représentée figure 3.3 par la bascule d'installation du modèle réduit ou complet. Le plugin d'hyperréduction se contente d'installer les modèles ; il n'effectue pas le calcul qui est laissé au code éléments finis. Du point de vue de l'architecture logicielle, cette bascule est mise en oeuvre en utilisant le schéma de conception d'état présenté sur la figure 2.5 de la section 2.2. Sur le diagramme de classe de la figure 3.4 on peut retrouver aisément le schéma de conception 2.5 : `class REDUCED_MODEL_MANAGER` correspond à `class CONTEXT`, client de la classe `class REDUCED_MODEL_STATE` dont l'analogue est l'interface `class STATE` de 2.5. Deux spécialisations `class REDUCING_STATE` et `class ADAPTING_STATE` répondent aux `class CONCRETE_STATE_A` et `class CONCRETE_STATE_B` de la figure 2.5. C'est dans ces spécialisations que sont commandés les passages entre les modèles réduits et le modèle éléments finis complet.

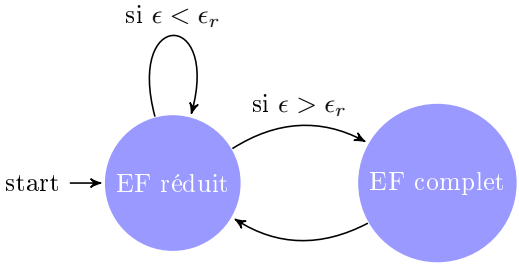


Figure 3.2 – Diagramme d'état de l'hyperréduction

Chargement du modèle réduit et complet Si la bascule commande le chargement du modèle approprié dans les routines éléments finis, le maillage réduit l'exécute. Pour cela, agissant comme un *handler*, il contient deux maillages : le maillage classique et une doublure de ce maillage qui contient des éléments réduits (fig. 3.4). Charger le modèle revient pour le maillage réduit à introduire dans le problème éléments finis le maillage souhaité - réduit ou complet - après s'être assuré que le modèle réduit est à jour et à déléguer le chargement des conditions aux limites réduites aux instances `class BC` et `class RELATIONSHIP` sous son contrôle comme en témoigne la figure 3.5.

Choix d'un modèle réduit `class REDUCED_BASIS_MANAGER` est une interface qui permet d'implémenter des spécialisations qui sont autant de manière de construire un modèle réduit. Nous avons mis en oeuvre une telle spécialisation `class DEFAULT_BASIS_MANAGER` comme l'illustre la figure 3.3.

3.1.1 Calcul hyperréduit

Les concepts fondamentaux d'un code éléments finis sont :

- les éléments ;
- les noeuds ;
- les degrés de liberté ;
- le maillage ;
- la matrice de rigidité ;
- les conditions aux limites (Dirichlet, Neumann, etc.)
- un algorithme (calcul de thermique, calcul de mécanique)

Il faut préciser quelques relations entre ces concepts :

- les noeuds contiennent systématiquement les degrés de liberté qui leur sont attachés ;
- les éléments contiennent l'ensemble des degrés de liberté présents en leur noeud ;
- les éléments sont capables d'associer à chaque degré de liberté le noeud auquel il réfère ;
- les conditions aux limites marquent les degrés de liberté concernés ;
- le maillage possède une liste de l'ensemble des éléments ;
- la matrice de rigidité est construite à partir du maillage.

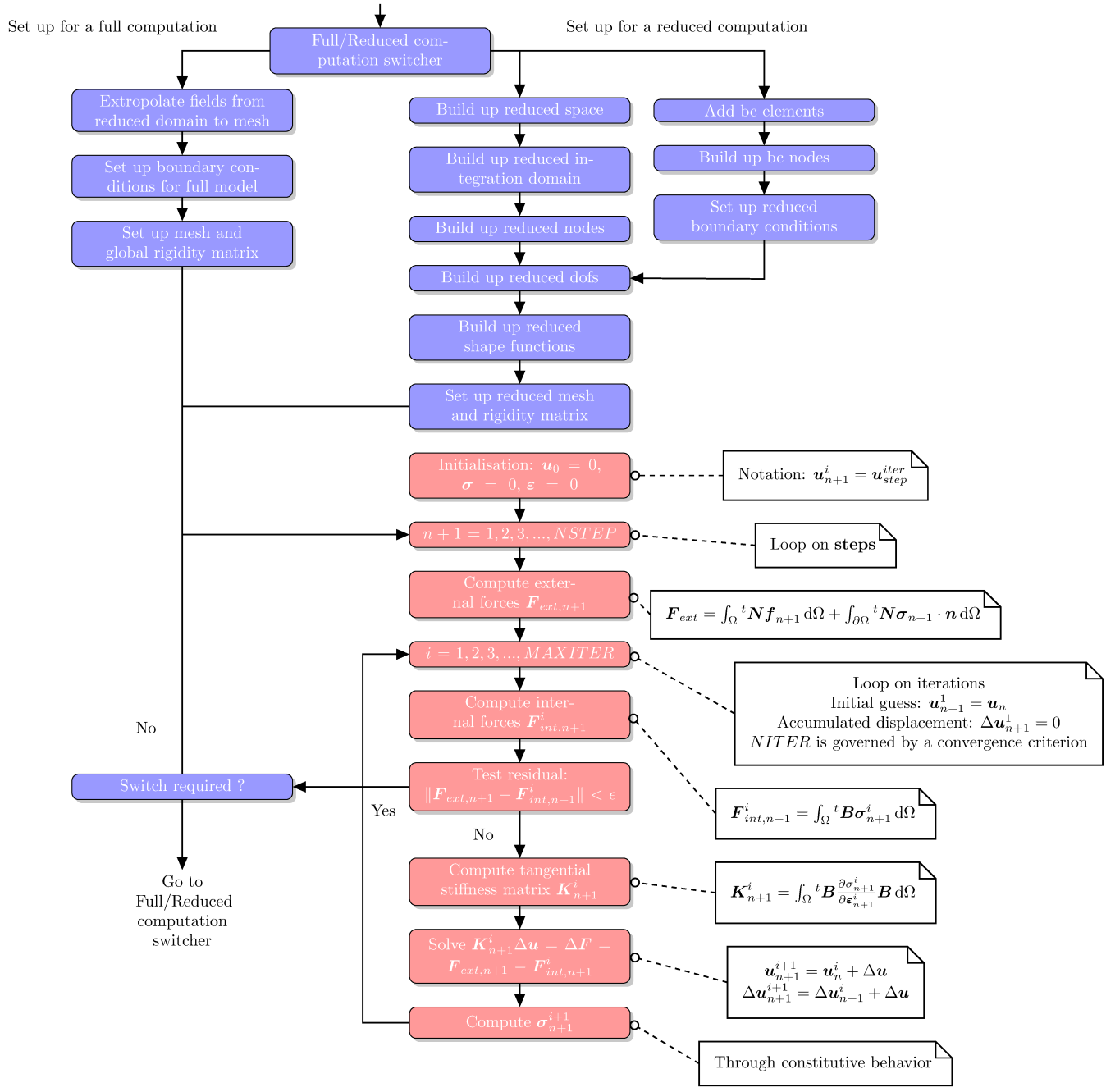


Figure 3.3 – Spécification pour l'algorithme d'hyperréduction. En rouge, les fonctionnalités de ZSET. En bleu, les fonctionnalités à développer.

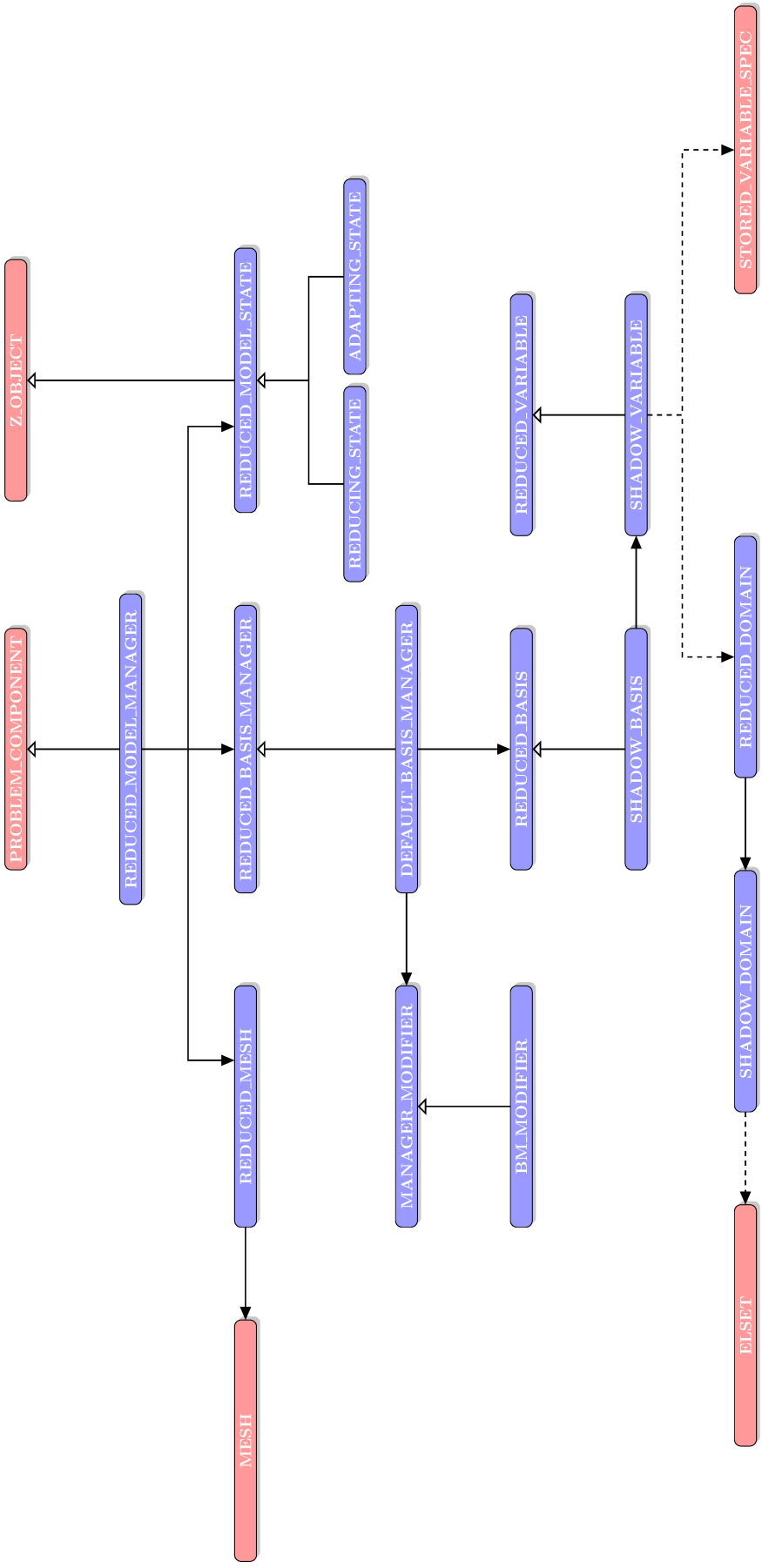


Figure 3.4 – Architecture orientée objet de l'hyperréduction. En rouge, les classes de ZSET. En bleu, les classes développées.

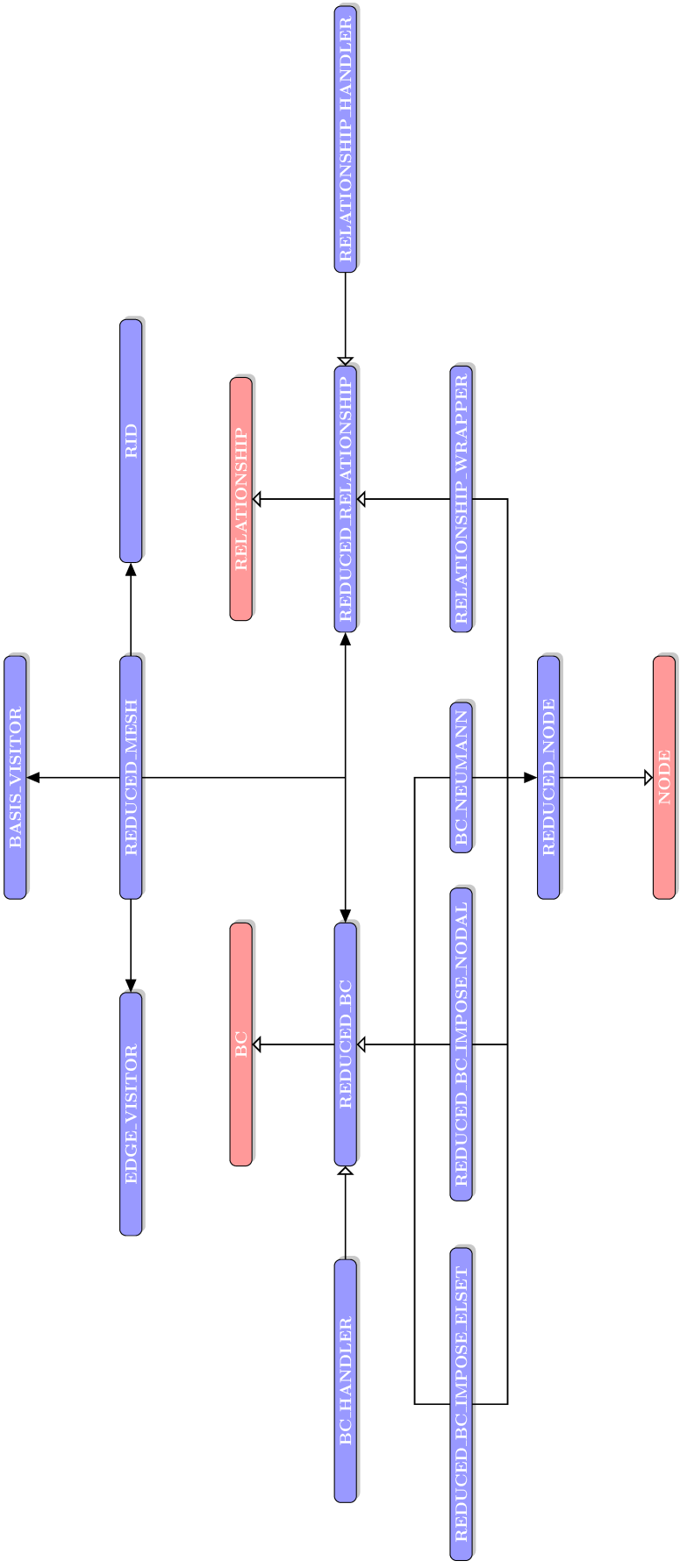


Figure 3.5 – Architecture orientée objet de l'hyperréduction. Conditions aux limites. En rouge, les classes de ZSET. En bleu, les classes développées.

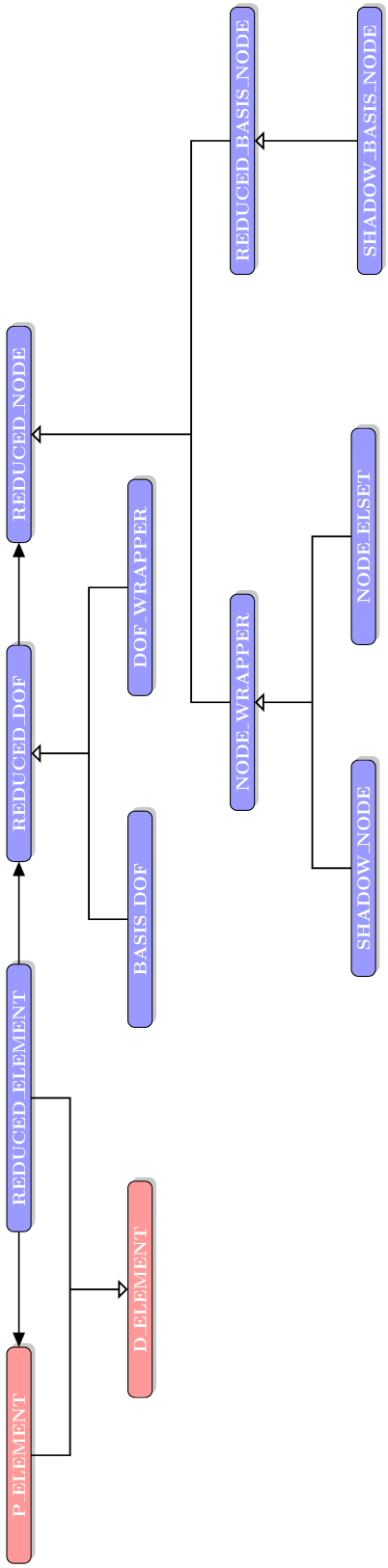


Figure 3.6 – Architecture orientée objet de l'hyperréduction. Eléments. Degrés de liberté. Noeuds. En rouge, les classes de ZSET. En bleu, les ajouts.

L'architecture proposée réinterprète l'ensemble de ces concepts utilisés en éléments finis en hyperréduction où chacun prend un sens différent. Il est ainsi possible de faire de l'hyperréduction en utilisant les possibilités d'un code éléments finis. L'organisation générale est fondée sur le concept de producteur-consommateur, très utilisé, par exemple, dans les problématiques de *multithreading* ([36], p. 87-88). Les éléments réduits tiennent lieu de produits et possèdent un cycle de vie. Ils sont consommés lors d'une phase *online* de calcul hyperréduit et servent comme outils pour atteindre l'équilibre mécanique ou thermique à faible coût. La consommation des éléments réduits est l'objet de la section 3.2. Avant d'être consommés, ils ont été produits lors d'une phase de calcul *offline* et stockés dans un maillage réduit. L'élaboration et la production des éléments réduits est coûteuse en ressources tandis que l'exploitation d'éléments adaptés pour résoudre l'équilibre mécanique est très peu gourmande. L'élaboration des éléments réduits est le sujet de la section 3.3. La pertinence d'un algorithme adaptatif hyperréduit en terme de diminution de coût de calcul tient à la capacité de réemploi des éléments réduits. Un indicateur d'erreur sert à contrôler leur cycle de vie.

3.2 Etape *online* : l'éléments réduit comme outil

Rappel : algorithme d'hyperréduction global Les architectures proposées généralement pour l'hyperréduction de modèle se fondent sur une modification de la procédure d'assemblage de la matrice de rigidité globale $\boldsymbol{K}$ et des réactions internes $\boldsymbol{R}$ impliquées dans la résolution itérative du système linéaire $\boldsymbol{K}\boldsymbol{u} = -\boldsymbol{R}$. Lors d'un calcul éléments finis classique, l'assemblage du système est effectué selon l'algorithme 7. Les implémentations des méthodes d'hyperréduction proposées se résument généralement à mettre en place l'algorithme 6. Seules les quelques lignes sélectionnées par une matrice $\boldsymbol{Z}$ revenant à ne retenir que la contribution d'un certain sous-ensemble d'éléments $\mathcal{M}_Z$ du système linéaire sont construites et donnent donc lieu à un assemblage partiel ${}^t\boldsymbol{Z}\boldsymbol{Z}\boldsymbol{K}$ et ${}^t\boldsymbol{Z}\boldsymbol{Z}\boldsymbol{R}$. Puis le système partiellement assemblé est projeté sur l'espace réduit $\boldsymbol{A}$ ce qui conduit au système linéaire hyperréduit global suivant qui est résolu à chaque itération :

$$ {}^t\boldsymbol{A}{}^t\boldsymbol{Z}\boldsymbol{Z}\boldsymbol{K}\boldsymbol{A}\delta\boldsymbol{u} = -{}^t\boldsymbol{A}{}^t\boldsymbol{Z}\boldsymbol{Z}\boldsymbol{R} \tag{3.1} $$

Proposition : un agorithme d'hyperréduction élémentaire Dans cette thèse, nous proposons non plus de travailler à l'échelle globale de la structure pour dégager une forme réduite ou hyperréduite mais à l'échelle locale, à l'échelle élémentaire. Nous avons mis en place la procédure d'assemblage décrite dans l'algorithme 2

Une telle mise en oeuvre présente plusieurs avantages :

- dans un code éléments finis où la brique élémentaire est l'élément, se concentrer sur l'assemblage de la matrice de rigidité et des réactions internes est sans doute se placer à une échelle moins pertinente que celle l'échelle élémentaire, beaucoup plus pertinente. En effet, cela permet d'utiliser l'ensemble des routines conçues pour des

: Assemblage des réactions internes et de la matrice de rigidité globale

input : $\mathcal{M}$

$\boldsymbol{K} \leftarrow 0$;

$\boldsymbol{F}_{int} \leftarrow 0$;

for $e \in \mathcal{M}$ **do**

  Compute $\int_\Omega {}^t\boldsymbol{B}_e {}^t\boldsymbol{\sigma}$;

  Compute $\int_\Omega {}^t\boldsymbol{B}_e \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\varepsilon}} \boldsymbol{B}_e$;

  $\boldsymbol{F}_{int} \leftarrow \boldsymbol{F}_{int} + \int_\Omega {}^t\boldsymbol{B}_e {}^t\boldsymbol{\sigma}$;

  $\boldsymbol{K} \leftarrow \boldsymbol{K} + \int_\Omega {}^t\boldsymbol{B}_e \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\varepsilon}} \boldsymbol{B}_e$;

output : $\boldsymbol{K}$ & $\boldsymbol{F}_{int}$

: Calcul des réactions internes réduites et de la matrice de rigidité globale réduite

for $e \in \mathcal{M}_Z$ **do**

  Compute $\int_\Omega {}^t\boldsymbol{B}_e {}^t\boldsymbol{\sigma}$;

  Compute $\int_\Omega {}^t\boldsymbol{B}_e \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\varepsilon}} \boldsymbol{B}_e$;

  $\boldsymbol{F}_{int} \leftarrow \boldsymbol{F}_{int} + \int_\Omega {}^t\boldsymbol{B}_e {}^t\boldsymbol{\sigma}$;

  $\boldsymbol{K} \leftarrow \boldsymbol{K} + \int_\Omega {}^t\boldsymbol{B}_e \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\varepsilon}} \boldsymbol{B}_e$;

$\boldsymbol{K} \leftarrow {}^t\boldsymbol{A}^t\boldsymbol{Z}\boldsymbol{Z}\boldsymbol{K}\ \boldsymbol{A}\ \boldsymbol{F}_{int} \leftarrow {}^t\boldsymbol{A}^t\boldsymbol{Z}\boldsymbol{Z}\boldsymbol{F}_{int}$

output : $\boldsymbol{K}$ & $\boldsymbol{F}_{int}$

Algorithme 2 : Algorithme proposé pour le calcul de l'hyperréduction

for $e \in \mathcal{M}_Z$ **do**

  Compute $\int_\Omega {}^t\boldsymbol{A}_e {}^t\boldsymbol{Z}_e \boldsymbol{Z}_e \boldsymbol{B}_e {}^t\boldsymbol{\sigma}$;

  Compute $\int_\Omega {}^t\boldsymbol{A}_e {}^t\boldsymbol{Z}_e \boldsymbol{Z}_e {}^t\boldsymbol{B}_e \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\varepsilon}} \boldsymbol{B}_e \boldsymbol{A}_e$;

  $\boldsymbol{F}_{int} \leftarrow \boldsymbol{F}_{int} + \int_\Omega {}^t\boldsymbol{A}_e {}^t\boldsymbol{Z}_e \boldsymbol{Z}_e {}^t\boldsymbol{B}_e {}^t\boldsymbol{\sigma}$;

  $\boldsymbol{K} \leftarrow \boldsymbol{K} + \int_\Omega {}^t\boldsymbol{A}_e {}^t\boldsymbol{Z}_e \boldsymbol{Z}^t \boldsymbol{B}_e \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\varepsilon}} \boldsymbol{B}_e \boldsymbol{A}_e$;

output : $\boldsymbol{K}$ & $\boldsymbol{F}_{int}$

calculs éléments finis classiques de façon complètement transparente pour le code cible ;

- une réduction de modèle opérant sur des parties seulement du maillage est plus aisée à mettre en oeuvre ;
- une implémentation dans d'autre code comme Abaqus en utilisant notamment le concept de *user-element* semble désormais moins lointaine. Dans Abaqus, on n'a pas accès à la matrice de rigidité globale mais on peut définir des éléments. Néanmoins les modalités et la faisabilité d'une telle implémentation restent encore étudier avec soin.
- en préstockant non plus seulement les bases réduites $\boldsymbol{A}$ mais les produits $\boldsymbol{B}A$ en chaque point de Gauss de chaque élément, on peut espérer enregistrer une diminution du coût d'assemblage *online*. En outre, cela ouvre la voie à une autre méthode de construction de bases réduites *offline*. Les constructions employées jusqu'à présent portent sur les bases de déplacements. Si les quantités d'intérêts lors de l'assemblage du modèle réduit sont les $\boldsymbol{B}A$ et non plus simplement les $\boldsymbol{A}$, cela signifie que l'on peut se préoccuper uniquement de construction de bases réduites $\boldsymbol{B}'$ portant sur les déformations. Mais, par ailleurs, puisque les déformations $\boldsymbol{\varepsilon}$ se décomposent classiquement en $\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}_p + \boldsymbol{\varepsilon}_e$ où $\boldsymbol{\varepsilon}_e$ est la déformation élastique et $\boldsymbol{\varepsilon}_p$ est la déformation plastique, il devient désormais possible d'intégrer dans les fonctions de forme de base réduite, des fonctions de forme plastiques et des fonctions de forme élastiques $\boldsymbol{B}' = \boldsymbol{B}'_e + \boldsymbol{B}'_p$, chose a priori moins évidente si l'on se limite à l'utilisation de base de déplacements, car il n'existe pas de déplacement plastique $\boldsymbol{u}$.

Dans la suite, nous allons présenter une mise en oeuvre de cet algorithme d'hyperréduction élementaire.

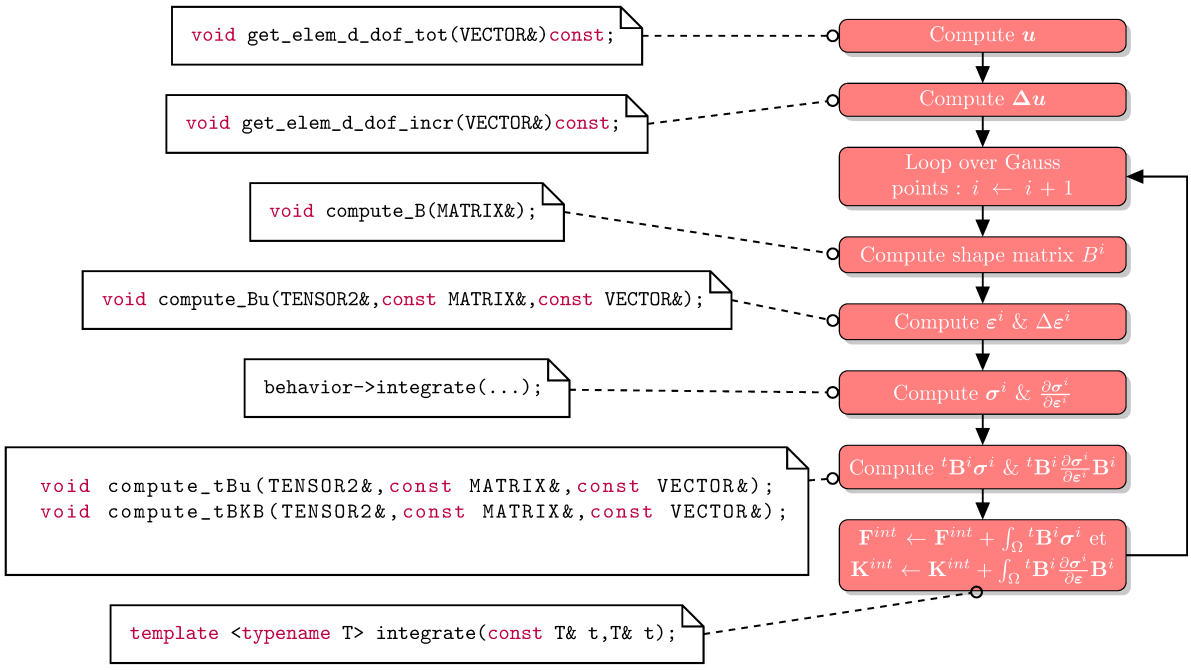


Figure 3.8 – Calcul des réactions internes

Rappels sur le calcul des réactions et de la matrice de rigidité élémentaires Nous rappelons le principe de la procédure d'assemblage élémentaire sur la figure 3.8. D'abord on effectue une projection des déplacements globaux $\boldsymbol{u}$ et aux incréments de déplacements globaux $\Delta\boldsymbol{u}$ sur des déplacements et incréments de déplacements locaux. Puis pour chaque point de Gauss de l'élément, on calcule la matrice $\boldsymbol{B}^i$ du gradient des fonctions de forme au point de Gauss i ce qui donne accès aux gradients des déformations $\boldsymbol{\varepsilon}^i$ et des incréments de déformation $\Delta\boldsymbol{\varepsilon}^i$. L'intégration de la loi de comportement permet d'en déduire les valeurs des contraintes $\boldsymbol{\sigma}^i$ et de la matrice tangente $\frac{\partial\boldsymbol{\sigma}^i}{\partial\boldsymbol{\varepsilon}^i}$. Vient ensuite l'étape de calcul des réactions internes locales $\boldsymbol{F}^{int}$ et $\boldsymbol{K}^{int}$ en additionnant la contribution de chaque point de Gauss. La figure 3.8 présente également les méthodes et les fonctions invoquées dans ZSET pour chacune des fonctionnalités. Ainsi donner un sens aux méthodes `compute_Bu(TENSOR2&, const MATRIX&,const VECTOR&, bool)`, `compute_Btu(VECTOR&, const MATRIX&, const TENSOR2&`, `compute_BtDB(MATRIX&, MATRIX&, const MATRIX&, const MATRIX&)`, `void get_elem_d_dof_tot(VECTOR&)const`, `void get_elem_d_dof_incr(VECTOR&)const` revient à créer un nouveau type d'élément en petites déformations en définissant la manière dont il contribue à l'équilibre mécanique global. C'est en conférant un sens particulier à ces cinq méthodes que nous définissons l'élément réduit.

Réactions et matrice de rigidité de l'élément réduit On distingue plusieurs types d'éléments réduits que l'on a rassemblés dans le tableau 3.2. L'unique différence entre le calcul des réactions internes et de la matrice de rigidité entre l'élément réduit ou hyperréduit et l'élément classique sous-jacent se situe au niveau du calcul de la matrice de fonction de forme $\boldsymbol{B}$ qui doit être remplacée par $\boldsymbol{B}'$ ou $\boldsymbol{B}''$ comme stipulé dans le tableau 3.2.

Afin d'illustrer les relations entre les éléments réduits et les éléments réels, sur la figure 3.11, on a représenté les déplacements aux noeuds d'un élément fini classique dans une structure

	Elément classique	Elément réduit	Elément hyperréduit
Matrice de rigidité	$\int_{\Omega^e} {}^t\boldsymbol{B}\dfrac{\partial\boldsymbol{\sigma}}{\partial\boldsymbol{\varepsilon}}\boldsymbol{B}\,\mathrm{d}\Omega$	$\int_{\Omega^e} \underbrace{{}^t\boldsymbol{A}{}^t\boldsymbol{B}}_{={}^t\boldsymbol{B}'}\dfrac{\partial\boldsymbol{\sigma}}{\partial\boldsymbol{\varepsilon}}\underbrace{\boldsymbol{B}\boldsymbol{A}}_{=\boldsymbol{B}'}\,\mathrm{d}\Omega$	$\int_{\Omega^e} \underbrace{{}^t\boldsymbol{A}{}^t\boldsymbol{S}{}^t\boldsymbol{B}}_{={}^t\boldsymbol{B}''}\dfrac{\partial\boldsymbol{\sigma}}{\partial\boldsymbol{\varepsilon}}\underbrace{\boldsymbol{B}\boldsymbol{A}}_{=\boldsymbol{B}'}\,\mathrm{d}\Omega$
Réaction interne	$\int_{\Omega^e} {}^t\boldsymbol{B}\boldsymbol{\sigma}\,\mathrm{d}\Omega$	$\int_{\Omega^e} {}^t\boldsymbol{B}'\boldsymbol{\sigma}\,\mathrm{d}\boldsymbol{\Omega}$	$\int_{\Omega^e} {}^t\boldsymbol{B}''\boldsymbol{\sigma}\,\mathrm{d}\Omega$
Rigidité	$\int_{\Omega^e} {}^t\boldsymbol{N}\dfrac{\partial f}{\partial u}\boldsymbol{N}\,\mathrm{d}\Omega$	$\int_{\Omega^e} \underbrace{{}^t\boldsymbol{A}{}^t\boldsymbol{N}}_{={}^t\boldsymbol{N}'}\dfrac{\partial f}{\partial u}\boldsymbol{N}\boldsymbol{A}\,\mathrm{d}\Omega$	$\int_{\Omega^e} \underbrace{{}^t\boldsymbol{A}{}^t\boldsymbol{S}{}^t\boldsymbol{N}}_{={}^t\boldsymbol{N}''}\dfrac{\partial f}{\partial u}\boldsymbol{N}\boldsymbol{A}\,\mathrm{d}\Omega$
Réaction	$\int_{\Omega^e} {}^t\boldsymbol{N}f\,\mathrm{d}\Omega$	$\int_{\Omega^e} {}^t\boldsymbol{N}'f\,\mathrm{d}\Omega$	$\int_{\Omega^e} {}^t\boldsymbol{N}''f\,\mathrm{d}\Omega$

Table 3.2 – Réactions internes et matrices de rigidité de certains éléments réduits

et sur la figure 3.12 les déformations induites. L'élément réduit paramétrise les déplacements qu'il astreint à évoluer dans un espace de faible dimension dont la projection sur l'élément est représenté sur la figure 3.13 par les pointillés. L'espace réduit est construit à partir des états passés de la structure. L'objet du calcul éléments finis réduit est de chercher dans cet espace réduit les déplacements satisfaisant à l'équilibre mécanique (fig. 3.14 et 3.12). A cela il faut ajouter un traitement spécifique pour les éléments du bord du domaine d'intégration réduit comme l'élément 1729 de la figure 3.9. A leurs noeuds frontière, l'équilibre n'est que partiellement calculé : la contribution des éléments gris (fig. 3.18) n'est volontairement pas prise en compte. En conclusion, l'élément hyperréduit est un élément non symétrique qui permet d'astreindre les déplacements dans un espace de faible dimension.

Sur le plan logiciel, l'élément réduit apparaît comme une spécialisation d'un `class D_ELEMENT` ayant contrôle sur un élément réel de type `P_ELEMENT`. Le schéma `class REDUCED_ELEMENT - class P_ELEMENT` peut être vu comme une application du schéma de conception *stratégie*, l'attribut de `class REDUCED_ELEMENT` gouvernant l'intégration de la classe hôte et la loi de comportement est intégrée dans le `P_ELEMENT` sous-jacent.

La programmation de l'élément réduit, c'est la façon dont on va surcharger les fonctions de forme de l'élément réel sous-jacent, pour le faire calculer avec ces nouvelles fonctions de forme, de manière complètement transparente pour lui, les réactions internes de l'élément réduit qui lui correspond.

De surcroît, la programmation des éléments réduits et hyperréduits se fait en tenant compte des spécifications suivantes :

- être complètement transparent à l'utilisateur ;
- être isolé de l'atelier de création de modèle réduit ;

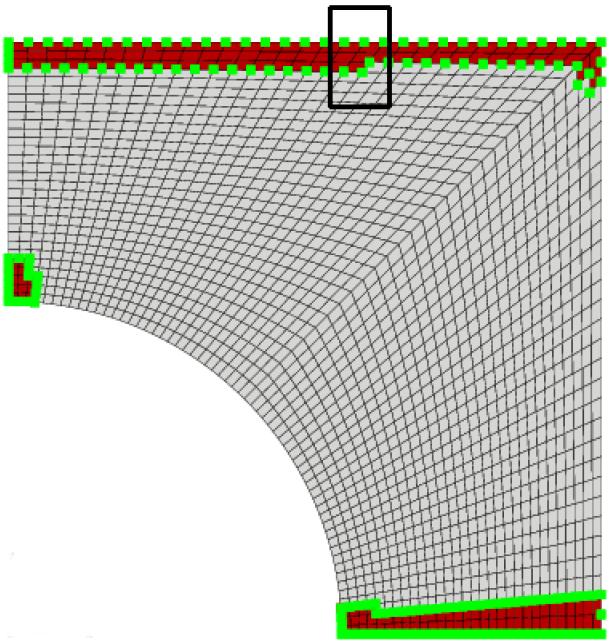


Figure 3.9 – Domaine d'intégration réduit (RID)

- être le moins intrusif possible dans ZSET ;
- utiliser les procédures d'assemblage de matrice de rigidité élémentaire et de calcul des réactions internes déjà existantes ;
- être la plus concise possible.

Le calcul des réactions internes $\int_\Omega {}^t\boldsymbol{B}\boldsymbol{\sigma}$ des éléments ainsi que celui de la matrice de rigidité locale $\int_\Omega {}^t\boldsymbol{B}\boldsymbol{K}\boldsymbol{B}$ est effectué par la méthode `internal_reaction(...)`. La modification du comportement de cette méthode d'intégration est possible en plaçant des *callbacks* sur un certain nombre de méthodes auxiliaires qu'elle appelle :

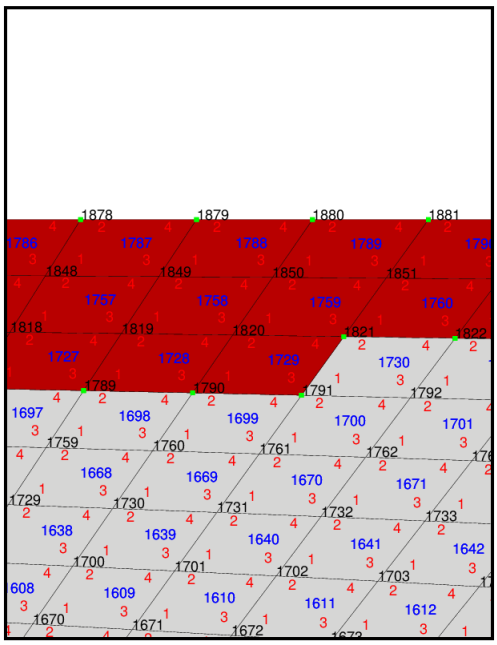


Figure 3.10 – Domaine d'intégration réduit (RID)

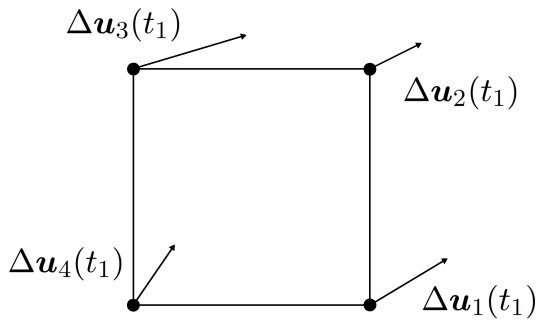


Figure 3.11 – Déplacement des noeuds de l'élément réel

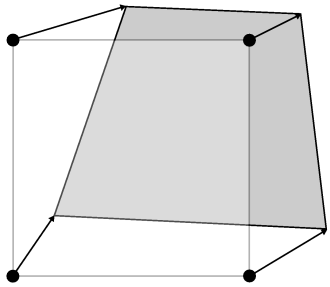


Figure 3.12 – Déformation de l'élément réel

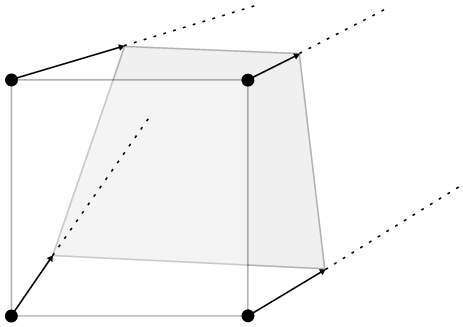


Figure 3.13 – Création de l'espace des déplacements réduits

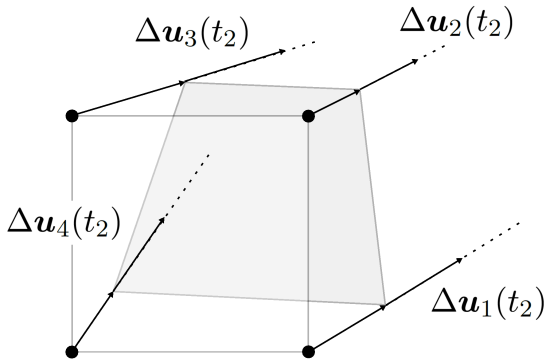


Figure 3.14 – Déplacement des noeuds de l'élément réel sous-jacent à l'élément réduit

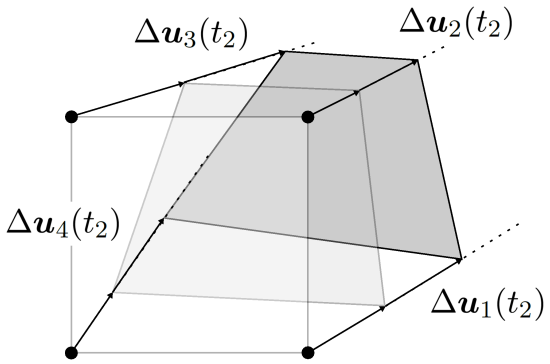


Figure 3.15 – Déformation de l'élément réel sous-jacent à l'élément réduit

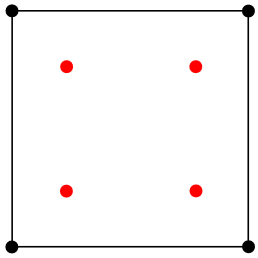


Figure 3.16 – un `P_ELEMENT`

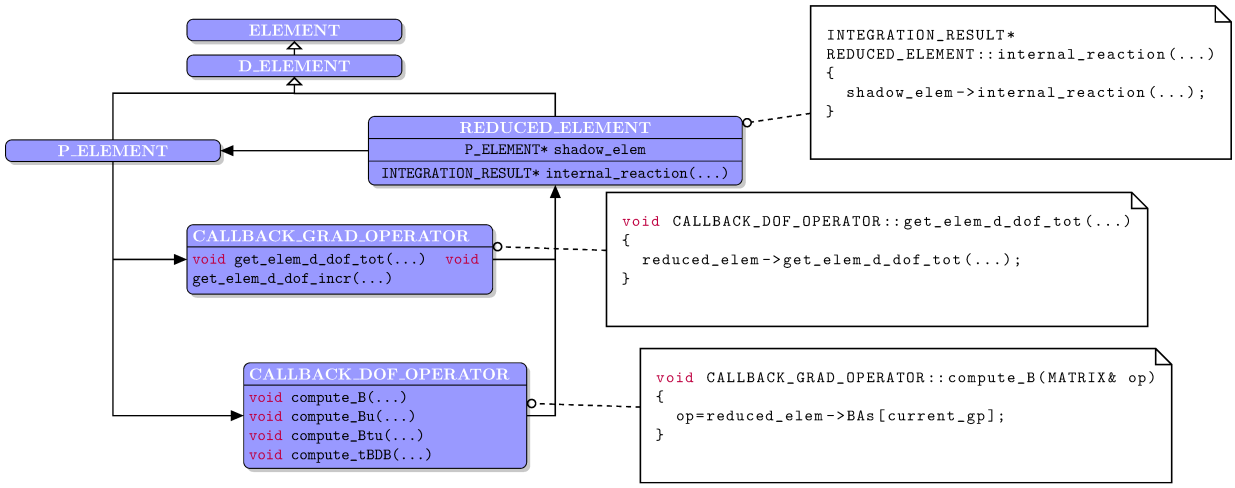


Figure 3.17 – Hiérarchie de l'élément réduit

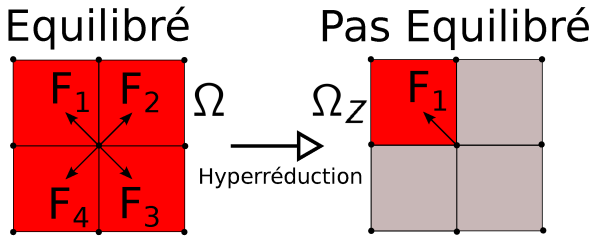


Figure 3.18 – Perte d'équilibre sur les éléments frontière lors de l'hyperréduction

- `compute_Bu(...)`. Cette méthode effectue la substitution : $\boldsymbol{B} \leftarrow \boldsymbol{BA}$ au moment du calcul des réactions internes de l'élément sous-jacent.
- `compute_Btu(...)`. Cette méthode effectue la substitution : ${}^t\boldsymbol{Bu} \leftarrow {}^t\boldsymbol{A}{}^t\boldsymbol{Z}\boldsymbol{Z}{}^t\boldsymbol{Bu}$ lors du calcul des réactions internes de l'élément sous-jacent.
- `compute_BtDB(...)`. Cette méthode effectue la substitution : ${}^t\boldsymbol{BDB} \leftarrow {}^t\boldsymbol{A}{}^t\boldsymbol{Z}\boldsymbol{Z}{}^t\boldsymbol{BDBA}$ lors du calcul des réaction internes de l'élément sous-jacent.

Ces trois méthodes regroupent les opérations élémentaires nécessaires au calcul de la matrice de rigidité élémentaire et aux réactions internes. Si ces trois opérations élémentaires sont nécessaires, le calcul et la connaissance des déplacements et des incréments de déplacements est également nécessaire pour interroger la loi de comportement. Ceci est réalisé par les deux méthodes suivantes :

- `get_elem_d_dof_tot(...)` qui permet de remplacer $\boldsymbol{u} \leftarrow \boldsymbol{Aq}$, i. e. de paramétrer $\boldsymbol{u}$
- `get_elem_d_dof_incr(...)` qui permet de remplacer $\Delta\boldsymbol{u} \leftarrow \boldsymbol{A}\Delta\boldsymbol{q}$, i. e. de paramétrer $\Delta\boldsymbol{u}$

Donner un sens à ces cinq méthodes auxiliaires, c'est créer un nouveau type d'élément en définissant la manière dont il contribue à l'équilibre mécanique global. Ainsi, c'est en conférant un sens particulier à ces cinq méthodes que nous obtenons l'élément réduit.

Après avoir décrit comme l'élément réduit est utilisé et consommé, nous abordons maintenant le processus d'élaboration d'un tel élément.

Fonctionnalité *online* de l'élément réduit	**Interface**
$\boldsymbol{B \leftarrow BA}$	`compute_Bu(...)`
${}^t\boldsymbol{Bu} \leftarrow {}^t\boldsymbol{A}{}^t\boldsymbol{Z}\boldsymbol{Z}{}^t\boldsymbol{Bu}$	`compute_Btu(...)`
${}^t\boldsymbol{BDB} \leftarrow {}^t\boldsymbol{A}{}^t\boldsymbol{Z}\boldsymbol{Z}{}^t\boldsymbol{BDBA}$	`compute_BtDB(...)`
$\boldsymbol{u \leftarrow Aq}$	`get_elem_d_dof_tot(...)`
$\boldsymbol{\Delta u \leftarrow A\Delta q}$	`get_elem_d_dof_incr(...)`

Table 3.3 – Méthodes de l'élément réduit

3.3 Etape *offline* : l'élément réduit comme produit

L'élaboration d'un élément hyperréduit se fait en plusieurs étapes que nous allons examiner successivement dans la présente section :

- l'élaboration d'un espace réduit global ce qui implique la collecte des données à l'aide d'une variable réduite sur un domaine donné, le domaine réduit, et leur synthèse réalisée dans une base réduite ;
- la création des degrés de liberté réduits à partir de l'espace réduit et des conditions aux limites ;
- le calcul des fonctions de forme hyperréduites par projection de l'espace réduit sur l'élément réduit.

3.3.1 Les domaines réduits

Dans ZSET, une classe `class ELSET` offre les fonctionnalités pour manipuler un ensemble d'éléments. Ces fonctionnalités sont insuffisantes pour pouvoir l'assimiler à un domaine réduit, i. e. sur lequel on peut construire un modèle réduit. En effet, les objets d'intérêt ne sont pas tant les éléments que leurs degrés de liberté ou les valeurs aux points de Gauss qu'ils contiennent. Du point de vue de la structure de données, un modèle est un couple $\mathcal{E} \times \mathcal{N}$. Sur la figure 3.19 est représenté en rouge un ensemble de 4 éléments destinés à la création d'un domaine réduit dont les degrés de liberté d sont numérotés ainsi que les points de Gauss m. Un objet de type `class REDUCED_DOMAIN` est une classe paramétrée par deux politiques (section 2.3.1) qui permettent de faire varier son comportement en fonction de ce qu'il s'agit d'un domaine de degrés de liberté, de points de Gauss ou qu'il soit ouvert ou fermé. On remarque que les politiques sont orthogonales. Le tableau 3.4 récapitule les conséquences de l'ajout d'un élément au domaine réduit par la méthode `add`. Si le domaine est un domaine de degrés de liberté et qu'il est ouvert, les degrés de liberté ajoutés sont ceux qui sont strictement partagés par l'ensemble des éléments ajoutés, i. e. dans l'exemple d_{17} et d_{18}. Ainsi les degrés de liberté d_{15} et d_{16} connectés à des éléments extérieurs au domaine ne sont pas ajoutés. Quand le domaine est fermé, tous les degrés de liberté qui appartiennent à un élément ajouté sont inclus. d_{17} et d_{18} sont inclus dans le domaine de degrés de liberté fermé. Pour les domaines de point de Gauss, la différence entre fermé et ouvert est indiscernable et

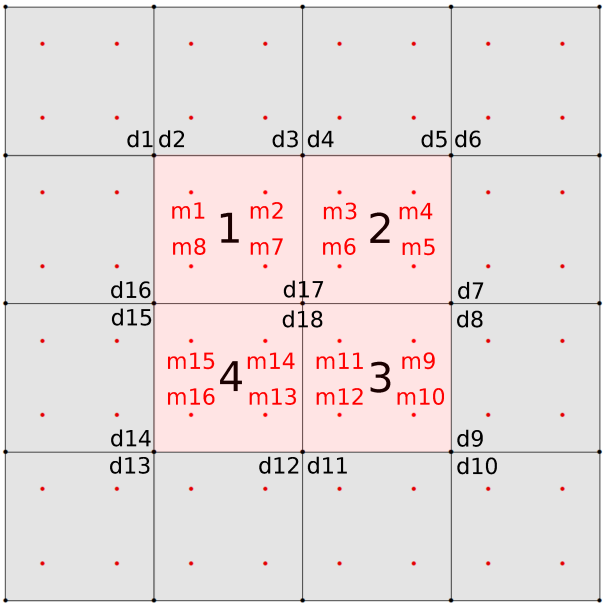


Figure 3.19 – Domain réduit. Degrés de liberté. Points de Gauss.

`add(e1); add(e2);` `add(e3); add(e4);`	Ouvert	Fermé
Domaine de degrés de liberté	$\{d_{17}, d_{18}\}$	$\{d_1, d_2, d_3, d_4, d_5, d_6,$ $d_7, d_8, d_9, d_{10}, d_{11}, d_{12},$ $d_{13}, d_{14}, d_{15}, d_{16}, d_{17}, d_{18}\}$
Domaine de points de Gauss	$\{m_1, m_2, m_3, m_4,$ $m_5, m_6, m_7, m_8,$ $m_9, m_{10}, m_{11}, m_{12},$ $m_{13}, m_{14}, m_{15}, m_{16}\}$	$\{m_1, m_2, m_3, m_4,$ $m_5, m_6, m_7, m_8,$ $m_9, m_{10}, m_{11}, m_{12},$ $m_{13}, m_{14}, m_{15}, m_{16}\}$

Table 3.4 – Ajout d'un point de Gauss ou d'un degré de liberté au domaine réduit

l'ensemble des points de Gauss situés dans les éléments ajoutés sont inclus. A titre d'exemple, le domaine d'intégration réduit apparaît ici comme un domaine de degrés de liberté ouvert et il est implémenté comme tel.

3.3.2 Les grandeurs mécaniques dans Zébulon

Le modèle réduit est construit à partir de grandeurs mécaniques qui se situent :

- aux points de Gauss ;
- aux noeuds.

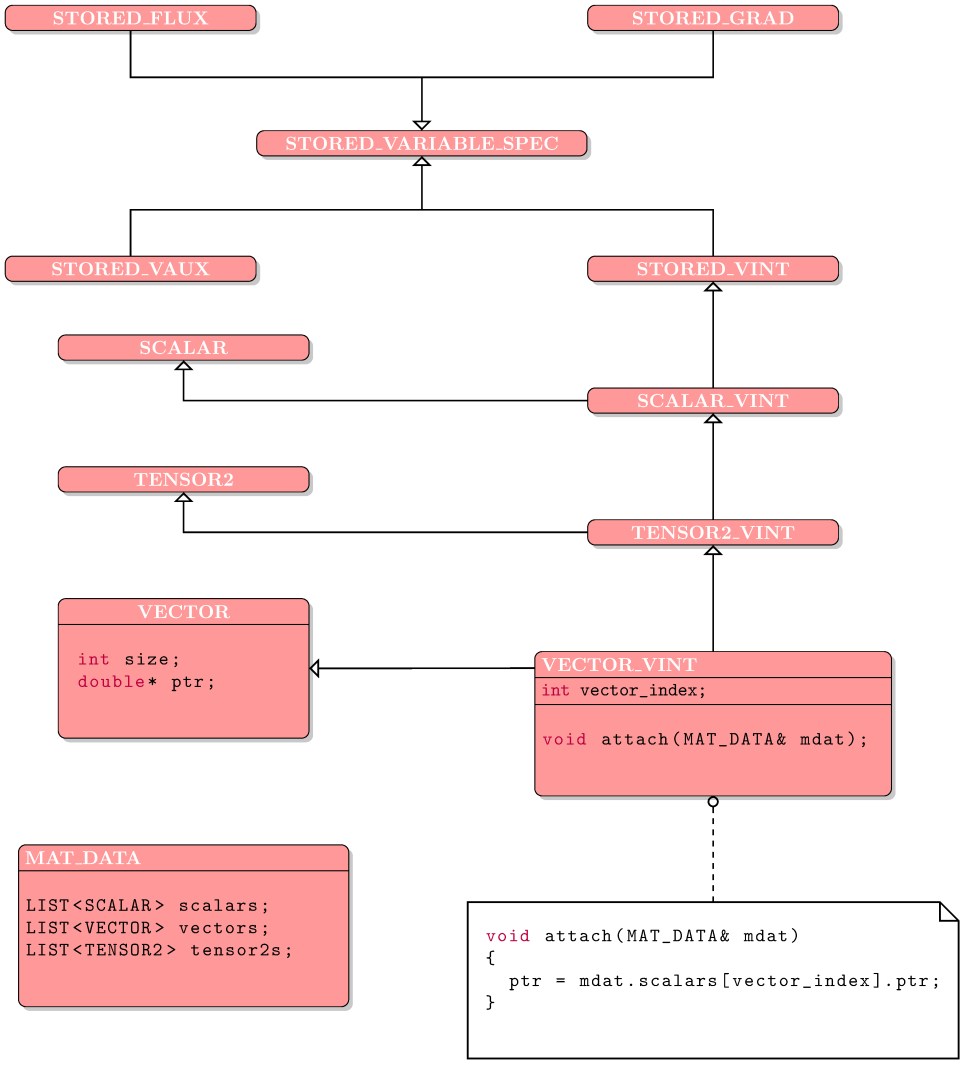


Figure 3.20 – Interface pour les variables mécaniques dans ZSET

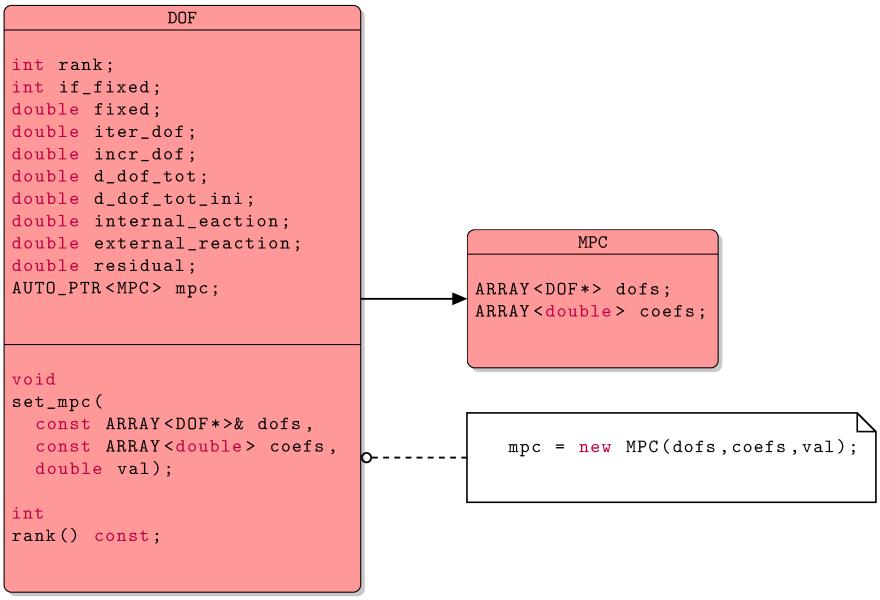


Figure 3.21 – Diagramme de `class DOF`

Dans la suite, on décrit la manière dont sont gérés les deux types de variables dans ZSET. Dans [29] p. 140 - 142 se trouve une présentation des principes de l'implémentation dans Zébulon. Mais il nous semble utile de fournir quelques détails qui viennent compléter ces explications.

On s'intéresse d'abord aux variables mécaniques stockées aux points de Gauss. Afin de faciliter la gestion et d'accéder à un niveau d'expressivité supérieure, on sépare :

- le lieu où les valeurs de ces variables sont stockées, typiquement `class MAT_DATA`. Il y a en général une instance de `class MAT_DATA` par point de Gauss, mais pour des besoins spécifiques, plusieurs peuvent être imaginées.
- l'interface par laquelle on manipule ces variables et qui permet de spécifier les caractéristiques d'une variable, typiquement `class STORED_VARIABLE_SPEC`. Une loi matériau est un ensemble de telles variables impliquées dans un système différentiel en temps.

Les caractéristiques d'une variable peuvent être classées en deux catégories :

- la nature de la quantité mécanique qu'elle représente, i. e. une variable interne `class STORED_VINT`, une variable auxiliaire `class STORED_VAUX`, une contrainte `class STORED_FLUX` , une déformation `class STORED_GRAD` ou d'autres cas plus complexes utilisés, par exemple, dans les calculs EF^2. Toutes ces variables dérivent de l'interface `class STORED_VARIABLE_SPEC` ainsi que présenté figure 3.20.
- le type mathématique de la variable, i. e. un scalaire `class SCALAR`, un vecteur `class VECTOR` ou un tenseur `class TENSOR2`.

On représente ainsi très logiquement la contrainte $\boldsymbol{\sigma}$ comme `class TENSOR2_FLUX`, la déformation $\boldsymbol{\varepsilon}$ comme `class TENSOR2_GRAD` en utilisant la notion très controversée d'héritage multiple (fig. 3.20).

Ayant maintenant défini une interface pour manipuler les variables, il reste à attacher une valeur aux variables que l'on manipule. Pour l'instant cette valeur est indéfinie. On va puiser ces valeurs dans les instances de `class MAT_DATA` et c'est ce que propose la méthode `attach`. Ce que fait `attach` est assez subtil. `attach` fait pointer l'instance de `class TENSOR2_FLUX` qui est un vecteur de taille `size` et de pointeur `ptr` sur le vecteur correspondant à l'indice `vector_index` de même taille de la liste où résident les valeurs de la variable interne. C'est le mécanisme illustré figure 3.20. Ainsi, après avoir appelé la méthode `attach`, l'instance de type `class VECTOR_VINT` contient la valeur de la contrainte au point de Gauss relatif à l'instance de `class MAT_DATA`.

Cette architecture permet un niveau d'expressivité supérieure en ce qu'il existe une unique instance variable par loi matériau alors que ces variables prennent des valeurs différentes en chaque point de Gauss. Manipuler un unique nom par variable indépendamment du point de Gauss est sans conteste une grande source de simplification.

De même que les éléments contiennent un ensemble de pointeurs vers les instances des `class MAT_DATA` aux points de Gauss, de même les noeuds `class NODE` contiennent un ensemble de pointeurs vers des degrés de libertés `class DOF`. C'est dans les instances de la `class DOF` que sont stockées les valeurs de déplacement total, d'incrément de déplacement comme représenté figure 3.21. Ce sont de simples attributs publics, sans accesseur, ce qui constitue une limitation au code qui ne permet pas de contrôler la lecture de ces variables. La `class DOF` peut posséder un attribut de type `class MPC` qui permet d'écrire des relations de liaison entre

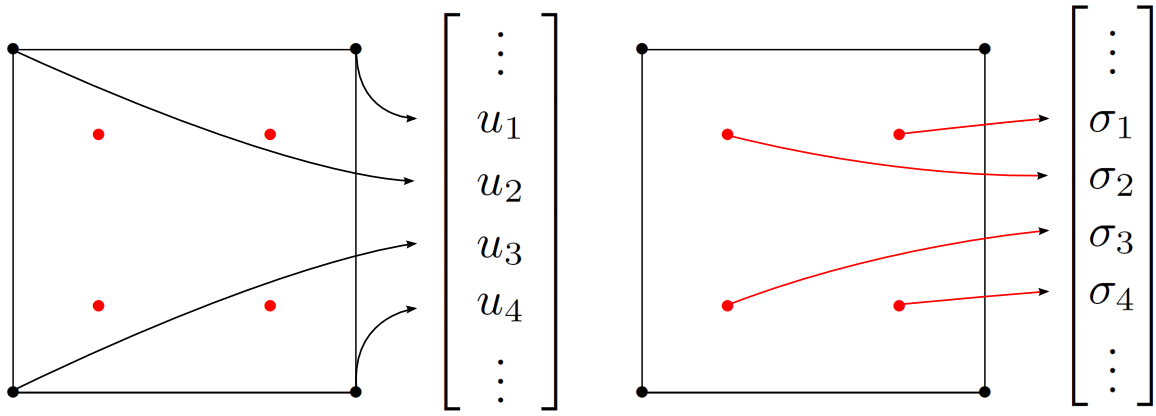


Figure 3.22 – Rôle d'une variable réduite

l'instance du degré de liberté actuel et d'autres, ce qui est utile lorsque l'on veut par exemple écrire des relations de périodicité entre degrés de liberté.

3.3.3 Les variables réduites

Spécifications Une variable réduite doit remplir les deux conditions générales suivantes :

- Elle doit être suffisamment générique en permettant le traitement des variables nodales stockées dans les degrés de liberté que sont les incréments de déplacements $\Delta\boldsymbol{u}$, la valeur totale des déplacements $\boldsymbol{u}$, les forces internes $\boldsymbol{F}^{int}$, le résidu $\boldsymbol{R}$; mais également en permettant le traitement des variables aux points de Gauss, i. e. la valeur locale du champ de contrainte $\boldsymbol{\sigma}$, du champ de déformation $\boldsymbol{\varepsilon}$, des variables internes comme la plasticité cumulée p, le tenseur des déformations plastiques $\boldsymbol{\varepsilon}_p$, l'endommagement d, etc.
- Elle doit être suffisamment optimisée, car on traite une grande quantité de valeurs de variables et une analyse de performance sur le précédent code de l'hyperréduction mise en oeuvre dans ZSET avait révélé qu'il s'agissait d'un réel goulot d'étranglement.

Prenant en compte toutes ces contraintes et les autres spécifications requises table 3.5, on obtient l'interface suivante - celle-là même qui est écrite dans le plugin - :

```
struct REDUCED_VARIABLE                                  1
{                                                        2
    virtual void                                         3
    set_up_for_reduction()=0;                            4
                                                         5
    virtual void                                         6
    set_up_for_hyperreduction()=0;                       7
                                                         8
    virtual void                                         9
    set_up_for_adaptation()=0;                           10
                                                         11
    virtual STRING                                       12
    get_name() const=0;                                  13
```

Fonctionnalité	Prototype de la méthode
prélever la valeur du champ sur l'ensemble de la structure	`VECTOR get_field()`
prélever la valeur du champ seulement sur le domaine d'intégration réduit	`VECTOR get_field(const RID&)`
réinjecter la valeur du champ sur l'ensemble de la structure dans le code éléments finis cible	`void set_field(const VECTOR&)`
renseigner sur le nom de la variable. Ex : $\boldsymbol{\sigma}$, $\boldsymbol{\varepsilon}$, d.	`STRING get_name()`
renseigner sur la taille du champ. Ex : champ scalaire (1), champ vectoriel 2D (2), champ tensoriel ordre 2 3D (6).	`int get_size()`
implémenter les 3 fonctionnalités de structure de l'hyperréduction	`void set_up_for_reduction()` `void set_up_for_hyperreduction()` `void set_up_for_adaptation()`

Table 3.5 – Spécifications pour la variable réduite

```
    virtual int
    get_size() const=0;

    virtual VECTOR
    get_field() const=0;

    virtual VECTOR
    get_field(const RID&) const=0;

    virtual void
    set_field(const VECTOR&)=0;

    virtual double
    get_magnitude() const=0;
};
```

Listing 3.1 – Interface des variables réduites

Mise en oeuvre Elle consiste à proposer une spécification pour ZSET de cette interface. L'enjeu de la mise en oeuvre proposée est de prendre en compte la concision et la lisibilité. Nous proposons l'architecture représentée sur la figure 3.23 [i].

i. En raison d'un manque de temps, une dernière factorisation manque pour parvenir à l'architecture canonique proposée. En revanche les fonctionnalités de l'interface sont bien sûr rigoureusement toutes implémentées

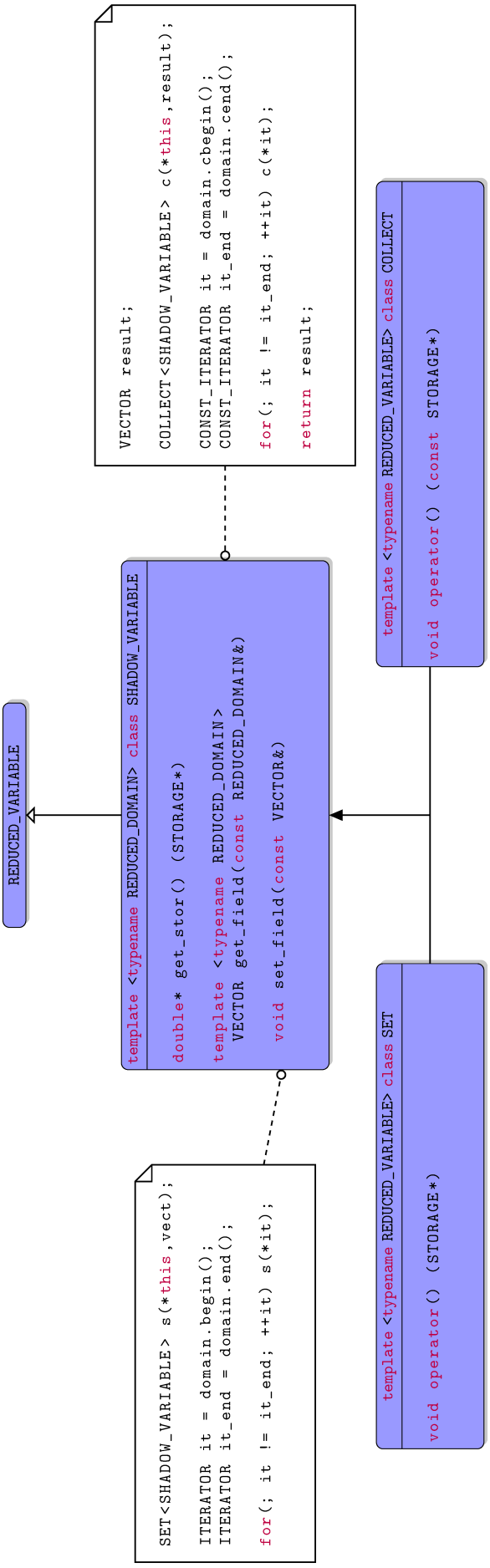


Figure 3.23 – Proposition d'architecture pour l'implémentation des variables réduites.

3.3.4 Bases réduites

Une base réduite est, avec le domaine réduit, une des composantes du modèle réduit. Très concrètement, il s'agit seulement d'une matrice un peu évoluée. L'interface de la base réduite est présentée dans 3.2. Elle compte des méthodes permettant de déterminer quelques caractéristiques de la base, `STRING get_name()const` qui permet d'obtenir le nom de la base et `int get_dim()` sa dimension ; des méthodes comme `const MATRIX get_field(const RID&)` ou `const MATRIX get_field()` ou bien encore `const VECTOR get_vector(int)` utiles pour extraire des informations du champ de base sur un domaine donné pour la première ou sur l'ensemble du support de base pour les deux dernières ; des méthodes permettant de modifier l'état de l'objet intégralement comme `void set_field(const MATRIX&)` ou partiellement comme `void add (const VECTOR&)` ; des méthodes dont le but est d'obtenir des informations sur le support de la base, comme `BUFF_LIST<ELEMENT*> get_elems(const BUFF_LIST<int>&)` capable de renvoyer un ensemble d'éléments correspondant à un numéro de degrés de liberté ou de point de Gauss lui appartenant ou `BUFF_LIST<ELEMENT*> get_support(const RID&)` qui renvoie l'ensemble des éléments intersection du domaine d'intégration réduit et du support de base ; enfin, des méthodes servant à orchestrer la conception du modèle réduit telles `void set_up_for_reduction()`, `void set_up_for_hyperreduction` et `void set_up_for_adaptation()`. Fournir une implémentation pour l'ensemble de ces méthodes revient à spécifier un objet de type base réduite qui peut être utilisé par les autres composants du code.

```
class REDUCED_BASIS
{
public:

    virtual void
    set_up_for_reduction()=0;

    virtual void
    set_up_for_hyperreduction()=0;

    virtual void
    set_up_for_adaptation()=0;

    virtual REDUCED_VARIABLE&
    get_reduced_var() const=0;

    virtual void
    operator()(int,int,int,int,MATRIX&) const=0;

    virtual STRING
    get_name() const=0;

    virtual int
    get_dim() const=0;

    virtual const MATRIX
    get_field() const=0;

    virtual const MATRIX
    get_field(const RID&) const=0;
```

Fonctionnalité	Symbole de la méthode
prélever la valeur de la base sur l'ensemble de la structure	`MATRIX get_field()`
prélever la valeur de la base seulement sur le domaine d'intégration réduit	`MATRIX get_field(const RID&)`
obtenir le support de la base	`BUFF_LIST<ELEMENT*> get_support()`
obtenir l'ensemble des éléments contenus dans le support de la base et le domaine d'intégration réduit	`BUFF_LIST<ELEMENT*> get_support()`
obtenir l'ensemble des éléments qui contiennent certains degrés de liberté ou points de Gauss repérés par leur numéro	`BUFF_LIST<ELEMENT*> get_support(const BUFF_LIST< int>&)`
obtenir le ième vecteur de base	`VECTOR get_vector(int i)`
modifier la base sur l'ensemble de la structure	`void set_field(const MATRIX&)`
ajouter un vecteur à la base	`void add(const VECTOR&)`
renseigner sur le nom de la base. Ex : $\boldsymbol{\sigma}$, $\boldsymbol{\varepsilon}$, d.	`STRING get_name()`
renseigner sur la dimension de la base.	`int get_dim()`
implémenter les 3 fonctionnalités de structure de l'hyperréduction	`void set_up_for_reduction()` `void set_up_for_hyperreduction ()` `void set_up_for_adaptation ()`

Table 3.6 – Spécifications pour la base réduite

```
                                                                    31
    virtual BUFF_LIST<ELEMENT*>                                     32
    get_elems(const BUFF_LIST<int>&) const=0;                       33
                                                                    34
    virtual BUFF_LIST<ELEMENT*>                                     35
    get_support(const RID&) const=0;                                36
                                                                    37
    virtual BUFF_LIST<ELEMENT*>                                     38
    get_support() const=0;                                          39
                                                                    40
    virtual const VECTOR                                            41
    get_vector(int) const=0;                                        42
                                                                    43
    virtual void                                                    44
    set_field(const MATRIX&)=0;                                     45
                                                                    46
    virtual void                                                    47
    add(const VECTOR&)=0;                                           48
                                                                    49
};                                                                  50
```

Listing 3.2 – Interface des bases réduites

Implémentation d'une base réduite La principale fonctionnalité recherchée consiste à parcourir la base réduite sur un domaine réduit de manière efficace. La proposition d'implémentation est illustrée figure 3.24. Elle se fonde sur le concept d'itérateur comme présenté sur la figure 2.7 et l'implémentation conjointe d'une méthode d'extraction du champ de base réduit `double** get_stor(STORAGE*)` qui à un degré de liberté ou point de Gauss fait correspondre la valeur du champ réduit. L'itérateur déréférencé donne le degré de liberté ou le point de Gauss courant et l'appel à la méthode `double** get_stor(STORAGE*)` donne la valeur du champ de base réduit en ce point de Gauss. C'est sur ces fonctionnalités élémentaires qu'est construite la spécification de l'interface de base réduite en particulier les méthodes de modification des données et de collecte.

Synthèse des données La synthèse des données collectées débouche sur la création d'une base et d'un domaine réduit. Si, dans la communauté, l'accord règne sur la manière de mener un calcul hyperréduit sur base et domaine réduits, la manière d'élaborer ces bases et ce domaine réduit est loin de faire l'unanimité et il existe toute une série de variations. Par conséquent, il est primordial de s'assurer de pouvoir élaborer facilement un modèle réduit. Il est du coup important de découpler la gestion de la base réduite qui s'assimile à une matrice de la transformation de la base qui s'effectue dans des classes, qui sont en fait des foncteurs. L'élaboration de ce modèle se fait dans un atelier de construction. A chaque fois qu'un développeur souhaitera créer un nouveau modèle réduit, il devra créer une spécialisation de l'atelier de construction.

Utilisation d'un décorateur La mise à jour des bases réduites s'effectue à l'aide d'un schéma de type décorateur (voir 2.2). Un décorateur de base réduite est une spécialisation de base réduite. Ainsi sur la figure 2.6 la `class COMPONENT` correspond à `class REDUCED_BASIS`, la `class DECORATOR` correspond à `class BASIS_DECORATOR` et ses spécialisations à des décorateurs pour les bases de variables internes ou pour les bases de déplacements. La méthode `Operation()` de la figure 2.6 correspond ici aux trois méthodes :

- `void set_up_for_reduction()` qui met à jour les bases réduites en utilisant une ou plusieurs méthodes de transformations de base réduite exposées figure 3.24 ;
- `void set_up_for_hyperreduction()` qui construit les noeuds réduits qui vont être partagés par les éléments réduits si la base réduite est une base de déplacements ;
- `void set_up_for_adaptation()`. Quand une adaptation du modèle réduit est requise, il est nécessaire d'extrapoler les bases de variables internes du domaine d'intégration réduit à l'ensemble du domaine. Nous précisons au point suivant comment nous procédons.

Extrapolation Une fois que la résolution de l'équilibre hyperréduit est effectuée, une estimation des champs est connue sur le domaine réduit. Il est possible d'avoir une carte des champs sur l'ensemble de la struture en les estimant à partir de leurs valeurs sur le domaine réduit : on pratique pour cela une extrapolation. On utilise la méthode des moindres carrés comme illustré sur la figure 3.25 :

$$\hat{\boldsymbol{\alpha}} = \arg\max_{\boldsymbol{\alpha}} \|\tilde{\boldsymbol{\sigma}} \cdot \boldsymbol{\alpha} - \boldsymbol{\sigma}\|_{\Omega_\Pi} \tag{3.2}$$

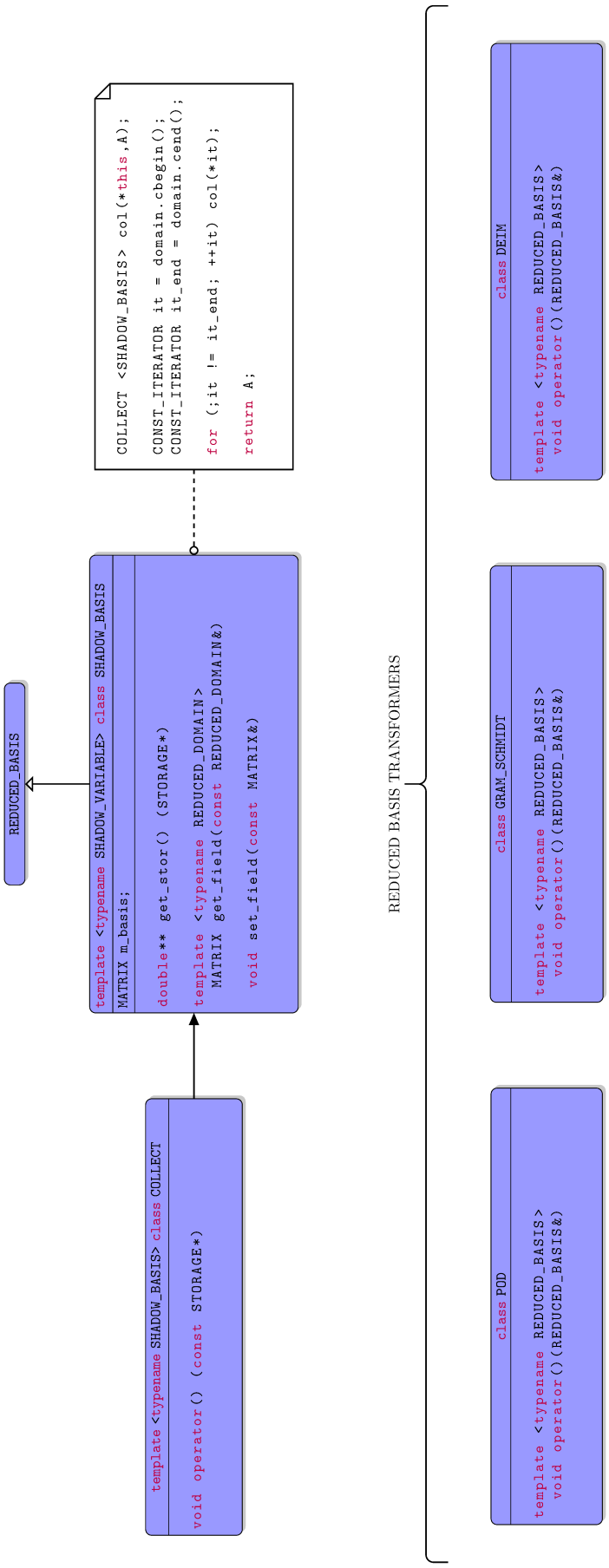


Figure 3.24 – Proposition d'architecture pour l'implémentation des bases réduites.

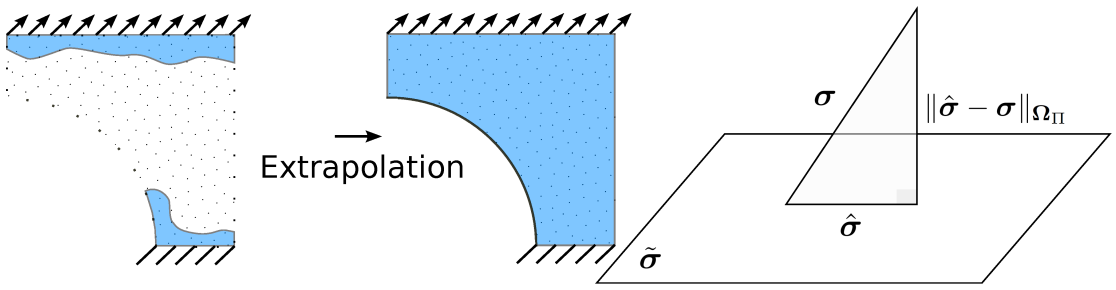


Figure 3.25 – Extrapolation des champs du domaine réduit à l'ensemble de la structure par une méthode des moindes carrés.

Puis on en déduit une extrapolation $\hat{\boldsymbol{\sigma}}$ du champ à l'ensemble du domaine de la structure :

$$\hat{\boldsymbol{\sigma}} = \tilde{\boldsymbol{\sigma}} \cdot \hat{\boldsymbol{\alpha}} \tag{3.3}$$

Dans l'hyperréduction [59], les bases des variables internes sont des bases scalaires :

$$\hat{\boldsymbol{\alpha}}_{ii} = \arg\max_{\boldsymbol{\alpha}_{ii}} \|\tilde{\boldsymbol{\sigma}}_{ii} \cdot \boldsymbol{\alpha}_{ii} - \boldsymbol{\sigma}_{ii}\|_{\Omega_\Pi} \tag{3.4}$$

Ceci n'est pas satisfaisant, notamment pour le champ de contrainte. En effet, lors de la réextrapolation, l'équilibre mécanique de ce dernier n'est pas satisfait lors d'un traitement composante par composante.

La nouveauté que nous proposons consiste à introduire des bases vectorielles ou tensorielles. Ainsi pour un champ de contraintes d'une structure 3D, les snapshots permettant d'enrichir la base prennent la forme suivante :

$$\boldsymbol{\sigma} = \begin{pmatrix} \vdots \\ \boldsymbol{\sigma}_{11}^{(k)} \\ \boldsymbol{\sigma}_{22}^{(k)} \\ \boldsymbol{\sigma}_{33}^{(k)} \\ \boldsymbol{\sigma}_{12}^{(k)} \\ \boldsymbol{\sigma}_{23}^{(k)} \\ \boldsymbol{\sigma}_{31}^{(k)} \\ \vdots \end{pmatrix} \tag{3.5}$$

où k correspond au k point d'intégration du maillage de la structure totale au lieu de :

$$\boldsymbol{\sigma}_{ii} = \begin{pmatrix} \vdots \\ \boldsymbol{\sigma}_{ii}^{(k)} \\ \vdots \end{pmatrix} \tag{3.6}$$

pour la base des ii composantes des contraintes dans le cas de base scalaire par composante pour les contraintes.

La base $\tilde{\boldsymbol{\sigma}}$ est constituée d'un tel arrangement de snapshots :

$$\tilde{\boldsymbol{\sigma}} = \begin{pmatrix} & \vdots & \\ \cdots & \tilde{\boldsymbol{\sigma}}_{11}^{(k)} & \cdots \\ \cdots & \tilde{\boldsymbol{\sigma}}_{22}^{(k)} & \cdots \\ \cdots & \tilde{\boldsymbol{\sigma}}_{33}^{(k)} & \cdots \\ \cdots & \tilde{\boldsymbol{\sigma}}_{12}^{(k)} & \cdots \\ \cdots & \tilde{\boldsymbol{\sigma}}_{23}^{(k)} & \cdots \\ \cdots & \tilde{\boldsymbol{\sigma}}_{31}^{(k)} & \cdots \\ & \vdots & \end{pmatrix} \tag{3.7}$$

Afin d'éviter la réplication de code, la programmation générique est utilisée très largement. La classe `class REDUCED_BASIS` est paramétrée par une `REDUCED_VARIABLE` qui donne le type de la variable mécanique considéré (scalaire, vecteur, tenseur) ce qui permet ainsi une gestion unifiée de base non nécessairement scalaires.

Chaque instance de `class REDUCED_BASIS` contient un noeud réduit de type `class BASIS_REDUCED_NODE <REDUCED_BASIS>`. Les instances de la classe `class BASIS_REDUCED_NODE<REDUCED_BASIS>` permettent de traduire et d'organiser les données de la base réduite afin qu'elles soient lisibles par les éléments réduits. Elles font le lien entre les éléments et la base réduite. Elles contiennent ainsi deux premières méthodes qui sont appelées lors de l'étape *offline* :

- `void set_up_for_reduction()` qui crée, pour chaque élément actif appartenant au domaine réduit, autant de degrés de liberté réduits associés que la base réduite contient de vecteurs. Les degrés de liberté créés sont ajoutés à chaque élément réduit actif présent dans le support de la base.
- `void set_up_for_hyperreduction()` rassemble l'ensemble des degrés de liberté réduits appartenant à la frontière du domaine réduit. Le rassemblement des degrés de liberté réduit est effectué par une méthode `edge`.

3.3.5 Noeuds réduits. Degrés de liberté réduits.

Prise en compte des degrés de liberté réduits Ainsi qu'illustré figure 3.27 les conditions aux limites réduites comme les bases réduites génèrent des noeuds réduits lors de l'étape 1. De la même manière que pour les noeuds éléments finis réels, à qui on associe un certain nombre d'éléments partageant le noeud, aux noeuds réduits sont liés des éléments réduits au cours de l'étape 2. Pour les noeuds réduits issus d'une base réduite, les éléments associés sont ceux se trouvant à l'intersection entre le support de la base et le domaine d'intégration réduit 3.9. Pour les noeuds réduits issus de conditions aux limites portant sur les noeuds réels, il s'agit des éléments réduits correspondant aux éléments réels sous-jacents partageant déjà lesdits noeuds sous-jacents. Dans les deux cas, ces noeuds réduits génèrent à leur tour des étiquettes de degrés de liberté réduit qui, en caractérisant un degré de liberté, vont permettre d'organiser le système réduit. Cela correspond à l'étape 3 de la figure 3.27. Les étiquettes de degrés de liberté réduits sont ensuite enregistrées auprès des éléments réduits appropriés

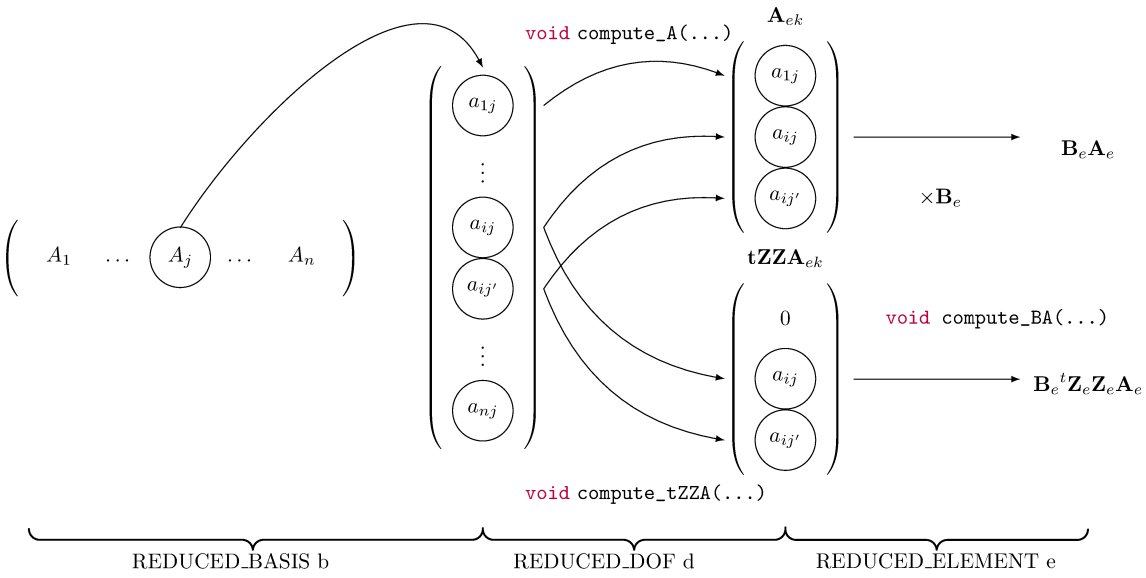


Figure 3.26 – Construction des fonctions de forme hyperréduites

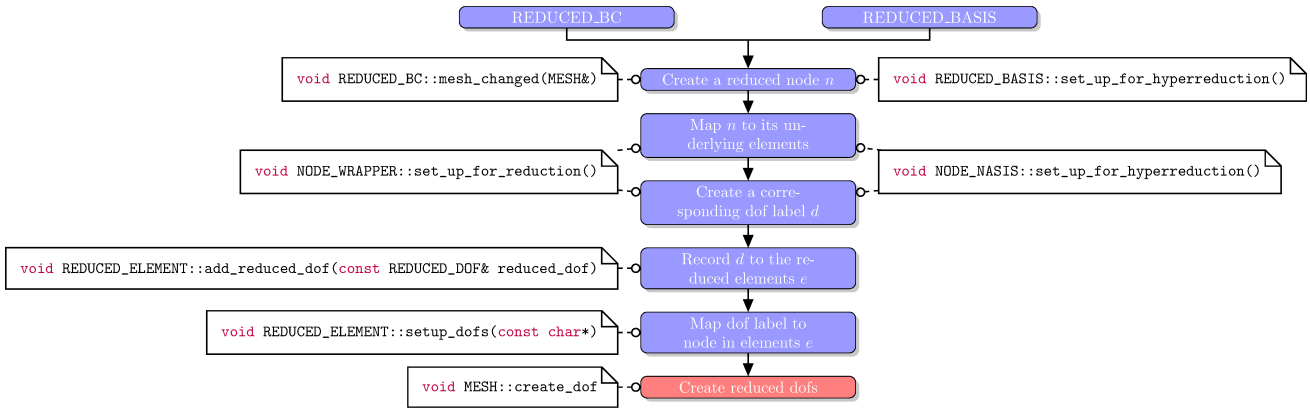


Figure 3.27 – Prise en compte des degrés de liberté réduits

pendant l'étape 4. Ainsi que présenté dans le tableau 3.7, les étiquettes de degré de liberté réduit, comportent une information sur le noeud auquel est relatif le degré de liberté. Ceci permet d'associer au niveau élémentaire, lors de l'avant-dernière étape de la figure 3.27, les noeuds réduits aux étiquettes de données correspondantes avant de créer effectivement les degrés de liberté du maillage réduit en dernière étape.

Caractérisation des degrés de liberté Des étiquettes qualifiant les degrés de liberté sont classiquement utilisés par ZSET pour caractériser les degrés de liberté à créer. Elles sont implémentées dans une structure `class DOF_TYPE` qui porte toutes les informations nécessaires pour identifier complètement un degré de liberté. Ces informations sont utilisées au niveau de chaque élément par le truchement de la méthode `void setup_dofs()` responsable de détailler les caractéristiques de l'ensemble des degrés de liberté au niveau purement élémentaire ainsi que les noeuds auxquels ils sont éventuellement rattachés. Un degré de liberté réduit est simplement une extension de cette notion d'étiquette de degré de liberté. Les étiquettes `class REDUCED_DOF` ajoutent aux fonctionnalités d'identification des étiquettes de `class DOF_TYPE` des fonctionnalités permettant de calculer la projection des espaces réduits sous-jacent sur un élément donné par le biais des méthodes `void compute_A(const REDUCED_ELEMENT&,int,MATRIX &)const` et `void compute_tZZA(const REDUCED_ELEMENT&,int,MATRIX&)const` ainsi que résumé dans le tableau 3.7. Par ailleurs l'étiquette de données conserve trace du noeud réduit auquel appartient le dof réduit. Ce mécanisme permet d'isoler complètement la construction des espaces réduits de leur utilisation par les éléments réduits.

Mise en place des degrés de liberté La figure 3.28 illustre la procédure d'ordonnancement des degrés de liberté aux noeuds lors d'un calcul éléments finis classique. L'idée est de ne compter qu'une fois chaque degré de liberté dans le maillage alors que ces degrés de liberté figurent dans plusieurs éléments. L'ordonnancement des degrés de liberté est réalisé dans le maillage. Pour chaque élément e, on parcourt l'ensemble des degrés de liberté puis, pour chaque degré de liberté d, on remonte au noeud associé n. Du noeud associé n, on obtient une liste des éléments auxquels il appartient. A titre d'exemple sur la figure 3.9, le noeud 1820 est lié aux éléments 1729,1759,1753 et 1726. Trois cas se présentent. L'élément e' en question est l'élément e : on s'assure alors de la non-redondance du degré de liberté d. S'il n'a pas été déjà comptabilisé, on le figure comme candidat potentiel à l'ajout à la liste des degrés de liberté du maillage. L'élément e' peut maintenant posséder un rang inférieur à l'élément actuel e. En ce cas, les degrés de liberté de e' sont susceptibles d'avoir déjà été comptabilisés et il faut s'assurer que d, le degré de liberté courant ne correspond à aucun d'entre eux. Si l'élément e' a un rang plus grand que e, aucune contre-indication à l'ajout n'est nécessaire. Si le degré de liberté satisfait les deux conditions pour l'ensemble des éléments du noeud n, il est ajouté. C'est ainsi qu'est traité le problème de la redondance des degrés de liberté. Cette procédure est une méthode de `class MESH`. C'est une des routines coeur du code éléments finis ZSET.

De la même manière que les éléments se partagent les degrés de liberté par l'intermédiaire de leurs noeuds communs, les éléments réduits et hyperréduits se partagent les degrés de liberté de la base à laquelle ils sont attachées. On définit un couple noeud-degré de liberté pour l'élément réduit afin de pouvoir utiliser la procédure 3.28. Cependant, afin de satisfaire aux

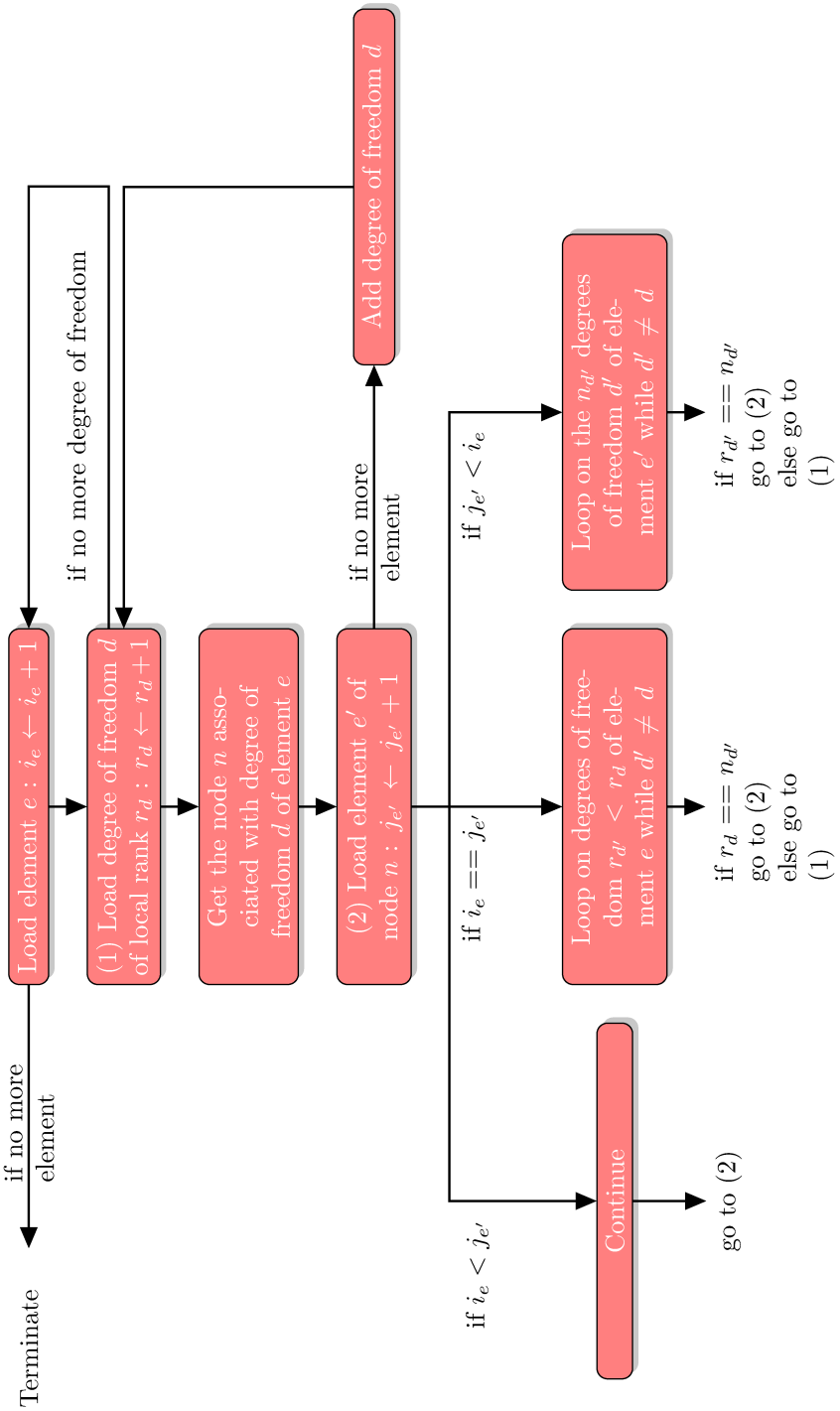


Figure 3.28 – Procédure de création des degrés de liberté.

Fonctionnalité	Prototype de la méthode
Identifier un degré de liberté réduit	`const STRING& get_name()const` et `const DOF_KIND get_kind() const`
Renvoyer le noeud auquel appartient le degré de liberté réduit	`const NODE* get_node()const`
Projeter la contribution de l'espace réduit sur l'élément réduit	`void compute_A(const REDUCED_ELEMENT&,int,MATRIX&) const`
Projeter la contribution de l'espace réduit test sur l'élément réduit	`void compute_tZZA(const REDUCED_ELEMENT&,int,MATRIX&) const`

Table 3.7 – Fonctionnalités du degré de liberté réduit

conditions aux limites réelles en bord de maillage réduit, les noeuds réels sur lesquels elles portent sont introduits dans le maillage réduit, un pour chaque degré de liberté associé.

```
struct REDUCED_DOF
{
    virtual const STRING&
    get_name() const=0;

    virtual const DOF_KIND
    get_kind() const=0;

    virtual const NODE*
    get_node() const=0;

    virtual void
    compute_A(const REDUCED_ELEMENT&,int,MATRIX&) const=0;

    virtual void
    compute_tZZA(const REDUCED_ELEMENT&,int,MATRIX&) const=0;
};
```

Listing 3.3 – Interface pour l'étiquette de degré de liberté

Cette interface définit une volée de méthodes que l'on peut ranger en trois catégories :

- le constructeur et le destructeur. On notera que dans cette architecture, les degrés de liberté réduits sont toujours associés à une noeud réduit, comme l'exige ce contructeur. Il existe dans le code ZSET d'autres types de degré de liberté qui ne sont pas liés à un noeud particulier. Par exemple les degrés de liberté reliés directement à un élément ou les degrés de liberté reliés à une famille d'éléments.
- des méthodes permettant de caractériser et d'identifier le degré de liberté réduit.
- des méthodes permettant de calculer la contribution du degré de liberté réduit dans les fonctions de forme élémentaires. C'est le cas des méthodes `void compute_A( const REDUCED_ELEMENT& reduced_elem, int j, MATRIX& A)const` et `void compute_tZZA(const REDUCED_ELEMENT& reduced_elem, int j, MATRIX& tZZA)const`. Ces méthodes permettent de calculer la projection d'une partie du vecteur de base et de la stocker dans la `j` de

la matrice `A` ou `tZZA`.

Pour effectuer l'assemblage des fonctions de forme réduites élémentaires, l'élément réduit invoque chaque degré de liberté réduit. L'élément réduit n'a aucune connaissance directe des noeuds réduits.

Spécialisations Ainsi qu'illustré figure 3.6, il existe deux types d'étiquettes de degré de liberté réduit : les degrés de liberté qui correspondent à `class BASIS_DOF` associés à une base réduite et les degrés de liberté associés à une condition au limite, ceux de type `class DOF_WRAPPER`. L'interface `class REDUCED_DOF` permet à l'élément réduit `class REDUCED_ELEMENT` de les manipuler indistinctement.

Traitement des conditions aux limites Les conditions aux limites permettent de solliciter de manière adéquate le modèle réduit. Dans ce plugin d'hyperréduction, elles sont prises en compte en introduisant des noeuds et des degrés de liberté additionnels dans le système réduit. L'interface pour les conditions aux limites est l'interface `class REDUCED_BC`. Les méthodes `void set_up_forreduction`, `void set_up_for_hyperreduction` permettent de créer, à partir du modèle complet, les conditions aux limites réduites correspondantes et de les insérer dans le modèle réduit. La méthode `void _update` permet de mettre à jour les conditions aux limites réduites dans le cas d'un calcul hyperréduit. La figure 3.5 montre l'ensemble des classes spécialisées dérivant de l'interface `class REDUCED_BC`. Ces spécialisations permettent de préciser l'impact sur le système réduit des conditions aux limites imposées sur la structure réelles, que cela soit des conditions aux limites imposées aux noeuds primales ou duales 3.8, des conditions aux limites de bord libre qui nécessitent un traitement particulier, des conditions aux limites portant sur les degrés de liberté partagés par un ensemble d'éléments. Afin de répercuter ces degrés de liberté additionnels, les spécialisations de `class REDUCED_BC` créent des noeuds réduits additionnels qui portent un degré de liberté réduit et les enregistrent auprès des éléments. Elles entretiennent ainsi une liste de noeuds comme l'illustre la figure 3.5. Les noeuds sont mis à jour classiquement au gré des incréments en utilisant la méthode `void _update` de `class BC` et ce de manière tout à fait transparente pour les autres composants du plugin. En particulier, pour l'élément réduit, les noeuds et les degrés de liberté provenant des conditions aux limites ou d'une base réduite sont impossibles à distinguer, ce qui permet un traitement unifié des degrés de liberté.

Construction des fonctions de forme hyperréduites Lorsque les espaces réduits de base et de conditions aux limites ont été mis à jour, que les degrés de liberté réduits ont été enregistrés auprès des éléments, il reste à calculer les fonctions de forme élémentaires ${}^t\boldsymbol{A}_e{}^t\boldsymbol{Z}_e\boldsymbol{Z}_e{}^t\boldsymbol{B}_e$ et $\boldsymbol{B}_e\boldsymbol{A}_e$ décrites dans le tableau 3.2. Comme illustré figure 3.26 l'assemblage s'effectue en deux temps : pour chaque colonne de base des déplacements qui correspond à un degré de liberté réduit et pour chaque élément hyperréduit possédant ce degré de liberté, on prélève les contributions locales des déplacements réduits. Les méthodes mises en jeu pour effectuer ces projections sont `void compute_A(...)` et `void compute_tZZA(...)` de l'interface `REDUCED_DOF`. Une fois les matrices de contribution élémentaires $\boldsymbol{A}_e$ et ${}^t\boldsymbol{Z}_e\boldsymbol{Z}_e\boldsymbol{A}_e$ constituées, les fonctions de forme réduites s'en déduisent aisément par une multiplication

Fonctionnalité	Interface
Gérer les conditions aux limites	`class BC`
Gérer les conditions aux limites réduites	`class REDUCED_BC`
Gérer les conditions aux limites réduites primales et duales imposées aux noeuds	`class REDUCED_BC_IMPOSE_NODAL`
Gérer les conditions aux limites réduites de bord libre	`class BC_NEUMANN`
Gérer les conditions aux limites portant sur un groupe d'élément	`class REDUCED_BC_IMPOSE_ELSET`

Table 3.8 – Spécification pour les conditions aux limites

par les fonctions de forme réelles $\boldsymbol{B}_e$ en chaque point de Gauss. Cette opération est effectuée dans `REDUCED_ELEMENT::compute_BA(...)`.

Toutes les méthodes préablement citées concernent l'étape de calcul *online*. L'élément fini réduit possède aussi des méthodes *offline* qui sont effectuées une fois pour toutes au cours du calcul ou à chaque adaptation. En effet, le gain de vitesse si prisé par les défenseurs de l'hyperréduction résulte d'une factorisation des calculs ce qui permet de ne pas effectuer sans cesse des calculs qui peuvent être réalisés seulement une fois de temps en temps. Ces méthodes sont :

- `void before_mesh_set_up_for_reduction()`
- `void after_mesh_set_up_for_reduction()`
- `void set_up_for_reduction()`
- `void set_up_for_adaptation()`

3.4 Conclusion

Dans ce chapitre, une architecture pour l'hyperréduction de modèle intégrée à un code modulaire éléments finis a été présentée.

Elle repose sur la notion centrale d'élément réduit et suit une logique de production et de consommation de ces éléments par différentes instances au cours de phases d'adaptation ou de calcul hyperréduit.

Un modèle réduit et son adaptation est créé par le programmeur à partir de briques élémentaires (définition de domaines réduits, de bases réduites, de méthodes de synthèse pour le modèle).

La grande similarité entre l'architecture des codes éléments finis orientés objet montre que l'architecture de ce plugin n'est pas spécifique à la logique ZSET et peut être développée dans d'autres codes.

Notre implémentation est documentée et conforme à un certain nombre de canons de programmation comme nous nous sommes attachés à le justifier dans le présent chapitre ce qui rend possible l'appréhension et la prise en main par d'autres chercheurs.

Les conséquences d'une telle logique d'implémentation sont nombreuses : les performances du calcul réduit sont intimement liés aux performances du code de calcul sous-jacent ce qui peut présenter un inconvénient si le code n'est pas suffisamment performant sur des problèmes de petite taille cependant que l'amélioration de la performance du code FEM entraînera celle du code d'hyperréduction.

Néanmoins, un travail d'amélioration du plugin reste à mener, notamment en ce qui concerne l'étude systématique des goulots d'étranglement des performances, la mise en oeuvre de méthodes plus efficaces de construction de modèles réduits. Il y a aussi la nécessité de développer plus de cas tests afin de rendre nos développements plus robustes.

Ayant maintenant acquis la capacité de construire un modèle réduit et de l'utiliser, nous allons présenter un certain nombre de cas d'application lors des chapitres suivants.

4

Conditions aux limites en hyperréduction

Avant de coupler l'hyperréduction avec les méthodes EF^2, ce qui revient à pratiquer l'hyperréduction sur un grand nombre de cellules périodiques simultanément, il convient de s'assurer que l'on sait hyperréduire sur une seule cellule périodique. Nous verrons que la prise en compte des conditions aux limites périodiques, de même que celle des bords libres, nécessite un soin particulier.

Sommaire

4.1 Problématique

Conditions de bord libre Dans [59] ou [40] est exposée une méthode d'hyperréduction de modèle appliquée à une plaque perforée en non-linéaire. Seul le traitement des conditions de type Dirichlet est présenté. Il consiste à chercher la solution du champ de déplacement dans l'espace affine $\boldsymbol{u}_0 + \boldsymbol{A}\delta\boldsymbol{u}$ où $\boldsymbol{u}_0$ est le vecteur incrément de déplacement sur les noeuds de Dirichlet et $\delta\boldsymbol{u}$ la composante des déplacements réduits. Or le domaine d'intégration réduit choisi peut contenir des éléments en bord de structure. La prise en compte des conditions de bord libre imposées dans l'équilibre éléments finis classique n'est pas répercutée sur les bords libres du domaine d'intégration réduit au moment du calcul de l'équilibre réduit. Sur le bord libre, le déplacement s'opère dans l'espace vectoriel réduit. Aucune condition de bord n'est imposée. Il en va de même dans les versions développées ultérieurement de l'hyperréduction, par exemple dans [60]. D'où la question : est-il possible d'écrire un équilibre hyperréduit prenant en compte les conditions de bord libre telle que l'assemblage et la résolution ne soit pas plus coûteux que l'assemblage et la résolution de [59] ou [60] ?

Conditions périodiques Dans [23] l'hyperréduction a été étendue au cas de structures hétérogènes ce qui signifie que le modèle réduit se compose de bases de variables internes par phase 4.1. La base des déplacements reste, elle, globale. La méthode d'hyperréduction hétérogène a été appliquée à des VER sollicités en conditions périodiques. Classiquement, on décompose le champ de déplacements en $\boldsymbol{u} = \boldsymbol{E} \cdot \boldsymbol{x} + \boldsymbol{v}$ où $\boldsymbol{E}$ représente les déformations macroscopiques, $\boldsymbol{x}$ représente la position du point courant à l'intérieur du VER et $\boldsymbol{v}$ est le terme de fluctuation de déplacements microscopiques. C'est à partir de ce dernier terme qu'on forme les bases de déplacements qui sont des bases de fluctuations microscopiques. Ainsi la périodicité du champ de fluctuation microscopique est-elle automatiquement assurée lors de la résolution de l'équilibre hyperréduit. En revanche, la condition d'anti-périodicité des $\boldsymbol{\sigma} \cdot \boldsymbol{n}$ n'est pas vérifiée dans l'équilibre hyperréduit. L'indicateur d'erreur permet de s'assurer de son établissement a posteriori. Mais des difficultés de convergence du calcul hyperréduit surgissent (fig. 4.3) alors ce qui conduit les auteurs à ignorer les éléments frontières du VER (fig. 4.2). Là encore, même question : est-il possible d'écrire un équilibre hyperréduit prenant en compte à faible coût les conditions d'anti-périodicité des contraintes ? De surcroît, si l'on construit sans plus de contraintes un domaine réduit, on se heurte à un problème de convergence des calculs d'hyperréduction, y compris en élasticité linéaire. Dans [23], les auteurs ont écarté le problème en excluant les éléments frontières. Nous proposons ici de les intégrer en opérant sur un domaine d'intégration réduit périodique.

Le présent chapitre est consacré à la résolution de ces trois problèmes, i. e. la prise en compte des conditions de bord libre, la prise en compte de l'anti-périodicité des contraintes dans l'équilibre hyperréduit et celui de la construction du domaine réduit périodique.

Dans la suite du chapitre, nous nous réfèrerons au schéma de principe 4.4.

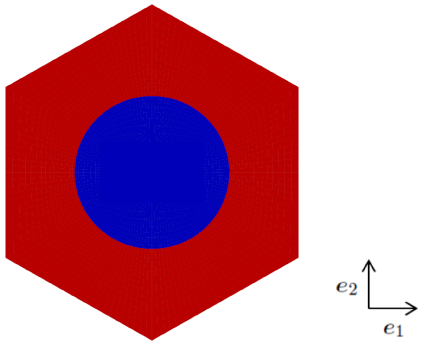


Figure 4.1 – Cellule périodique hexagonale sur laquelle est pratiquée l'hyperréduction de modèle dans [23]

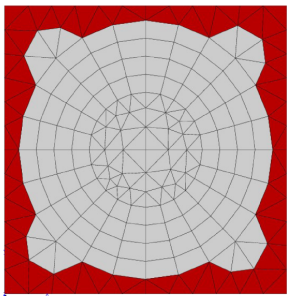


Figure 4.2 – Dans [23], les calculs effectués en condition périodique excluent systématiquement les éléments du bord de la cellule. Malgré cela, les résultats obtenus par l'hyperréduction de modèle sont en bon accord avec les résultats obtenus par un calcul éléments finis classique.

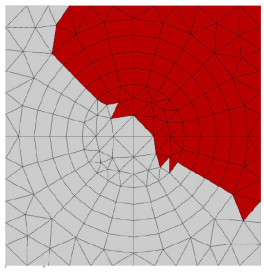


Figure 4.3 – Si on reprend certains calculs de [23] et que l'on omet d'imposer la condition d'exclusion des éléments du bord, on obtient des domaines d'intégration réduit de ce type sur lesquels il est impossible d'aboutir à la convergence des calculs, même dans le cas élastique linéaire.

4.2 Les conditions de bord libre

4.2.1 Théorie et mise en oeuvre

Principe Lors d'un calcul éléments finis classique, ne rien spécifier en terme de conditions aux limites revient à imposer des conditions aux limites de bord libre. Lorsque l'on parle d'hyperréduction, la projection sur l'espace réduit escamote ces conditions et ne rien spécifier revient simplement à imposer que les déplacements aux noeuds libres frontière soient inclus

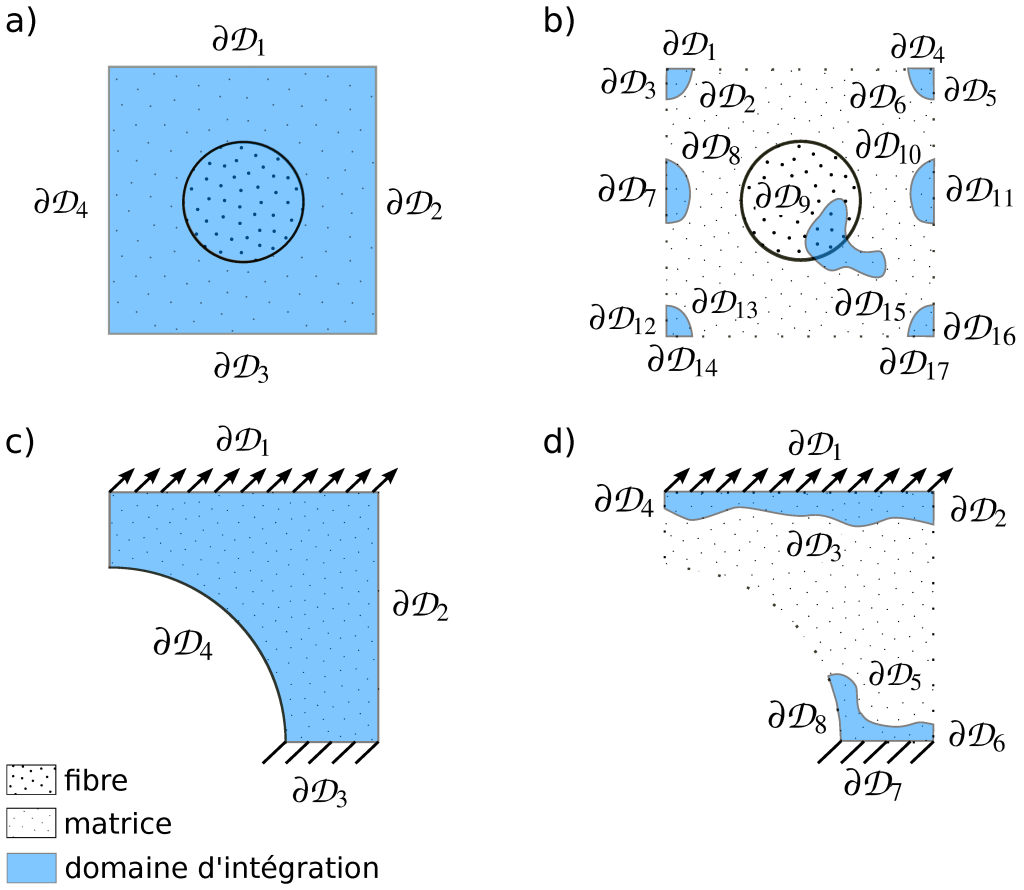


Figure 4.4 – Principe du calcul d'hyperréduction. a) et c) calcul complet de référence, le domaine d'intégration est étendu à toute la structure. b) et d) calcul d'hyperréduction, le domaine d'intégration est limité à un domaine réduit. Un calcul d'hyperréduction engendre une multiplication des bords $\partial\mathcal{D}_i$ sur lesquels il faut imposer des conditions aux limites.

dans un espace réduit. L'agrégation de la réaction sur l'espace réduit conduit à la perte de la condition de bord libre. Du point de vue du temps de calcul, il est peu coûteux de rétablir cette condition en ajoutant des degrés de liberté réduits sur les éléments réduits dont le sous-jacent voit les conditions de bord libre. C'est cette approche que nous avons adoptée.

Rappels sur les éléments hyperréduits La matrice de rigidité élémentaire sur un élément du domaine d'intégration réduit s'écrit sous la forme :

$$\int_{\Omega_e} {}^t\boldsymbol{A}_e\, {}^t\boldsymbol{Z}_e \boldsymbol{Z}_e\, {}^t\boldsymbol{B}_e \frac{\partial\sigma}{\partial\varepsilon} \boldsymbol{B}_e \boldsymbol{A}_e \tag{4.1}$$

et le vecteur de réaction interne prend la forme :

$$\int_{\Omega_e} {}^t\boldsymbol{A}_e\, {}^t\boldsymbol{Z}_e \boldsymbol{Z}_e\, {}^t\boldsymbol{B}_e \boldsymbol{\sigma} \tag{4.2}$$

où $\boldsymbol{A}_e$ est la projection de la base des déplacements sur l'élément considéré, $\boldsymbol{Z}_e$ la matrice de sélection, $\boldsymbol{B}_e$ la matrice de fonction de forme élémentaire.

Mise en oeuvre La prise en compte des conditions de bord libre sur le j degré de liberté de l'élément e se fait en

- substituant à la matrice $\boldsymbol{A}_e$ la matrice $\begin{pmatrix}\boldsymbol{A}_e & \boldsymbol{v}_e\end{pmatrix}$: $\boldsymbol{A}_e \leftarrow \begin{pmatrix}\boldsymbol{A}_e & \boldsymbol{v}_e\end{pmatrix}$, avec $\boldsymbol{v}_e = (\delta_{ij})_{0\leq i<n}$
- créant un degré de liberté réduit approprié, i. e. substituant $\delta\boldsymbol{q}_e \leftarrow \begin{pmatrix}\delta\boldsymbol{q}_e \\ \delta q_e\end{pmatrix}$

L'objet éléments finis entrant en jeu dans ces deux opérations est l'objet `class REDUCED_DOF`. Plus précisément :

- la première fonctionalité est réalisée par les méthodes de `class REDUCED_DOF void compute_A(const REDUCED_ELEMENT& e,int jloc,MATRIX& A)const` et `void compute_A(const REDUCED_ELEMENT& e,int jloc,MATRIX& tZZA)const` qui permettent de construire les matrices de base des déplacements pour chaque degré de liberté ;
- la deuxième fonctionalité est mise en oeuvre en invoquant la méthode `void add_reduced_dof (AUTO_PTR<REDUCED_DOF> reduced_dof)` de `class REDUCED_ELEMENT` qui se résume à signaler l'existence du degré de liberté réduit en question dans l'élément courant. La méthode `void setup_dofs(const char*)` est classiquement appelée par le code afin de créer le maillage.

Architecture logicielle La création des instances de `class REDUCED_DOF` s'effectue au niveau d'un objet susceptible de gérer les conditions aux limites réduite de type Neumann, `class BC_NEUMANN`. Très logiquement, `class BC_NEUMANN` s'inscrit dans la hiérarchie suivante :

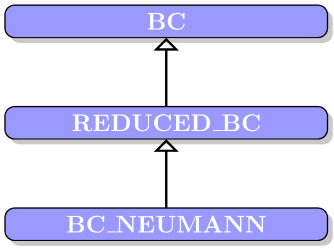


Figure 4.5 – Diagramme de `class BC_NEUMANN`

Une instance de `class BC_NEUMANN` représente bien en effet une condition aux limites que l'on cherche à imposer sur la structure réduite. C'est pourquoi elle dérive de `class REDUCED_BC`. `class REDUCED_BC` dérive de `class BC` qui est la classe de ZSET servant à implémenter n'importe quelle type de conditions aux limites. La méthode `void set_up_for_hyperreduction()` permet de relever les degrés de liberté de bord libre :

```
void                                                                    1
BC_NEUMANN::set_up_for_hyperreduction()                                 2
{                                                                       3
  for (int inode=0; inode != rid.nb_edge_node(); inode++)               4
  {                                                                     5
     const NODE* node = rid.get_edge_node(inode);                       6
     for (int idof=0; idof != node->nb_dof(); idof++)                   7
     {                                                                  8
        const DOF* dof = node->get_dof(idof);                           9
                                                                        10
```

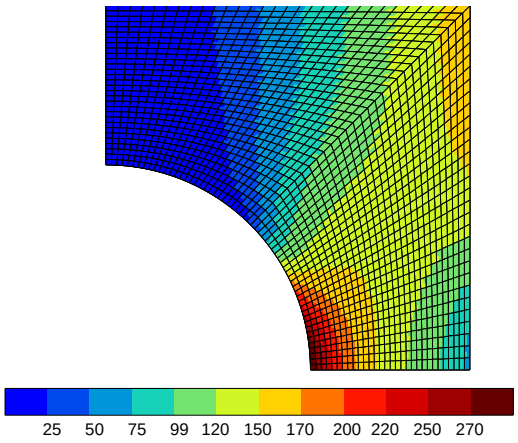


Figure 4.6 – Plaque de référence ($E = 120GPa$, $\nu = 0.3$) en traction simple. $\boldsymbol{\sigma}^{\text{mises}}$ (en MPa).

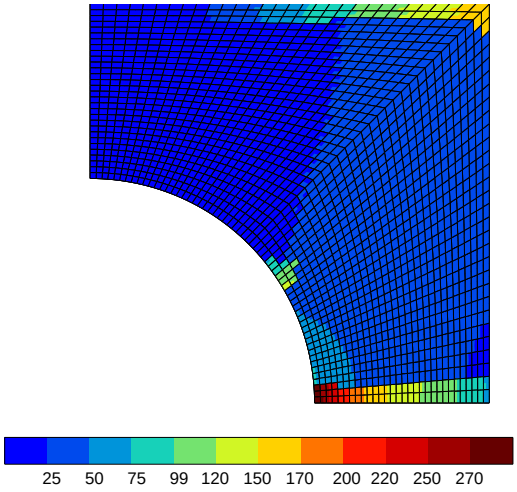


Figure 4.7 – Plaque hyperréduite ($E = 120GPa$, $\nu = 0.3$) en traction simple. $\boldsymbol{\sigma}^{\text{mises}}$ (en MPa).

```
        if(!rid.has_wrapper_dof(dof) &&                    11
           !rid.has_internal_dof(dof))                     12
           add(dof);                                       13
    }                                                      14
  }                                                        15
}                                                          16
```

Aucune autre opération n'est requise, les forces externes étant automatiquement initialisées à 0 en début d'incrément.

4.2.2 Validation sur un cas linéaire

Afin de s'assurer du fonctionnement correcte de la mise en oeuvre des conditions aux limites de Neumann, des cas tests sur un faible nombre d'éléments ont été réalisés (1, 4, 16 éléments). Ces cas tests ne sont pas présentés ici. Ils sont fournis avec le plugin d'hyperréduction. Nous y renvoyons le lecteur intéressé.

Sur les figures 4.6 et 4.7 est présenté la réalisation du calcul de la figure 4.4 cas c) et d). Une plaque trouée est sollicitée en élasticité linéaire. Cela suffit pour preuve de concept en ce qui concerne la prise en compte des conditions aux limites. On observe que les champs de contraintes dans le cas hyperréduit et le cas de référence sont égaux.

4.2.3 Validation sur un cas non-linéaire

Le matériau a un comportement régi par un critère de von Mises de seuil $\boldsymbol{R}_0 = 2,2 \times 10^2 MPa$ et un écrouissage isotrope linéaire $H = 1,5 \times 10^5 MPa$. Il a un module d'Young de $\boldsymbol{E} =$

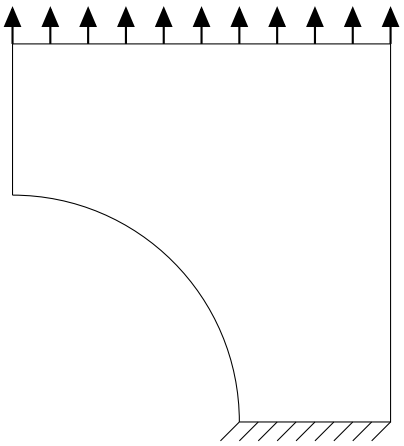


Figure 4.8 – Sollicitation d'une plaque trouée en traction

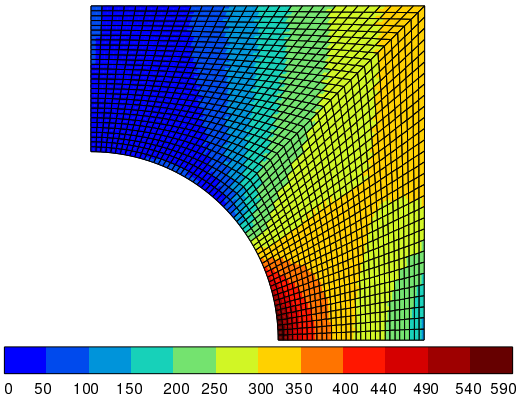


Figure 4.9 – Plaque de référence. $\boldsymbol{\sigma}^{\text{mises}}$ (en MPa).

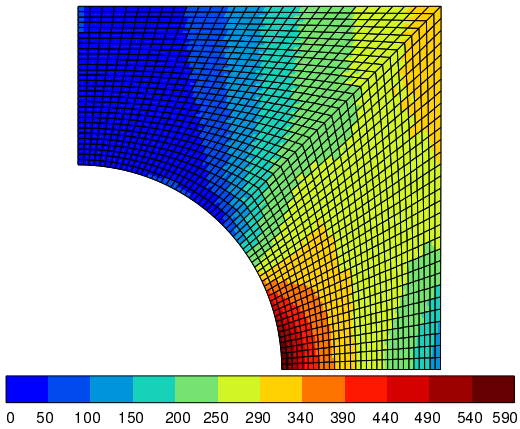


Figure 4.10 – Plaque hyperréduite. $\boldsymbol{\sigma}^{\text{mises}}$ (en MPa).

$1,2 \times 10^5 MPa$ et un coefficient de Poisson de $\nu = 0.3$. Afin de construire la base réduite, nous avons mis en oeuvre une procédure de type Gram-Schmidt recommandée dans [33]. Cet article invite à l'utilisation d'un tel algorithme par rapport à l'emploi d'une décomposition de Karhunen-Loève notamment pour la raison qu'il n'accorde pas un poids trop conséquent aux valeurs importantes des *snapshots* correspondant à des états ultérieurs par rapport à l'état de la structure initiale. Le domaine d'intégration réduit a été construit en suivant une méthode de type DEIM présentée dans [19] et appliquée aux bases construite par la méthode de Gram-Schmidt. Le facteur d'élargissement du domaine réduit est fixé à 1. Le ratio d'adaptation est de l'ordre de 0.01.

Sur la figure 4.9 est présenté le cas d'une plaque en traction. La vitesse de déformation imposée selon $U2$ est de 0.025s^{-1} pendant $2,5$s sur la partie supérieure de la plaque puis de 0.015s^{-1} pendant 10s ainsi que stipulé dans le fichier de mise en données (tab. 4.2).

Les figures 4.9 et 4.10 présentent les isovaleurs du champ de contraintes sigmises respectivement pour le calcul de référence et pour le calcul hyperréduit. On constate que dans la zone d'intérêt, i. e. au niveau du domaine hyperréduit dans le coin intérieur gauche de la plaque les champs de contraintes sigmises correspondent. Ainsi la procédure d'hyperréduction

adaptative est capable de décrire le comportement de la structure soumise à ce type de sollicitation. En dehors des zones d'intérêt, les champs ne correspondent pas. Il est possible de pratiquer une réextrapolation des champs du domaine réduit à l'ensemble de la structure pour disposer des champs complets à l'issue d'une procédure d'hyperréduction.

Le déroulement du calcul d'hyperréduction pour la plaque perforée est décrit figure 4.1. On observe qu'alternent les étapes de calcul complet où le problème éléments finis possède environ 11000 degrés de liberté avec les étapes de calcul réduit où le nombre de degrés de liberté est de l'ordre de la centaine. La pertinence du modèle réduit se mesure au rapport entre le nombre d'adaptation requises sur le nombre total d'incréments effectués.

Ce calcul correspond au fichier de mise en données de la table 4.2

Algorithme 3 : Intégration d'un snapshot s au modèle réduit par un algorithme de Gram-Schmidt

input : $\mathcal{B} = (b_1, ..., b_n)$;
output : $\mathcal{B}$;
Get a snapshot s ;
if $\|s\| < \varepsilon$ **then**
  stop ;
if $\dim \mathcal{B} = 0$ **then**
  $\mathcal{B} \leftarrow \left(\frac{s}{\|s\|}\right)$;
  stop ;
Set $r \leftarrow \|s\|$;
for $b \in \mathcal{B}$ **do**
  $r \leftarrow r - (s \cdot b)b$;
if $r > \varepsilon$ **then**
  for $b \in \mathcal{B}$ **do**
    $s \leftarrow s - (s \cdot b)b$;
  $\mathcal{B} \leftarrow \left(\mathcal{B}, \frac{s}{\|s\|}\right)$;

4.3 Les conditions périodiques

4.3.1 Théorie et mise en oeuvre

Le travail sur les conditions périodiques est un peu plus délicat.

Le problème éléments finis consiste à trouver un champ de déplacement de la forme $\boldsymbol{u} = \boldsymbol{E} \cdot \boldsymbol{x} + \boldsymbol{v}$ et un champ de contrainte $\boldsymbol{\sigma}$ tels que :

- $\boldsymbol{v}$ est périodique au bord du domaine, i. e. $\boldsymbol{v}^+ = \boldsymbol{v}^-$;
- $\boldsymbol{\sigma} \cdot \boldsymbol{n}$ est anti-périodique au bord du domaine, i. e. $\boldsymbol{\sigma}^+ \cdot \boldsymbol{n}^+ = -\boldsymbol{\sigma}^- \cdot \boldsymbol{n}^-$;

```
key: hyperreduction
key: mesh
key: linear_solver
key: resolution
key: bc
key: material
key: table

Done loading plaque5.geof
Number of global DOFs: 10982

Number of elements              : 1800
Total number of dof             : 10982
Estimated memory needed (MB)    : 7

seq:  1 incr:  1 dt:0.8333   t:0.8333
 iter:  1 | R |/| Fext |= 1.624e-04  | Fext |=608.605
 iter:  2 | R |/| Fext |= 2.978e-08  | Fext |=608.624

Number of global DOFs: 147

Number of elements              : 1800
Total number of dof             : 147
Estimated memory needed (KB)    : 37

 iter:  1 | R |/| Fext |= 1.627e-04  | Fext |=608.617
 iter:  2 | R |/| Fext |= 5.336e-09  | Fext |=608.624

seq:  1 incr:  2 dt:0.25     t:1.083
 iter:  1 | R |/| Fext |= 1.027e-02  | Fext |=787.492
 iter:  2 | R |/| Fext |= 2.103e-05  | Fext |=787.114

Number of elements              : 1800
Total number of dof             : 10982
Estimated memory needed (MB)    : 7

seq:  1 incr:  2 dt:0.25     t:1.083
 iter:  1 | R |/| Fext |= 2.135e-02  | Fext |=787.489
 iter:  2 | R |/| Fext |= 9.244e-04  | Fext |=788.987
 iter:  3 | R |/| Fext |= 1.048e-07  | Fext |=789.035

Number of global DOFs: 316

Number of elements              : 1800
Total number of dof             : 316
Estimated memory needed (KB)    : 119

seq:  1 incr:  2 dt:0.25     t:1.083
 iter:  1 | R |/| Fext |= 1.441e-02  | Fext |=787.490
 iter:  2 | R |/| Fext |= 4.147e-04  | Fext |=789.031
 iter:  3 | R |/| Fext |= 2.449e-08  | Fext |=789.035

seq:  1 incr:  3 dt:0.25     t:1.333
 iter:  1 | R |/| Fext |= 1.301e-02  | Fext |=962.286
 iter:  2 | R |/| Fext |= 6.870e-05  | Fext |=963.813
```

Table 4.1 – Déroulement du calcul d'hyperréduction sur plaque trouée.

```
****calcul
 ***mesh  plane_stress
  **file  plaque.geof

 ***hyperreduction
  **reduced_model
   *magnitude U 0.001
   *magnitude eto 0.0001
   *magnitude sig 1.
   *magnitude eel 0.0001
   *magnitude epcum 0.0001
   *magnitude epi 0.0001
   *widen 1.
   *ratio 0.01

 ***resolution
  **sequence
   *time 5.
   *increment 6
   *ratio 0.01
   *iteration  10
   *algorithm p1p2p3
  **automatic_time automatic_by_sequence global 4
   *divergence 2. 10
   *security 1.5
   *max_dtime 0.25

 ***bc
  **impose_nodal_dof
  haut0  U2  0.01 tab1
  bas1 U2 0.
  bas2 U1 0.

 ***material
   *file materiau.mat
   *integration theta_method_a 1.0 1.e-4 1500
 ***table
  **name tab1
   *time  0.    2.5  10.
   *value 0.    10.  15.
****return
```

Table 4.2 – Mise en données du calcul d'hyperréduction non-linéaire sur plaque trouée

- l'équilibre mécanique est vérifié, i. e. $\operatorname{div} \boldsymbol{\sigma} = 0$;
- la loi de comportement est vérifiée.

Rappels sur la mise en oeuvre des conditions aux limites de type périodique Nous nous intéressons au premier terme dans l'équation d'équilibre qui s'écrit classiquement :

$$\int_{\Omega_e} \boldsymbol{\varepsilon}^* : \boldsymbol{\sigma} = \int_{\Omega_e} \boldsymbol{u}^* \boldsymbol{\sigma} \cdot \boldsymbol{n} + \int_{\Omega_e} \boldsymbol{u}^* \boldsymbol{f} \tag{4.3}$$

En règle générale, chaque terme de l'équation d'équilibre correspond à un type d'élément fini. Ainsi le premier terme de l'équation d'équilibre 4.3 correspond à ce que l'on appelle un élément périodique.

Mais puisque $\boldsymbol{u} = \boldsymbol{E} \cdot \boldsymbol{x} + \boldsymbol{v}$, en déduisant les déformations $\boldsymbol{\varepsilon} = \boldsymbol{E} + \boldsymbol{\varepsilon}'$, on obtient :

$$\int_{\Omega_e} \boldsymbol{E}^* : \boldsymbol{\sigma}(\boldsymbol{E}, \boldsymbol{\varepsilon}') + \int_{\Omega_e} \boldsymbol{\varepsilon}^{*\prime} : \boldsymbol{\sigma}(\boldsymbol{E}, \boldsymbol{\varepsilon}') \tag{4.4}$$

En petites déformations, les réactions internes locales discrétisées et la matrice de rigidité locale discrétisée s'écrivent pour la forme standard non discrétisée :

$$\int_{\Omega_e} \boldsymbol{\varepsilon}^* : \boldsymbol{\sigma}(\boldsymbol{\varepsilon}) \tag{4.5}$$

respectivement $\int_{\Omega_e} {}^t\boldsymbol{B}_e \boldsymbol{\sigma}$ et $\int_{\Omega_e} {}^t\boldsymbol{B}_e \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\varepsilon}} \boldsymbol{B}_e$

La prise en compte des degrés de liberté macroscopiques, i. e. l'obtention des formes discrétisées relatives à forme 4.4 à partir des formes discrétisées relatices à la forme 4.5 se fait en :

- substituant à la matrice $\boldsymbol{B}_e$ la matrice $\begin{pmatrix} \boldsymbol{B}_e & \boldsymbol{I}_e \end{pmatrix}$: $\boldsymbol{B}_e \leftarrow \begin{pmatrix} \boldsymbol{B}_e & \boldsymbol{I}_e \end{pmatrix}$ où $\boldsymbol{I}_e$ est la matrice identité ;
- créant les degrés de liberté macroscopiques associés $\delta\boldsymbol{E}$: $\delta\boldsymbol{v}_e \leftarrow \begin{pmatrix} \delta\boldsymbol{v}_e \\ \delta\boldsymbol{E} \end{pmatrix}$

Comme dans la section 4.2.1, c'est la méthode `void setup_dofs(const char*)` du `class D_ELEMENT` qui est chargée d'effectuer ces opérations.

Hyperréduction de l'élément périodique Elle se fait en deux étapes :

- La prise en compte des relations de périodicité du champ de fluctuation et d'anti-périodicité de $\boldsymbol{\sigma} \cdot \boldsymbol{n}$ en bord de cellule ;
- La projection des fonctions de forme $\boldsymbol{B}_e$ sur l'espace hyperréduit approprié ;

Rappel : prise en compte des relations entre degrés de liberté On décompose des déplacements en une somme directe : $\delta\boldsymbol{u} = \delta\boldsymbol{u}^0 + \delta\boldsymbol{u}^+ + \delta\boldsymbol{u}^-$ de sorte que $\delta\boldsymbol{u}^0$ s'annule à la frontière du domaine $\partial\Omega$. Cette décomposition peut alors être répercutée sur la forme linéaire de réaction :

$$(\mathcal{R}, \delta\boldsymbol{u}) = (\mathcal{R}^0, \delta\boldsymbol{u}^0) + (\mathcal{R}^+, \delta\boldsymbol{u}^+) + (\mathcal{R}^-, \delta\boldsymbol{u}^-) \tag{4.6}$$

$$= \underbrace{\int_{\partial\Omega} \delta\boldsymbol{u}^0 \boldsymbol{\sigma}.\boldsymbol{n}}_{=0} - \int_\Omega \delta\boldsymbol{\varepsilon}^0 : \boldsymbol{\sigma} + \int_{\partial\Omega^+} \delta\boldsymbol{u}^+ \boldsymbol{\sigma}.\boldsymbol{n} - \int_{\Omega^+} \delta\boldsymbol{\varepsilon}^+ : \boldsymbol{\sigma} + \int_{\partial\Omega^-} \delta\boldsymbol{u}^- \boldsymbol{\sigma}.\boldsymbol{n} - \int_{\Omega^-} \delta\boldsymbol{\varepsilon}^- : \boldsymbol{\sigma} \tag{4.7}$$

où $\delta\boldsymbol{u}^+$ et $\delta\boldsymbol{u}^-$ désigne les degrés de liberté liés par une relation. $\delta\boldsymbol{u}^0$ désigne les degrés de liberté internes. L'équation d'équilibre 4.7 permet ainsi de distinguer entre les degrés de liberté frontières et les degrés de liberté internes. On impose une relation entre les déplacements des degrés de liberté frontières $\boldsymbol{u}^- = \boldsymbol{g}(\boldsymbol{u}^+)$ qui se traduit par une relation dans l'espace tangent $\delta\boldsymbol{u}^- = \partial_{\boldsymbol{u}^+}\boldsymbol{g} \cdot \delta\boldsymbol{u}^+$. En substituant dans l'équation 4.7, on obtient :

$$(\mathcal{R}, \delta\boldsymbol{u}) = (\mathcal{R}^+ + {}^t\partial_{\boldsymbol{u}^+}\boldsymbol{g} \cdot \mathcal{R}^-, \delta\boldsymbol{u}^+) + (\mathcal{R}^0, \delta\boldsymbol{u}^0) \tag{4.8}$$

La périodicité n'est qu'un type particulier de ces relations où on choisit la variété linéaire $\boldsymbol{v}^- = \boldsymbol{v}^+$ ce qui conduit en substituant dans 4.8 à :

$$(\mathcal{R}, \delta\boldsymbol{u}) = (\mathcal{R}^+ + \mathcal{R}^-, \delta\boldsymbol{u}^+) + (\mathcal{R}^0, \delta\boldsymbol{u}^0) \tag{4.9}$$

On a l'équivalence suivante :

$$\boldsymbol{\sigma}\cdot\boldsymbol{n} \text{ anti-périodique} \Leftrightarrow (\mathcal{R}^+, \delta\boldsymbol{u}^+) + (\mathcal{R}^-, \delta\boldsymbol{u}^-) = -\int_{\Omega^+} \boldsymbol{\varepsilon}^+ : \boldsymbol{\sigma} - \int_{\Omega^-} \boldsymbol{\varepsilon}^- : \boldsymbol{\sigma} \tag{4.10}$$

En outre, on a la relation suivante :

$$\left\{ \begin{array}{l} \boldsymbol{\sigma}\cdot\boldsymbol{n} \text{ anti-périodique} \\ \operatorname{div}\boldsymbol{\sigma} = 0 \end{array} \right. \Leftrightarrow (\mathcal{R}^0, \delta\boldsymbol{u}^0)+(\mathcal{R}^+, \delta\boldsymbol{u}^+)+(\mathcal{R}^-, \delta\boldsymbol{u}^-) = -\int_\Omega \boldsymbol{\varepsilon}^0 : \boldsymbol{\sigma} - \int_{\Omega^+} \boldsymbol{\varepsilon}^+ : \boldsymbol{\sigma} - \int_{\Omega^-} \boldsymbol{\varepsilon}^- : \boldsymbol{\sigma} = 0 \tag{4.11}$$

Dans ce cas, la matrice de rigidité s'écrit :

$$\partial_{(\boldsymbol{u}^+,\boldsymbol{u}^0)}({}^t\partial_{\boldsymbol{u}^+}\boldsymbol{g} \cdot \mathcal{R}^-)(h,k,l) = ({}^t\partial^2_{\boldsymbol{u}^+}\boldsymbol{g} \cdot \mathcal{R}^-)(h,l) + ({}^t\partial_{\boldsymbol{u}+}\boldsymbol{g} \cdot \partial_{\boldsymbol{u}^0}\mathcal{R}^-)(k,l) + ({}^t\partial_{\boldsymbol{u}}\boldsymbol{g} \cdot \partial_{\boldsymbol{u}^+}\mathcal{R}^-)(h,l) \tag{4.12}$$

$$+ ({}^t\partial_{\boldsymbol{u}^+}\boldsymbol{g} \cdot \partial_{\boldsymbol{u}^-}\mathcal{R}^- \cdot \partial_{\boldsymbol{u}^+}\boldsymbol{g})(h,l)$$

$$\partial_{(\boldsymbol{u}^+,\boldsymbol{u}^0)}\mathcal{R}^+(h,k,l) = \partial_{\boldsymbol{u}^+}\mathcal{R}^+(h,l) + (\partial_{\boldsymbol{u}^-}\mathcal{R}^+ \cdot \partial_{\boldsymbol{u}^+}\boldsymbol{g})(h,l) + (\partial_{\boldsymbol{u}^0}\mathcal{R}^+)(k,l) \tag{4.13}$$

Hyperréduction : cas périodique Pour calculer la contribution des réactions internes et de la matrice de rigidité dans le cas hyperréduit périodique, on reprend le raisonnement précédent, en particulier on s'appuie sur l'équation 4.9 en le produit scalaire est limité au domaine réduit Ω_Π :

$$(\mathcal{R}, \delta\boldsymbol{u})_{\Omega^+_\Pi \cup \Omega^-_\Pi} = (\mathcal{R}^+ + \mathcal{R}^-, \delta\boldsymbol{u}^+)_{\Omega^+_\Pi \cup \Omega^-_\Pi} + (\mathcal{R}^0, \delta\boldsymbol{u}^0)_{\Omega_\Pi} \tag{4.14}$$

On utilise ensuite la paramétrisation suivante :

$$(\boldsymbol{u}^+, \boldsymbol{u}^0) = \boldsymbol{h}(\boldsymbol{u}^+, \boldsymbol{q}) \tag{4.15}$$

On remarque cependant que les réactions internes et la matrice de rigidité hyperréduite périodique se déduisent des réactions internes et de la matrice de rigidité périodique. Pour assembler le problème hyperréduit périodique, différents objets interviennent dans la prise en compte de ces conditions :

- des degrés de liberté maîtres (+) et esclaves (−). Le programme **??** permet de créer les degrés de liberté maître et esclaves réduits par analyse des degrés de liberté maître et esclaves du maillage sous-jacent.
- un objet susceptible de traduire la relation, `class MPC`. Le diagramme de classe représentant la hiérarchie est donné figure **??**. La `class RELATIONSHIP_HANDLER` est destiné à gérer l'ensemble des `class RELATIONSHIP_WRAPPER`, i. e. des liens entre les degrés de liberté maîtres (+) et esclaves (−).

La projection sur l'espace hyperréduit comme indiqué dans l'algorithme **??** s'effectue en :

- substituant à la matrice $\boldsymbol{B}_e$ la matrice $\boldsymbol{B}_e\boldsymbol{A}_e$: $\boldsymbol{B}_e \leftarrow \boldsymbol{B}_e\boldsymbol{A}_e$. Cette substitution permet de faire la projection de l'espace réel sur l'espace réduit.
- substituant à la matrice ${}^t\boldsymbol{Z}_e\boldsymbol{Z}_e$ la matrice $\begin{pmatrix} {}^t\boldsymbol{Z}_e\boldsymbol{Z}_e & 0 \\ 0 & 0 \end{pmatrix}$. Cette substitution concerne le traitement de la réaction associée aux degrés de liberté macroscopiques. La réaction des degrés de liberté macroscopiques ne doit pas intervenir dans l'équilibre hyperréduit. La suppression des réactions des degrés de liberté macroscopiques s'imposent pour la raison que ceux-ci sont partiellement calculés quand on sélectionne un nombre réduit d'éléments dans le VER.
- substituant à la matrice ${}^t\boldsymbol{B}_e$ la matrice ${}^t\boldsymbol{A}{}^t\boldsymbol{Z}_e\boldsymbol{Z}_e{}^t\boldsymbol{B}_e$: ${}^t\boldsymbol{B}_e \leftarrow {}^t\boldsymbol{A}{}^t\boldsymbol{Z}_e\boldsymbol{Z}_e{}^t\boldsymbol{B}_e$
- créant les degrés de liberté macroscopiques associés $\delta\boldsymbol{q}_e$: $\delta\boldsymbol{v}_e \leftarrow \delta\boldsymbol{q}_e$

La prise en compte des conditions aux limites périodiques portant sur le degré de liberté j de l'élément e s'effectue en :

- substituant à la matrice $\boldsymbol{A}_e$ la matrice $\begin{pmatrix} \boldsymbol{A}_e & \boldsymbol{v}_e \end{pmatrix}$ avec $\boldsymbol{v}_e = (\delta_{ij})_{0\leq i<n}$: $\boldsymbol{A}_e \leftarrow \begin{pmatrix} \boldsymbol{A}_e & \boldsymbol{v}_e \end{pmatrix}$
- créant un degré de liberté réduit approprié, i. e. substituant $\delta\boldsymbol{q}_e \leftarrow \begin{pmatrix} \delta\boldsymbol{q}_e \\ \delta q_e \end{pmatrix}$

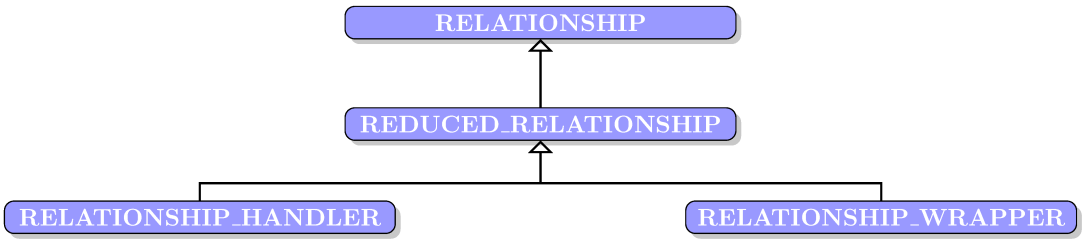


Figure 4.11 – *Composite pattern* destiné à traiter indistinctement les objets de fort granularité et l'ensemble d'une hiérarchie, celle des relations entre les degrés de liberté réduits

Algorithme 4 : Calcul des fonctions de forme réduites pour l'élément périodique

input : Compute $\boldsymbol{A}$

for $e \in \Omega_\Pi$ **do**

- Compute $\boldsymbol{B}_e$;
- Create the associated dofs $\delta\boldsymbol{v}_e$;
- $\boldsymbol{B}_e \leftarrow \begin{pmatrix} \boldsymbol{B}_e & \boldsymbol{I}_e \end{pmatrix}$;
- $\delta\boldsymbol{v}_e \leftarrow \begin{pmatrix} \delta\boldsymbol{v}_e \\ \delta\boldsymbol{E} \end{pmatrix}$;
- $\boldsymbol{A}_e \leftarrow \begin{pmatrix} \boldsymbol{A}_e & \boldsymbol{v}_e \end{pmatrix}$;
- $\delta\boldsymbol{q}_e \leftarrow \begin{pmatrix} \delta\boldsymbol{q}_e \\ \delta q_e \end{pmatrix}$;
- $\boldsymbol{B}_e \leftarrow \boldsymbol{B}_e\boldsymbol{A}_e$;
- ${}^t\boldsymbol{Z}_e\boldsymbol{Z}_e \leftarrow \begin{pmatrix} {}^t\boldsymbol{Z}_e\boldsymbol{Z}_e & 0 \\ 0 & 0 \end{pmatrix}$;
- ${}^t\boldsymbol{B}_e \leftarrow {}^t\boldsymbol{A}{}^t\boldsymbol{Z}_e\boldsymbol{Z}_e{}^t\boldsymbol{B}_e$;
- ${}^t\boldsymbol{A}{}^t\boldsymbol{Z}_e\boldsymbol{Z}_e{}^t\boldsymbol{B}_e : {}^t\boldsymbol{B}_e \leftarrow {}^t\boldsymbol{A}{}^t\boldsymbol{Z}_e\boldsymbol{Z}_e{}^t\boldsymbol{B}_e$;
- $\delta\boldsymbol{v}_e \leftarrow \delta\boldsymbol{q}_e$;

output : $\boldsymbol{B}_e$ & ${}^t\boldsymbol{B}_e$

??

```
void
RELATIONSHIP_WRAPPER::set_up_for_hyperreduction()
{
  for (int i=0; i != shadow_mesh.nb_dof(); i++)
  {
    const DOF* dof = shadow_mesh.get_dof(i);

    if(!rid->is_in(dof))
      continue;

    if(is_slave_dof(dof))
      add(dof);

    if(has_slave_dofs(dof))
      for (int k=0; k != slave_dofs[dof->rank()].size(); k++)
        add(slave_dofs[dof->rank()][k]);

  }
}
```

Figure 4.12 – Analyse des degrés de liberté maîtres/esclaves. Mise en place des degrés de liberté maîtres/esclaves réduits.

Limitation Plusieurs niveaux de récursion dans les relations de liaison entre degrés de liberté (MPC) ne sont pas pris en charge. Nous ne savons pas si ZSET lui-même est capable de les prendre en charge.

4.3.2 Validation sur un cas linéaire

La construction d'un domaine d'intégration périodique Sur la figure **??** est représenté un domaine d'intégration réduit sélectionné par l'algorithme DEIM. L'ajout des degrés de liberté correspondant aux noeuds situés à la frontière externe du domaine permet de prendre en compte les conditions aux limites. Si un noeud frontière est sélectionné par la DEIM, le noeud correspondant où s'appliquent les conditions de périodicité doit être sélectionné également et ses degrés de liberté introduits dans l'équilibre hyperréduit. L'introduction des degrés de liberté conduit à l'introduction des éléments hyperréduits correspondant afin que l'équilibre soit convenablement calculé en ces degrés de liberté. Les degrés de liberté externes, i. e. à la frontière du maillage peuvent être :

- omis s'ils sont partagés par des éléments extérieurs au domaine réduit
- pris en compte s'ils sont partagés seulement par des éléments appartenant au domaine réduit

Sur la figure **??** est présenté un calcul en élasticité linéaire d'un composite biphasique sollicité en conditions périodiques. La figure **??** présente le même composite simulé dans les mêmes conditions de chargement en hyperréduction. Dans les deux cas, le calcul a convergé en une itération et les isovaleurs des champs de contraintes sont identiques sur le domaine d'intégration réduit. Les conditions aux limites choisies en bord de domaine d'intégration réduit permettent représenter le chargement appliqué à l'ensemble de la cellule.

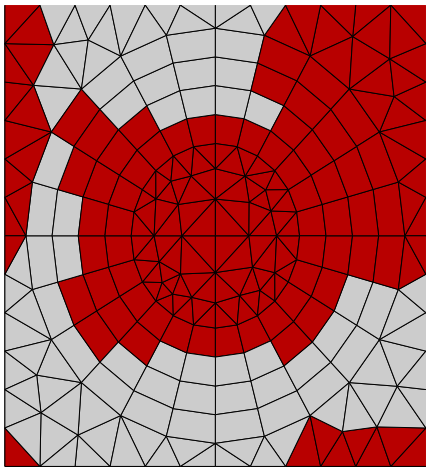


Figure 4.13 – Domaine réduit périodique

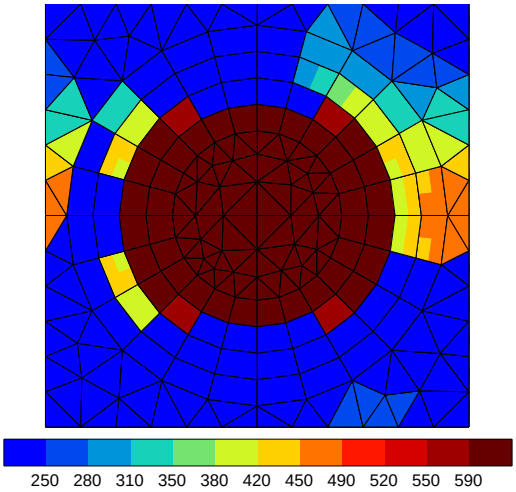


Figure 4.14 – Isovaleurs du champ de contraintes de sigmises (en MPa)

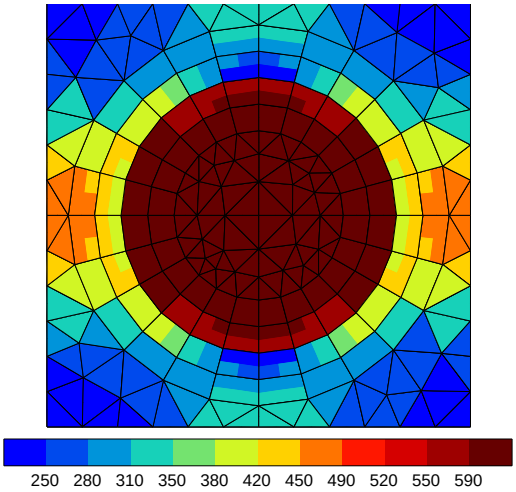


Figure 4.15 – Isovaleurs du champ de contraintes de sigmises (en MPa)

4.4 Conclusion

Dans ce chapitre, nous nous sommes efforcés de présenter une manière de tenir compte fidèlement des conditions aux limites de bord libre et périodiques au moment de la résolution de l'équilibre hyperréduit. Nous avons illustré nos propos par un exemple de sollicitation de plaque trouée linéaire et non-linéaire et par un cas de cellule périodique en linéaire. Etudier le cas linéaire suffit pour justifier une prise en compte correcte des conditions aux limites. Dans le cas nonlinéaire, l'amélioration de l'algorithme d'hyperréduction réside dans la synthèse de la base réduite. L'algorithme que nous avons retenu n'est clairement pas assez performant. Mais notre objectif n'était pas ici d'accélérer les calculs mais de rendre compte de certains types de conditions aux limites lors de la résolution de l'équilibre hyperréduit. Maintenant que nous avons déterminé une méthode de réduction sur une cellule périodique, le chapitre suivant sera consacré à l'examen de l'extension du calcul d'hyperréduction à une

population de volumes élémentaires représentatifs, i. e. plus brièvement un couplage entre l'hyperréduction et les méthodes EF^2.

5

Couplage EF^2 - Hyperréduction

Les méthodes d'hyperréduction ont connu un véritable essor ces dernières années à la fois en mécanique des matériaux et en mécanique des fluides [55]. Elles permettent en effet de diminuer, parfois notablement, les coûts de calcul grâce à la construction d'un modèle réduit représentatif du comportement de la structure. On peut les considérer comme une extension du principe de symétrie. Elles sont très efficaces quand il s'agit de traiter les problèmes de redondance dans les calculs. D'où l'idée de chercher à les utiliser dans le cadre des méthodes EF^2 qui accusent un coût de calcul souvent prohibitif.

Dans ce chapitre, on montre comment un problème EF^2 classique peut se reformuler à l'échelle microscopique en un problème de réduction de modèle multidimensionnel où le paramètre externe considéré n'est autre que la position courante macroscopique $\boldsymbol{X}$. C'est ce problème multidimensionnel micro qui est substitué à une loi de comportement analytique que l'on écrirait directement à l'échelle macroscopique. A ce titre, il doit renvoyer en chaque point de Gauss macroscopique, pour une valeur de déformation $\boldsymbol{E}$, non seulement une valeur $\boldsymbol{\Sigma}$ des contraintes macroscopiques qui entrent en jeu dans l'évaluation du résidu macro, mais aussi la valeur de la matrice tangente $\frac{\partial \Delta \boldsymbol{\Sigma}}{\partial \Delta \boldsymbol{E}}$ utilisée dans l'assemblage de la matrice de rigidité macro. Or si les contraintes macroscopiques sont accessibles à l'issue de chaque incrément du problème multidimensionnel puisque l'on connaît le champ de contraintes microscopiques, on ne dispose pas de l'information suffisante pour calculer les matrices tangentes macroscopiques en chaque point de Gauss, par la formulation même de l'algorithme d'hyperréduction [60]. Néanmoins, plusieurs méthodes peuvent être imaginées pour pallier cette difficulté, notamment celle qui consiste à utiliser une base réduite dans laquelle on exprimerait les coefficients de la matrice de rigidité multidimensionnelle micro comme suggéré dans [18]. Connaissant une estimation sur une base réduite de la matrice de rigidité micro, on pourrait alors en déduire une estimation des matrices tangentes aux points de Gauss macro. La construction d'un algorithme d'hyperréduction avec reconstruction des variables internes et des coefficients de la matrice de rigidité lacunaire est exposée dans [18]. Construire une base réduite pour la matrice tangente permet d'avoir une estimation de cette matrice tout en conservant une faible complexité à la dimension. Mais ce procédé peut s'avérer gourmand en mémoire et en temps de calcul dans les phases d'adaptation du modèle réduit au modèle réel qui conduisent à une réévaluation de la base. On propose donc ici une méthode alternative pour évaluer la matrice de rigidité macroscopique qui repose sur l'utilisation de l'algorithme BFGS [11].

Sommaire

5.1 Problème EF^2 multidimensionnel

Un schéma EF^2 classique est constitué de deux étapes : une étape de localisation qui permet de passer de l'échelle macroscopique à l'échelle inférieure suivie d'une étape d'homogénéisation qui permet de revenir à l'échelle macro. En chaque point de Gauss du maillage macroscopique, les déformations $\boldsymbol{E}$ sont prélevées et appliquées sur une cellule représentative de la microstructure locale du matériau que l'on sollicite en condition périodique. Ainsi, on cherche un champ de déplacement $\boldsymbol{u}$ de la forme $\boldsymbol{u} = \boldsymbol{E} \cdot \boldsymbol{x} + \boldsymbol{v}$, avec $\boldsymbol{v}$ un terme de fluctuation périodique, i. e. $\boldsymbol{v}$ prend des valeurs égales aux points frontières homologues du domaine $\mathcal{D}_i$ de la cellule i (on note par la suite $\mathcal{D} = \mathcal{D}_1 \cup ... \cup \mathcal{D}_p$). On impose en outre à $\boldsymbol{\sigma} \cdot \boldsymbol{n}$ d'être antipériodique, i. e. $\boldsymbol{\sigma} \cdot \boldsymbol{n}$ prend des valeurs opposées aux points frontières homologues du domaine $\mathcal{D}_i$ de la cellule i. Sous ces conditions l'unicité des champs microscopiques est assurée. C'est dans le cadre de ce type de schéma EF^2 qu'on se place dans la suite. Pour plus de précision sur cette approche EF^2, on renvoie à [31].

Pour permettre d'utiliser des techniques d'hyperréduction multidimensionnelle, une réécriture du schéma EF^2 a été effectuée. L'objectif est pouvoir traiter de front l'ensemble des problèmes éléments finis sur les cellules microscopiques. Ainsi, au lieu de traiter successivement les problèmes micro, un seul problème contenant l'ensemble des cellules est considéré. Les quantités commençant par Δ se réfèrent à des variations constatées à l'issue d'une itération du problème macroscopique tandis que les quantités débutant par δ relèvent de variations enregistrées à la suite d'une itération du problème microscopique. A chaque incrément de ce problème multidimensionnel micro, les conditions aux limites macroscopiques, i. e. l'ensemble des déformations macroscopiques $\Delta \boldsymbol{E}_i$ pour i parcourant l'ensemble des points de Gauss macro sont imposées puis, dans un deuxième temps, l'équilibre microscopique $\boldsymbol{R}^{micro}(\boldsymbol{u}) = \boldsymbol{F}^{micro}_{ext} - \boldsymbol{F}^{micro}_{int}(\boldsymbol{u}) = 0$ est calculé par une méthode de Newton-Raphson. Cette méthode nécessite d'assembler d'une matrice de rigidité $\boldsymbol{J}$ vérifiant :

$$\boldsymbol{J}^{micro} \delta \boldsymbol{u} = -\boldsymbol{R}^{micro} \tag{5.1}$$

L'hypothèse de séparation des échelles sous-jacente au schéma EF^2 implique que la matrice $\boldsymbol{J}$ est diagonale par bloc, chaque bloc correspondant à un problème microscopique. On peut écrire :

$$\boldsymbol{J}^{micro} = \begin{bmatrix} \boldsymbol{J}_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \boldsymbol{J}_p \end{bmatrix} \tag{5.2}$$

avec, si l'on considère des éléments de type périodique, $\boldsymbol{J}_i$ telle que :

$$\boldsymbol{J}_i \delta \boldsymbol{u} = \begin{bmatrix} \boldsymbol{J}_i^{(11)} & \boldsymbol{J}_i^{(12)} \\ \boldsymbol{J}_i^{(21)} & \boldsymbol{J}_i^{(22)} \end{bmatrix} \begin{bmatrix} \boldsymbol{E} + \Delta \boldsymbol{E} \\ \delta \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} + \Delta \boldsymbol{R} + \delta \boldsymbol{R} \\ 0 \end{bmatrix} \tag{5.3}$$

où $\boldsymbol{R}_{kl} = \boldsymbol{\Sigma}_{kl} = \int_{\mathcal{D}_i} \boldsymbol{\sigma}_{kl} dV$ si $i = j$ et $\boldsymbol{R}_{kl} = 2\boldsymbol{\Sigma}_{kl} = 2\int_{\mathcal{D}_i} \boldsymbol{\sigma}_{kl} dV$ si $i \neq j$, $\boldsymbol{\Sigma}$ désignant la contrainte macroscopique. $\mathcal{D}_i$ est le domaine de la cellule périodique relative au $i$ème point de Gauss.

Une fois l'équilibre microscopique mutidimensionnel établi, l'algorithme procède à la convergence vers l'équilibre macroscopique en deux étapes : l'évaluation du nouveau résidu et celle de la matrice de rigidité macro. A cette échelle, l'évaluation des forces internes $\boldsymbol{F}^{int}_{macro}$ entrant dans le calcul du résidu nécessite la connaissance des contraintes $\boldsymbol{\Sigma}$ en chaque point de Gauss macro. En effet :

$$\boldsymbol{F}^{macro}_{int} = \int_{\Omega} {}^{t}\boldsymbol{B}\boldsymbol{\Sigma}\,\mathrm{d}V \tag{5.4}$$

où Ω désigne le domaine macroscopique et $\boldsymbol{B}$ est la matrice des dérivées des fonctions de forme du modèle éléments finis macroscopique. De même, pour assembler la matrice de rigidité macro $\boldsymbol{K}^{macro}$, il est indispensable de calculer les matrices tangentes en chaque point de Gauss macro pour assembler la matrice de rigidité macro globale selon :

$$\boldsymbol{K}^{macro} = \int_{\Omega} {}^{t}\boldsymbol{B}\frac{\partial\Delta\boldsymbol{\Sigma}}{\partial\Delta\boldsymbol{E}}\boldsymbol{B}\,\mathrm{d}V \tag{5.5}$$

Les contraintes macroscopiques $\boldsymbol{\Sigma}$ sont calculées cellule par cellule en prenant simplement la moyenne des contraintes microscopiques. L'évaluation de la matrice tangente $\frac{\partial\Delta\boldsymbol{\Sigma}}{\partial\Delta\boldsymbol{E}}_i$ en un point de Gauss macro i est effectuée à partir de la $i$ème matrice bloc, en pratiquant un complément de Schur sur le système (éq. **??**) qui permet d'éliminer $\delta\boldsymbol{v}$ et d'en déduire la matrice tangente associée au $i$ème point de Gauss ($|\mathcal{D}_i|$ représente le volume de D_i) :

$$\left[\frac{\partial\Delta\boldsymbol{\Sigma}}{\partial\Delta\boldsymbol{E}}\right]_i = \frac{1}{|\mathcal{D}_i|}\left(\boldsymbol{J}^{(11)}_i - \boldsymbol{J}^{(12)}_i\boldsymbol{J}^{(22)-1t}_i\boldsymbol{J}^{(21)}_i\right) \tag{5.6}$$

Le schéma **??** résume le principe d'un calcul EF^2 multidimensionnel exposé dans cette section.

5.2 Principe de l'hyperréduction sur un cas simple

La discrétisation de la forme variationnelle de l'équilibre mécanique à résoudre s'écrit :

$$\boldsymbol{J}^{micro}\delta\boldsymbol{u} = -\boldsymbol{R}^{micro} \tag{5.7}$$

où $\boldsymbol{J}^{micro} \in M_n(\mathbb{R})$ est la matrice de rigidité et $\boldsymbol{R}^{micro} \in \mathbb{R}^n$ le résidu. L'analyse de la méthode éléments finis révèle deux sources de complexité algorithmique : l'assemblage de $\boldsymbol{J}^{micro}$, une matrice creuse de grande dimension et la résolution du système (éq. **??**). La réduction que l'on opère consiste à chercher la solution dans un espace réduit préalablement construit. Si l'on appelle $\boldsymbol{A} \in M_{n,N}(\mathbb{R})$ l'expression des vecteurs d'une base de cet espace dans la base des fonctions de forme élément fini classique, on se ramène à résoudre le système de faible dimension :

$${}^{t}\boldsymbol{A}\boldsymbol{J}^{micro}\boldsymbol{A}\delta\boldsymbol{q} = -{}^{t}\boldsymbol{A}\boldsymbol{R}^{micro} \tag{5.8}$$

où $\delta\boldsymbol{q} \in \mathbb{R}^N$. Mais cette réduction est souvent insuffisante. En effet, l'opération ${}^{t}\boldsymbol{A}\boldsymbol{R}$ consiste en $2Nn$ addition-multiplication. En outre, si l'on note ω la largeur de bande de la matrice $\boldsymbol{J}$, le nombre d'addition-multiplication à effectuer pour calculer ${}^{t}\boldsymbol{A}\boldsymbol{J}\boldsymbol{A}$ est $2\omega nN + 2N^2n$. Le calcul de $\boldsymbol{J}$ est en $O(\omega n)$. Finalement la complexité de l'algorithme réduit dépend linéairement du

nombre total de degrés de liberté n du problème microscopique complet. Pour abaisser cette complexité algorithmique, on remarque que $rg(\boldsymbol{J}^{micro}) = n$ alors que le système d'équation est de taille $N \ll n$ et qu'une matrice $\boldsymbol{J}^{micro}$ de rang N devrait suffire à rendre (éq. **??**) inversible. L'hyperréduction consiste alors à ne sélectionner que quelques degrés de liberté. L'ensemble des éléments qui possèdent ces degrés de liberté constituent un sous-ensemble de $\mathcal{D}$ qu'on appelle *domaine d'intégration réduit* et qu'on note $\mathcal{D}_\Pi$. On introduit une matrice de sélection $\boldsymbol{Z} \in M_{M,n}(\mathbb{R})$. M est le nombre de degrés de liberté sélectionnés. On doit s'assurer que $M \geq N$ pour rendre le système inversible. $Z_{ij} = 1$ si j est le $i$ème degré de liberté selectionnés et $Z_{ij} = 0$ dans tous les autres cas. On cherche à résoudre :

$$ {}^t\boldsymbol{A}\,{}^t\boldsymbol{Z}\boldsymbol{Z}\boldsymbol{J}^{micro}\boldsymbol{A}\delta q = -{}^t\boldsymbol{A}\,{}^t\boldsymbol{Z}\boldsymbol{Z}\boldsymbol{R}^{micro} \tag{5.9} $$

$\boldsymbol{Z}\boldsymbol{J}^{micro} \in M_{M,n}(\mathbb{R})$, $\boldsymbol{A} \in M_{n,N}(\mathbb{R})$ et ${}^t\boldsymbol{A}\,{}^t\boldsymbol{Z} \in M_{N,M}(\mathbb{R})$. Le nombre d'addition-multiplication requis pour calculer $({}^t\boldsymbol{A}\,{}^t\boldsymbol{Z})(\boldsymbol{Z}\boldsymbol{J}^{micro})\boldsymbol{A}$ est $2MN^2 + 2MN\omega$. Quand à la complexité du produit $({}^t\boldsymbol{A}\,{}^t\boldsymbol{Z})\boldsymbol{Z}\boldsymbol{R}^{micro}$, elle est en $O(NM)$. Généralement, $N \sim M$ ce qui conduit à affirmer que la complexité est indépendante du nombre d'éléments du problème complet mais, qu'en revanche, elle dépend linéairement du nombre de degrés de liberté M sélectionnés i.e, en définitive, du nombre d'éléments figurant dans le domaine d'intégration réduit $\mathcal{D}_\Pi$.

Une illustration du principe d'hyperréduction est proposée dans un cas simple. A l'échelle macroscopique, on considère une barre représentée par deux éléments comportant chacun quatre points de Gauss qui renvoient à autant de cellules microscopiques. Le maillage du problème microscopique multidimensionnel est représenté sur la figure **??**. Ces cellules correspondent à un composite métallique Sic/Ti modélisé en linéaire en 2D par une inclusion élastique ($E = 400000MPa$, $\nu = 0.19$) dans une matrice élastique ($E = 110000MPa$, $\nu = 0.3$). On sollicite la barre par deux tractions simples successives selon x et selon y en déformation plane. Les calculs sont menés en utilisant un modèle hyperréduit préalablement déterminé. Le modèle qui rend compte de ces deux modes de sollicitation est décrit par la donnée :

- d'une base réduite de deux éléments $(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2)$ correspondant aux deux modes tirés des deux tractions. Pour plus de précision concernant cette étape de construction, qui est fondée sur une méthode type P.O.D., nous renvoyons à [60] ;
- d'un domaine d'intégration $\mathcal{D}_\Pi$ réduit. Les degrés de liberté sélectionnés sont ceux portés par les noeuds se situant à l'intérieur du domaine.

Dans le cas élastique, ce modèle réduit permet de déterminer exactement les champs de contraintes, de déformation et de déplacement de la structure.

La figure **??** correspond au problème multidimensionnel microscopique. Chaque cellule renvoie à un point de Gauss du maillage macroscopique. Le domaine hyperréduit $\mathcal{D}_\Pi$ représenté en rouge est composé de 6 noeuds. Dans ce cas d'application, seules deux cellules sur huit entrent en jeu dans l'équilibre hyperréducit car seuls les noeuds situés strictement à l'intérieur du domaine sont considérés lors de l'assemblage du système. Comme la base réduite est de dimension 2, une condition nécessaire pour que le système **??** soit inversible est que $rg({}^t\boldsymbol{Z}\boldsymbol{Z}) \geq 2$, ce qui est vérifié dans ce cas où $rg({}^t\boldsymbol{Z}\boldsymbol{Z}) = 12$.

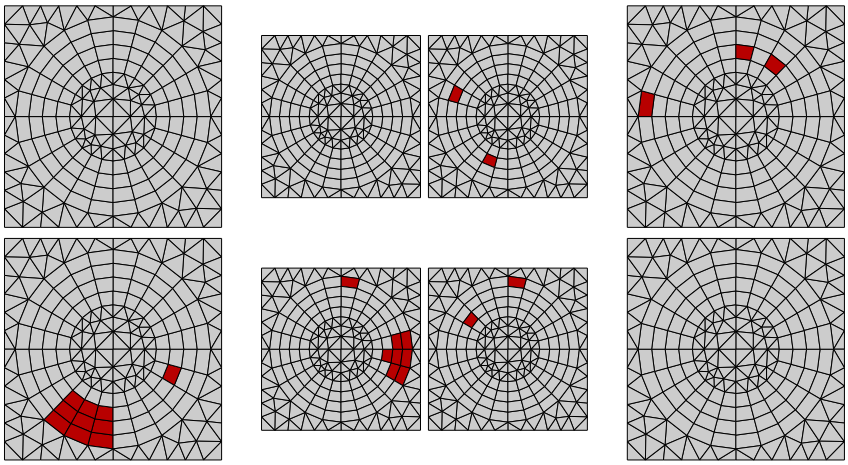


Figure 5.1 – Domaine d'intégration réduit du modèle

5.3 Nécessité de l'algorithme BFGS

Les algoritmes de type BFGS (Broyden, Fletcher, Goldfarb et Shanno) sont utilisés en éléments finis pour actualiser des matrices. Ils étaient utilisés pour actualiser des matrices de taille importante et offraient une alternative au calcul complet ce qui conduisait à une diminution des coût CPU. Mais avec l'augmentation continuelle des puissances des calculateurs leur intérêt pour cette application se tarit : la différence entre le calcul complet et l'actualisation n'est plus significative. Cependant, ce type d'algorithme reste pertinent quand l'on doit fournir une matrice actualisée sans pouvoir la recalculer car l'information dont on dispose est insuffisante. Ces algoritmes sont ainsi appliqués en calcul des structures mécaniques dans le cadre des approches non-intrusives non-linéaires [35], où l'accès à la matrice de rigidité globale du système mécanique est impossible. Dans ce cas, elle est inférée à partir de la matrice de rigidité élastique linéaire initiale, des incréments de déplacement et des forces externes. Lorsqu'à l'échelle micro, le calcul multidimensionnel est opéré sur base réduite, la matrice de rigidité est partiellement assemblée et il devient impossible d'en déduire la matrice de rigidité du système macroscopique. Il convient alors de l'estimer. Deux pistes ont été explorées :

- la première consiste à actualiser les matrices tangentes une à une et, à partir de ces matrices tangentes, à assembler la matrice de rigidité globale. L'actualisation est effectuée grâce à une méthode de type BFGS en se servant de la valeur des déformations et des contraintes macroscopiques qui, elles, sont toujours connues.
- la seconde, plus classique, est d'actualiser directement la matrice tangente macro à l'aide des déplacements macroscopiques à chaque itération.

5.3.1 Première méthode d'actualisation

Connaissant la matrice tangente en un point de Gauss $\boldsymbol{K}^{(i)}$ à la $i$ème itération, l'évaluation de $\boldsymbol{K}^{(i+1)}$ est menée sous une double condition :

- La matrice tangente en chaque point de Gauss à la $i+1$ème itération dénotée $\boldsymbol{K}^{(i+1)} = \frac{\partial \Delta \boldsymbol{\Sigma}}{\partial \Delta \boldsymbol{E}}^{(i+1)}$ présente des symétries mineures et majeures qui doivent être respectées lors de la mise à jour à partir de l'opérateur $\boldsymbol{K}^{(i)}$. Ainsi, l'on doit avoir, si l'on adopte les notations de Voigt :

$$\boldsymbol{K}_{kl}^{(i+1)} = \boldsymbol{K}_{lk}^{(i+1)} \tag{5.10}$$

- De surcroît, la matrice cherchée $\boldsymbol{K}^{(i+1)}$ doit être telle que :

$$\boldsymbol{K}^{(i+1)} \Delta \boldsymbol{\Sigma}^{(i+1)} = \Delta \boldsymbol{E}^{(i+1)} \tag{5.11}$$

  où $\Delta \boldsymbol{\Sigma}^{(i+1)}$ et $\Delta \boldsymbol{E}^{(i+1)}$ sont les variations prescrites à la fin de l'itération $i+1$

Ainsi $\boldsymbol{K}^{(i+1)}$ est cherchée comme solution du problème d'optimisation :

$$\min_{\substack{\boldsymbol{K}_{ij}=\boldsymbol{K}_{ji} \\ \boldsymbol{K}\Delta\boldsymbol{\sigma}^{(i+1)}=\Delta\boldsymbol{E}^{(i+1)}}} \|\boldsymbol{K} - \boldsymbol{K}^{(i)}\|_{\boldsymbol{W}} \tag{5.12}$$

où $\boldsymbol{W}$ est une matrice de pondération symétrique définie positive choisie de sorte que : $\boldsymbol{W}\boldsymbol{E}^{(i+1)} = \boldsymbol{\sigma}^{(i+1)}$ et $\|\cdot\|_{\boldsymbol{W}} = \|\boldsymbol{W}^{-1/2} \cdot \boldsymbol{W}^{-1/2}\|$.

Sous ces conditions, on démontre que le problème d'optimisation admet une unique solution $\boldsymbol{K}^{(i+1)}$, indépendante du $\boldsymbol{W}$ choisi et donnée par :

$$\boldsymbol{K}^{(i+1)} = \left(\mathcal{I} - \frac{\Delta\boldsymbol{\Sigma}\,{}^t\Delta\boldsymbol{E}}{{}^t\Delta\boldsymbol{\Sigma}\Delta\boldsymbol{E}}\right)\boldsymbol{K}^{(i)}\left(\mathcal{I} - \frac{\Delta\boldsymbol{E}\,{}^t\Delta\boldsymbol{\Sigma}}{{}^t\Delta\boldsymbol{\Sigma}\Delta\boldsymbol{E}}\right) + \frac{\Delta\boldsymbol{\Sigma}\,{}^t\Delta\boldsymbol{\Sigma}}{{}^t\Delta\boldsymbol{\Sigma}\Delta\boldsymbol{E}} \tag{5.13}$$

où on note $\mathcal{I}$ la matrice identité.

Cette méthode d'actualisation a été implémentée et testée. Elle ne permet en général pas de faire converger le problème macroscopique. En effet, la matrice de rigidité macroscopique à l'itération i vérifie par définition :

$$\boldsymbol{K}^{macro}\Delta\boldsymbol{u} = -\boldsymbol{R}^{macro} \tag{5.14}$$

ce qui n'est pas garanti ici si l'on actualise chaque matrice tangente et que l'on assemble le système complet ensuite. La convergence a été obtenue pour des incréments de temps très courts ou des maillages très raffinés. Une telle méthode n'étant donc pas satisfaisante, une autre a été envisagée.

5.3.2 Deuxième méthode d'actualisation

Cette méthode consiste à actualiser directement la matrice de rigidité macroscopique $\boldsymbol{K}^{macro}$ à partir des incréments de déplacement $\Delta\boldsymbol{u}$ calculés à l'itération précédente. L'algorithme est le même que celui de la section précédente. Il suffit simplement de remplacer $\boldsymbol{K}$ par $\boldsymbol{K}^{macro}$, $\Delta\boldsymbol{E}$ par $\Delta\boldsymbol{u}$ et $\boldsymbol{\Sigma}$ par $-\boldsymbol{R}^{macro}$ dans les formules (**??**), (**??**), (**??**) et (**??**). Cependant l'implémentation est différente. A l'étape initiale, la matrice de rigidité macroscopique est calculée. C'est cette matrice qui est, par la suite, mise à jour. En vue de pouvoir prendre en compte les simulations de fatigue, une bascule a été mise en place. Ainsi, quand survient une étape d'adaptation à l'échelle microscopique, il est possible de commander le recalcul complet de la matrice de rigidité macro $\boldsymbol{K}^{macro}$. Néanmoins, cette possibilité n'a pas été utilisée dans le cas d'application présenté.

5.4 Cas d'application

Une analyse paramétrique a été menée. Elle est proposée en guise d'illustration. On a repris la géométrie du cas simple (inclusion de fibre dans une matrice) en choisissant pour la matrice non plus une loi de comportement élastique linéaire, mais une loi de comportement viscoplastique avec un écrouissage cinématique et un écrouissage isotrope. Le maillage de la pièce macroscopique présente une structure en L, comptant seulement 3 éléments $2D$ de 4 points de Gauss chacun et renvoyant à un problème multidimensionnel micro qui comporte 12 cellules. Au total, le problème micro est un problème à 5000 degrés de liberté. La partie supérieure du L est déplacée de 0.8% de la hauteur en $30s$ de sorte d'obtenir des déformations de l'ordre de 0.8% et des vitesses de sollicitation de l'ordre de $10^{-3}s^{-1}$ (Fig. **??**).

A chaque fois, on compare un calcul avec utilisation de la méthode d'hyperréduction et actualisation de la matrice de rigidité macro par la méthode BFGS à un calcul EF^2 de référence lancé avec les mêmes paramètres et dont la matrice de rigidité macro est actualisée par la méthode BFGS. Les indications CPU pour les différents calculs sont regroupées dans les tableaux. On commence par effectuer un premier calcul hyperréduit ($S1$) pour constituer une base réduite en prenant une faible valeur de $R_0 = 50MPa$. On constate un gain d'un facteur 2 par rapport au calcul de référence. Ensuite, la base précédemment calculée est utilisée au cours d'un second calcul ($S3$) où $R_0 = 150MPa$. La plasticité étant alors moins marquée, le calcul converge dans les deux cas plus rapidement. Le gain de temps entre le calcul réduit et le calcul de référence est plus élevé (facteur 3.5) grâce à l'utilisation de la base réduite. Enfin, une dernière simulation ($S2$) est effectuée sur la base réduite existante à l'issue du calcul ($S3$). Un facteur d'accélération 4 est obtenu. Dans tous les cas, au maximum, un écart de l'ordre de $2-3\%$ a été enregistré sur les déformations plastiques cumulées. Sur les autres variables, l'accord avec le calcul EF^2 est encore plus grand.

La figure **??** présente le domaine d'intégration réduit utilisé dans la simulation (S2). Dans ce cas, le calcul éléments finis est réalisé uniquement sur ce domaine et avec les bases réduites préalablement calculées. La figure **??** permet de comparer le champ de σ_{eq} obtenu à l'issue

de la simulation (S2) et celui obtenu en sollicitant la même structure dans le cas d'un calcul EF^2 complet de référence. La différence relative entre les deux champs est inférieure à 3%.

	S1 ($R_0 = 50MPa$)	S2 ($R_0 = 100MPa$)	S3 ($R_0 = 150MPa$)
Réaction interne (calcul micro)	37s	20s	21s
Solver	1.0s	0.2s	0.2s
Temps utilisateur	42s	25s	25s

Table 5.1 – Indication CPU - calcul avec hyperréduction

	S1 ($R_0 = 50MPa$)	S2 ($R_0 = 100MPa$)	S3 ($R_0 = 150MPa$)
Réaction interne (calul micro)	90s	78s	70s
Solver	11.4s	10.8s	9.8s
Temps utilisateur	114s	100s	91s

Table 5.2 – Indication CPU - Calcul de référence

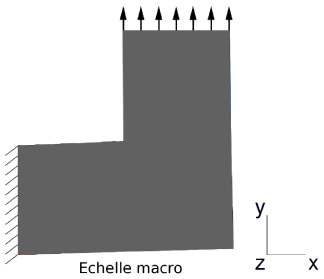


Figure 5.2 – Sollicitation de la pièce en L.

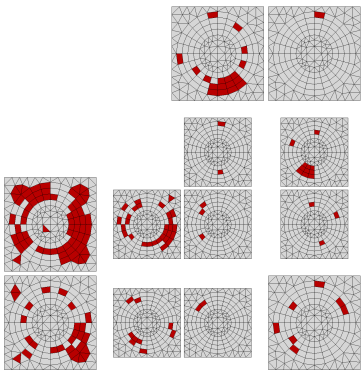


Figure 5.3 – Domaine d'intégration réduit en fin de simulation (S2).

5.5 Limites du Couplage EF^2/Hyperréduction

Les méthodes EF^2 consistent à simuler la loi matériau par un calcul éléments finis. Deux types de formulation ont été mises au point. La formulation historique qui consiste à assembler et résoudre cellule par cellule. Plus récemment une nouvelle formulation numérique du problème

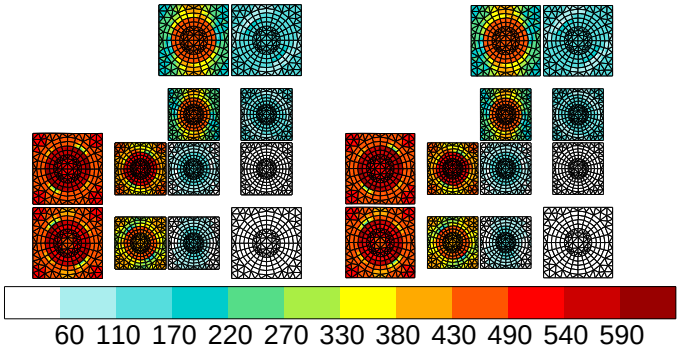


Figure 5.4 – Isovaleurs de σ_{eq} (en MPa), simulation avec hyperréduction (S2) à gauche, simulation de référence (S2) à droite

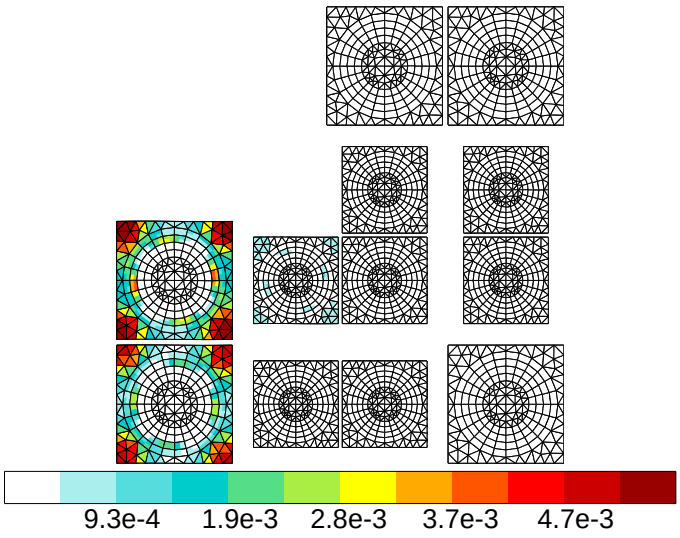


Figure 5.5 – Isovaleurs de evcum (en MPa), simulation avec hyperréduction (S2).

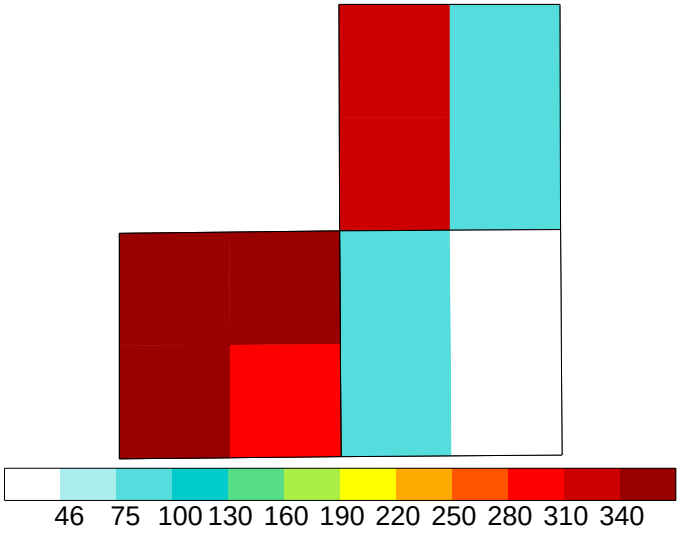


Figure 5.6 – Isovaleurs de σ_{eq} (en MPa), simulation de référence (S2).

EF^2 a été proposée. Elle consiste à assembler l'ensemble des cellules. Puis a résoudre le système linéaire diagonal par bloc en un coup. C'est sur ce deuxième problème que nous avons appliqué l'hyperréduction. Dans ce chapitre, nous montrons les limites de cette méthode en terme d'accélération des calculs.

Algorithme 5 : USUAL FE^2 ALGORITHM

Compute $\boldsymbol{E}$;
for $i \in \{cell_1, \ldots, cell_n\}$ **do**
  Compute $\boldsymbol{u} = \boldsymbol{E} \cdot \boldsymbol{x} + \boldsymbol{v}$
  Assemble the local stiffness matrix and compute the internal reactions :

$$\begin{pmatrix} \boldsymbol{K}_{EE} & \boldsymbol{K}_{Ev} \\ \boldsymbol{K}_{vE} & \boldsymbol{K}_{vv} \end{pmatrix} \begin{pmatrix} \delta\boldsymbol{E} \\ \delta\boldsymbol{v} \end{pmatrix} = \begin{pmatrix} \delta\boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix} \tag{5.15}$$

  with $\delta\boldsymbol{R} = \int_\Omega \boldsymbol{\sigma}$;
  Solve **??** ;
  Compute $\boldsymbol{\Sigma} = \langle \boldsymbol{\sigma} \rangle$ and $\frac{\partial \boldsymbol{\Sigma}}{\partial \boldsymbol{E}} = \boldsymbol{K}_{EE} - \boldsymbol{K}_{Ev}\boldsymbol{K}_{vv}^{-1}\boldsymbol{K}_{vE}$;
Compute the global stiffness matrix and the internal reactions :

$$\boldsymbol{K}\boldsymbol{u} = -\boldsymbol{R} \tag{5.16}$$

using elementary stiffness matrix and internal reactions

Algorithme 6 : FULL FE^2 ALGORITHM

for $i \in \{cell_1, \ldots, cell_n\}$ **do**
  Add local stiffness matrix and internal reaction to global stiffness matrix and internal reaction :

$$\begin{pmatrix} \boldsymbol{K}_{EE}^{(1)} & \boldsymbol{K}_{Ev}^{(1)} & & & \\ \boldsymbol{K}_{vE}^{(1)} & \boldsymbol{K}_{vv}^{(1)} & & & \\ & & \ddots & & \\ & & & \boldsymbol{K}_{EE}^{(n)} & \boldsymbol{K}_{Ev}^{(n)} \\ & & & \boldsymbol{K}_{vE}^{(n)} & \boldsymbol{K}_{vv}^{(n)} \end{pmatrix} \begin{pmatrix} \delta\boldsymbol{E}^{(1)} \\ \delta\boldsymbol{v}^{(1)} \\ \vdots \\ \delta\boldsymbol{E}^{(n)} \\ \delta\boldsymbol{v}^{(n)} \end{pmatrix} = \begin{pmatrix} \delta\boldsymbol{R}^{(1)} \\ \boldsymbol{0} \\ \vdots \\ \delta\boldsymbol{R}^{(n)} \\ \boldsymbol{0} \end{pmatrix} \tag{5.17}$$

Solve **??** ;
for $i \in \{cell_1, \ldots, cell_n\}$ **do**
  Compute $\boldsymbol{\Sigma} = \langle \boldsymbol{\sigma} \rangle$ and $\frac{\partial \boldsymbol{\Sigma}}{\partial \boldsymbol{E}} = \boldsymbol{K}_{EE} - \boldsymbol{K}_{Ev}\boldsymbol{K}_{vv}^{-1}\boldsymbol{K}_{vE}$;
Compute the global stiffness matrix and the internal reactions :

$$\boldsymbol{K}\boldsymbol{u} = -\boldsymbol{R} \tag{5.18}$$

using elementary stiffness matrix and internal reactions

Le raisonnement qui va suivre permet de souligner les limites de cette approche de couplage entre les méthodes EF^2 et l'hyperréduction multidimensionnelle.

A l'échelle macrosopique, une structure en coude a été maillée afin de limiter les concentrations de contrainte à l'intérieur du coude du L lorsque l'on procédera à son chargement (fig. **??**

à gauche). Chaque point de Gauss macroscopique renvoie à une cellule microscopique (fig. **??** à droite). Les cellules microsocopiques sont contrôlées en déformation et sollicitées en conditions périodiques.

Le tableau **??** présente, pour différents types de calculs effectués, le temps utilisateur constaté, et à défaut indique l'absence de convergence.

On constate qu'un calcul EF^2 multidimensionnel est 2 fois moins coûteux qu'un calcul EF^2 multidimensionnel micro où l'on commande la mise à jour quasi-systématique du modèle réduit. Le ratio du critère d'erreur servant à mesurer l'écart entre le modèle réduit et le modèle réel est pris ici très serré afin de s'assurer de la construction d'un espace réduit qui sera réinjecté par la suite mais aussi parce qu'il est impossible de converger en relâchant le ratio pour adapter moins souvent et calculer davantage avec le modèle réduit. Par ailleurs le doublement du temps de calcul n'est pas imputable au temps supplémentaire nécessaire pour construire le modèle réduit. Après chaque adaptation suit une phase d'hyperréduction, i. e. de calcul sur modèle réduit. Une fois le calcul réduit convergé entre en jeu l'estimation de l'erreur à chaque fin d'incrément. Après quelques incréments hyperréduits, dans le calcul (2) l'erreur devient trop importante et une adaptation est lancée. Cependant, avant d'adapter, les champs de contraintes, de déformations et de variables internes résultant du calcul hyperréduit cantonnés à une partie de la structure, le domaine d'intégration réduit, sont extrapolés à l'ensemble de la pièce. Un incrément de calcul éléments finis classique est ensuite exécuté à partir des champs reconstruits. Seulement, les itérations nécessaires pour faire converger cet incrément sont nombreuses car l'état mécanique réextrapolé est assez éloigné d'un état d'équilibre mécanique. Nonobstant, le calcul (2) parvient à converger vers un état final qui est identique à celui observé à la fin du calcul (1). L'application de l'hyperréduction a doublé le temps utilisateur nécessaire à l'exécution du calcul. On pourrait arguer que ce doublement du temps de calcul découle de ce que l'on a fixé un ratio de tolérance à l'erreur excessive et que le moindre écart est réprimé. Nous allons prouver qu'il est impossible de desserrer ce ratio d'erreur.

Le calcul (3) du tableau **??** présente un calcul EF^2 multidimensionnel hyperréduit. Les conditions aux limites imposées sur la structure macroscopique sont les mêmes que dans les cas (1) et (2). Par rapport au calcul hyperréduit (2), on interdit l'adaptation, i. e. le recalcul de la base si le modèle réduit est jugé inadéquat. On procède donc à un calcul purement hyperréduit sur le modèle réduit calculé à l'issue du (2) et réinjecté ici. A priori on s'attend à ce que le modèle (2) dont la tolérance à l'erreur était minimale ait produit un modèle réduit suffisamment pertinent pour pouvoir, soumis à des conditions de chargement identiques, converger vers le même état mécanique. Or il n'en est rien. On ne parvient plus à converger dès que la plasticité commence à se manifester sur les cellules situées à l'intérieur du coude. L'explication de ce phénomène numérique est à chercher au niveau des itérations. A chaque incrément macroscopique correspond une série d'itérations macroscopiques. A chaque itération macroscopique correspond un incrément microscopique. Et chaque incrément microscopique est constitué d'itérations microscopiques comme représentées dans **??**. Le modèle EF^2 hyperréduit permet, étant donné des valeurs de déformations macroscopiques $\boldsymbol{E}$ pour chaque cellule, d'accélérer le processus itératif microscopique. Mais ces valeurs $\boldsymbol{E}$ sont données par le processus itératif macroscopique. Elles dépendent donc du chemin de convergence emprunté par le calcul et non pas à un quelconque état mécanique. Les chemins

de convergence, i. e. l'ensemble des valeurs du champ de déformation macroscopique $\boldsymbol{E}$ au cours des itérations macroscopiques, sont suffisamment différents entre les modèles (2) et (3) pour rendre la base réduite inopérante dans le cas (3). La base réduite construite dans le cas (2) est dépendante du chemin de convergence et non pas simplement des états mécaniques du système. On peut ainsi parler de pollution de la base réduite.

Afin d'étayer notre thèse, nous avons mené une série complémentaire de 4 calculs supplémentaires. Le calcul (4), calcul multidimensionnel de référence, n'est pas un calcul EF^2. C'est un calcul éléments finis classique. Il est effectué à partir des états convergés du problème EF^2 (1). Plus précisément, à chaque fin d'incrément du problème macroscopique, c'est-à-dire quand l'état macroscopique se situe dans un état convergé, l'état microscopique ayant également convergé, on peut estimer que l'on a atteint un état mécanique réel aux deux échelles - à la précision commandé par les ratios de convergence près - ; à ce moment, on relève la valeur du champ de déformation macroscopique $\boldsymbol{E}$. On répète ainsi l'opération à chaque fin d'incrément macroscopique et on obtient ainsi le chemin des états mécaniques de déformations au cours du temps. On sollicite l'ensemble des cellules microscopiques avec les champs de déformation ainsi obtenus. Cela donne le calcul (4). Ce calcul converge plus rapidement d'un facteur 5 environ que le calcul (1). Ce n'est pas étonnant puique cinq itérations macroscopiques par incrément sont nécessaires en moyenne pour faire converger le calcul (1). Ce calcul converge vers la même solution microscopique que le calcul (1). Ce qui exclut toute source d'erreur de manipulation.

Le calcul (4) étant un calcul éléments finis classique, on peut lui appliquer sans difficulté particulière l'hyperréduction. On impose un ratio très serré pour l'estimateur qui rejettera ainsi tout divergence de modèle hyperréduit par rapport au calcul éléments finis réel. Une conséquence de ce ratio serré est que l'on constate que le temps utilisateur du calcul (5) est proche du temps utilisateur du calcul de référence (4) : presque à chaque incrément, une adaptation est réalisée. En se servant de la base calculée au cours de (5) dans le calcul (6) et, à nouveau, en interdisant l'adaptation, on obtient la convergence à l'issue des incréments vers la solution finale avec une excellente présicion, avec gain en temps d'un facteur 4.0. Le comportement de ce dernier calcul est tout à fait classique et correspond en ordre de grandeur aux gains qui sont couramment obtenus par le code d'hyperréduction fourni dans [60]. La convergence vers la solution avec une grande précision, même si elle est dénuée de preuve mathématique aujourd'hui, est un fait maintes fois constaté dans les expérimentations numériques qui ont pu être menées dans [60]. Il est à remarquer que la base réduite calculée à la volée dans (5) est constituée de vecteur d'états convergés, i. e. de réels états mécaniques.

Une dernière expérience numérique permettant d'asseoir définitivement notre raisonnement consiste à essayer de lancer un calcul EF^2 hyperréduit sans adaptation en prenant comme base réduite celle générée par (5) et en sollicitant la structure macroscopique de manière toujours identique. C'est l'objet du calcul (7). A la lumière des résultats précédents, on peut faire une prédiction : le calcul (7) ne doit pas converger. En effet, pour converger le calcul EF^2 emprunte un chemin d'états $\boldsymbol{E}$ non physiques qui non seulement ne figurent pas dans la base réduit (5) comprenant uniquement des états mécaniques, mais ne peuvent pas être exprimés en terme de ces états-là avec une précision suffisante donnée par le ratio d'adaptation. Effectivement (7) ne converge pas.

On parvient à la conclusion que :

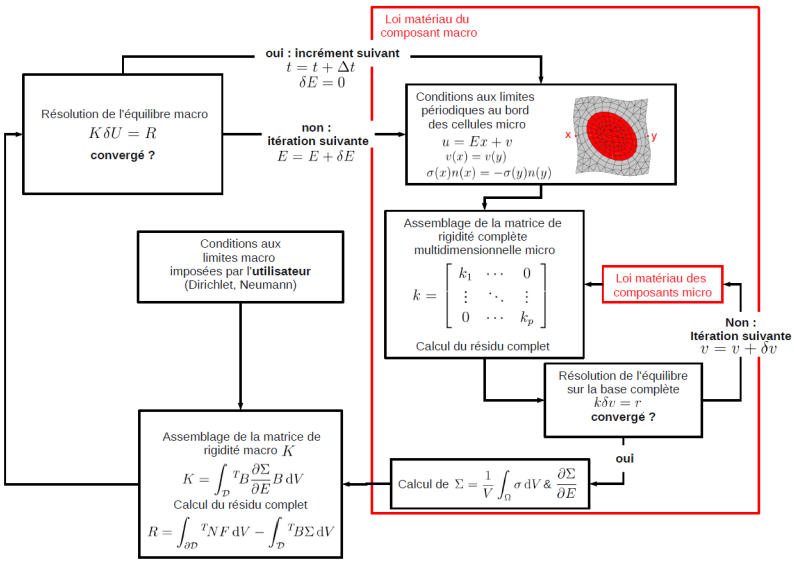


Figure 5.7 – Méthode EF^2 multidimensionnelle

- si le ratio d'adaptation est trop serré, i. e. on choisit d'adapter dès que le modèle réduit semble commettre un très faible écart avec le modèle éléments finis complet, on ne parvient pas à accélérer les calculs, parce que l'on se met à suivre un chemin d'état non convergés, suffisamment différents les uns des autres. C'est le cas (2).
- si le ratio d'adaptation est trop peu serré, le modèle réduit ne représente plus le modèle éléments finis complet.

En résumé :

Dans le cas des problèmes EF^2 multidimensionnels hyperréduits non triviaux, il n'existe pas de ratio d'adaptation qui permette de représenter par un modèle réduit le modèle éléments finis sous-jacent. Dit plus simplement : la méthode de couplage EF^2/hyperréduction ne permet pas d'accélérer les calculs.

Afin d'examiner l'efficacité de la méthode d'hyperréduction multidimensionnelle appliquée à ce cas de calcul EF^2, on se place dans la cas le plus simple à imaginer et le plus optimal possible : une éprouvette rectangulaire en traction. A l'échelle macroscopique, l'éprouvette est suffisamment discrétisée pour que la partie microscopique du calcul EF^2 que l'on cherche à réduire soit pourvu d'un nombre de degrés de liberté suffisamment grand. Les temps de calcul mesurés ont alors un sens. La figure **??** représente la partie microscopique d'un tel calcul rassemblant à peu près 400000 degrés de liberté.

Le tableau **??** regroupe les résultats en terme de temps utilisateur. Le calcul EF^2 est le calcul EF^2 multidimensionnel réalisé sans utiliser l'hyperréduction multidimensionelle à l'échelle microscopique. C'est le plus coûteux en temps. Le calcul EF^2 avec adaptation désigne le calcul EF^2 multidimensionnel pour lequel on a appliqué l'hyperréduction à l'échelle microscopique. On autorise l'adaptation, i. e., l'enrichissement des bases réduites, si la convergence n'a pas été atteinte. On observe une diminution d'un facteur à peu près 2.5 du temps de calcul par rapport au cas complet. Le dernier calcul, dit EF^2 sur base réduite, consiste à mener le calcul en réinjectant la base réduite calculée lors de l'étape 2 et en interdisant l'adaptation.

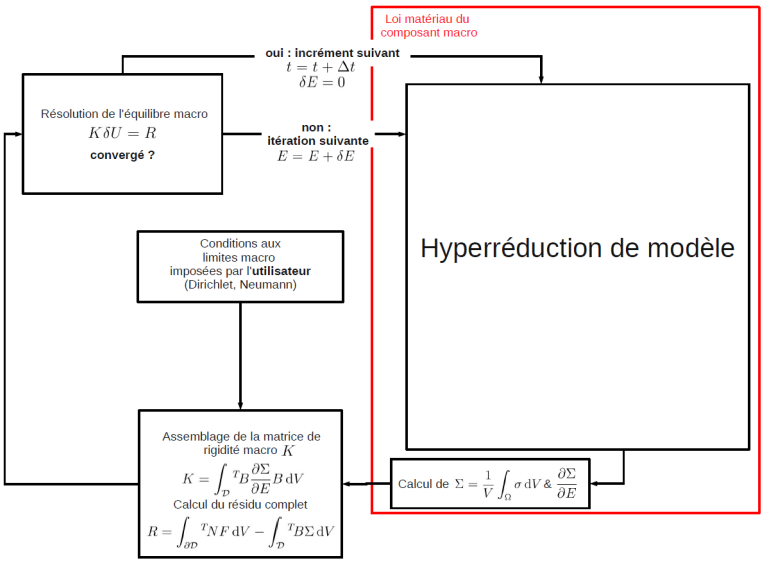


Figure 5.8 – Méthode EF^2 multdimensionnelle hyperréduite

N°	Type de calcul	Temps CPU (en s)
(1)	EF^2 de référence	6335
(2)	EF^2 avec adaptation	2544
(3)	EF^2 sur base réduite	1577

Table 5.3 – Temps de calcul couplage EF^2/hyperréduction pour le cas de l'éprouvette rectangulaire en traction

Ce dernier calcul converge vers un état final qui est identique à celui obtenu lors de l'étape 1 ce qui prouve que le modèle réduit préablement déterminé a correctement enregistré la succession des champs de déplacements de la structure et est suffisant pour modéliser ce trajet de chargement. Sur les structures en coude déjà présentées, la convergence n'était pas réalisée dans le cas EF^2 sur base réduite. Les différences entre ces deux cas résident dans l'absence dans le cas présent de cellules sur lesquelles se concentrent la contrainte et de ce que les cellules, à chaque incrément, au cours des itérations de convergence, reçoivent toutes les mêmes sollicitations E. Par conséquent l'espace vectoriel des sollicitations de l'ensemble des cellules périodiques conserve toujours une faible dimension ce qui fait que le problème multidimensionnel microscopique est hyperréductible et donne la propriété de convergence.

Le temps de calcul a encore diminué par rapport à l'étape 2 ce qui est conforme à ce que l'on pouvait espérer en interdisant l'adaptation. Néanmoins le rapport de temps utilisateur entre le calcul EF^2 de référence et le calcul EF^2 sur base réduite, de l'ordre de 4.0, ne s'est pas accru par rapport aux calculs à 5000 degrés de liberté que nous avons menés pour prouver la faisabilité de la méthode, où ce facteur était aussi de l'ordre de 4.0-5.0.

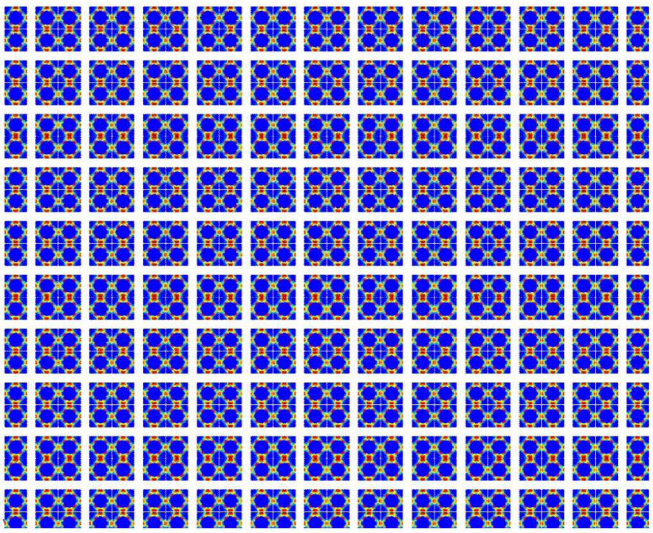


Figure 5.9 – Cellules périodiques à l'échelle microscopique d'un calcul EF^2 dont l'échelle macro est une éprouvette rectangulaire en traction simple.

N°	Type de calcul	Temps CPU (en s)
(1)	EF^2 de référence	5156
(2)	EF^2 avec adaptation	10368
(3)	EF^2 calculé sur base réduite à l'issu de (2)	ne converge pas
(4)	Calcul multidimensionnel de référence	1055
(5)	Calcul multidimensionnel avec adaptation	946
(6)	Calcul multidimensionnel sur base réduite	268
(7)	Calcul EF^2 sur base réduite calculée à l'issue de (5)	ne converge pas

Table 5.4 – Temps de calcul couplage EF^2/Hyperréduction

5.6 Conclusion

Dans ce chapitre, nous avons mis en place une stratégie de couplage EF^2-hyperréduction multidimensionnelle. Nous avons prouvé que cette stratégie fonctionne pour les calculs EF^2 à faible nombre de cellules élémentaires, i. e. à faible nombre de degrés de liberté. Cependant nous avons montré qu'en dehors de quelques cas triviaux de calcul à nombre de degrés de liberté plus important, une telle stratégie est mise en échec.

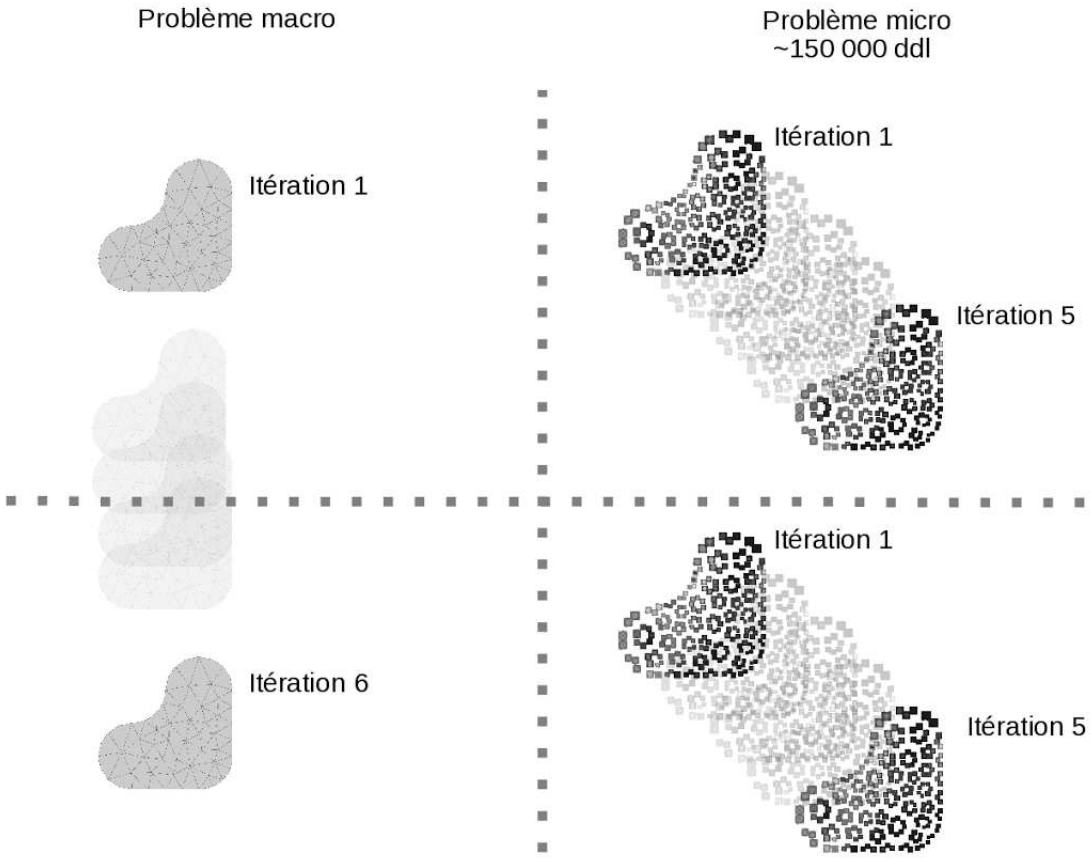


Figure 5.10 – Principe d'un calcul EF^2 non-linéaire

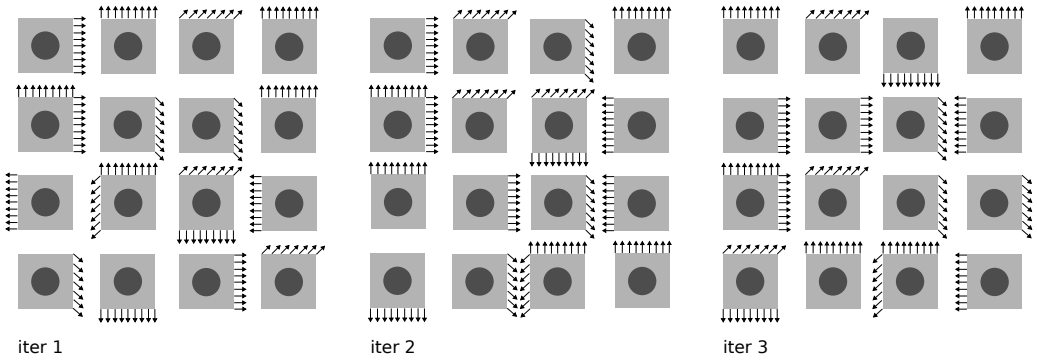


Figure 5.11 – Conditions aux limites en bord de cellules.

Conclusion & Perspectives

Dans cette thèse, une architecture logicielle orientée objet pour le plugin d'hyperréduction a été proposée puis appliquée sur des cas tests élémentaires. Les conditions de bord libre ont été prises en compte dans l'équilibre hyperréduit. Une étude théorique a été menée pour mettre en place des conditions aux limites périodiques dans le code de l'hyperréduction. Ces conditions aux limites périodiques sont respectées au niveau de l'équilibre hyperréduit et non simplement au moment de l'extrapolation. Un cas test sur une plaque trouée avec un comportement mécanique non-linéaire a été réalisé ainsi que sur un VER en condition périodique avec une loi élastique linéaire pour les deux phases. Deux algorithmes de mise à jour du modèle réduit ont été mis en oeuvre ainsi qu'un indicateur d'erreur. Les difficultés rencontrées résident dans la faible performance de l'hyperréduction de modèle *a priori*. Cet algorithme ne nous semble pas satisfaisant.

Dans ce travail, nous nous sommes intéressés à une méthode de réduction, l'hyperréduction de modèle, en tant que telle, puis à sa capacité à réduire un problème EF^2. Afin d'améliorer le couplage EF^2/hyperréduction, nous nous sommes lancés dans une reprogrammation intégrale du code de l'hyperréduction afin d'augmenter la lisibilité et de le rendre plus performant. Ce code pourra être réutilisé pour des travaux ultérieurs en réduction de modèle. S'il permet de faire de l'hyperréduction, sa conception séparant nettement une phase de calcul *online* d'une phase de construction du modèle réduit *offline* et son architecture très modulaire permettront de l'employer pour développer de nouvelles méthodes de réduction de modèle autre que le couplage $EF^2/hyperréduction$. Nous croyons d'ailleurs que l'hyperréduction en tant qu'algorithme adaptatif n'est pas assez performant et que cet aspect adaptatif sera abandonné dans les développements futurs de la méthode au profit d'une construction plus élaborée du modèle réduit. Des modèles réduits seront élaborés, et stockés lors d'une phase de calcul *offline* très coûteuse. L'utilisateur chargera ensuite le modèle réduit correspondant au besoin de sa simulation dans une phase *online* clairement séparée. Le plugin d'hyperréduction que nous avons développé a été pensé pour répondre à ce type de besoin.

Dans les perspectives à court terme, une optimisation des algorithmes implémentés et un cas d'application 3D permettraient de mettre en oeuvre la méthode sur des structures réelles. Enfin une étude de la possibilité d'implémentation en tant que *user-element* dans Abaqus de l'élément hyperréduit rapprocherait sans consteste ces développements du monde industriel : le modèle réduit ne s'exprimerait plus en terme de base mais en terme d'éléments qui reflètent un certain nombre d'états.

A plus long terme, les perspectives consistent en une réflexion sur le choix des *snapshots*. L'utilisation de l'élément réduit ouvre la voie à l'utilisation directe de bases de déformation plastique et élastique. Au lieu d'utiliser les fonctions de forme réduite $\boldsymbol{BA}$, une simple projection d'une base réduite des déformations plastiques sur l'élément est nécessaire. Une mise au point d'un estimateur d'erreur pour l'hyperréduction reste à nos yeux une perspective plus lointaine et plus hasardeuse. Une réduction du système dynamique de la loi matériau serait pertinent. En effet, l'hyperréduction ne réduit que l'équilibre. C'est la réduction de cet équilibre qui conduit à n'intégrer la loi matériau qu'en quelques points de Gauss. On ne peut pas parler de réduction pour la loi matériau dont il faudrait astreindre les variables

internes à évoluer dans un espace réduit en posant : $\begin{bmatrix} \dot{\boldsymbol{\sigma}}_r \\ \dot{\boldsymbol{\alpha}}_r \end{bmatrix} = \boldsymbol{F}(\boldsymbol{\sigma}_r, \boldsymbol{\alpha}_r, \dot{\boldsymbol{\varepsilon}})$. Cela éviterait notamment la phase de reconstruction des champs avant l'adaptation. Plus profondément, si la méthode LATIN approche la solution éléments finis par une technique de Lagrangien augmenté convergeant en itérant entre un état vérifiant la loi de comportement et un autre l'équilibre mécanique, si les méthodes classiques employées dans les codes de calcul non-linéaire utilisent une méthode de substitution de la loi de comportement dans l'équilibre mécanique, une troisième voie semble possible : celle consistant en la résolution simultanée de l'équilibre et de la loi de comportement au sein d'un seul et unique processus itératif. La formulation et la réduction de ce système permettrait notamment d'attaquer à nouveau le problème du couplage EF^2/hyperréduction avec une chance d'accélérer le calcul. La mise en oeuvre d'un tel projet dépasse largement le cadre de ce travail. .

Bibliographie

[1] David Abrahams and Aleksey Gurtovoy. *C++ template metaprogramming : concepts, tools, and techniques from Boost and beyond*. Pearson Education, 2004.

[2] Andrei Alexandrescu. *Modern C++ design : generic programming and design patterns applied*. Addison-Wesley, 2001.

[3] Amine Ammar, B Mokdad, Francisco Chinesta, and Roland Keunings. A new family of solvers for some classes of multidimensional partial differential equations encountered in kinetic theory modelling of complex fluids : Part ii : Transient simulation using space-time separated representations. *Journal of Non-Newtonian Fluid Mechanics*, 144(2) :98–121, 2007.

[4] David Amsallem and Charbel Farhat. Stabilization of projection-based reduced-order models. *International Journal for Numerical Methods in Engineering*, 91(4) :358–377, 2012.

[5] David Amsallem, Matthew J Zahr, and Charbel Farhat. Nonlinear model order reduction based on local reduced-order bases. *International Journal for Numerical Methods in Engineering*, 92(10) :891–916, 2012.

[6] Wilkins Aquino. An object-oriented framework for reduced-order models using proper orthogonal decomposition (pod). *Computer Methods in Applied Mechanics and Engineering*, 196(41) :4375–4390, 2007.

[7] Patricia Astrid. Fast reduced order modeling technique for large scale ltv systems. In *American Control Conference, 2004. Proceedings of the 2004*, volume 1, pages 762–767. IEEE, 2004.

[8] Patricia Astrid, Siep Weiland, Karen Willcox, and Ton Backx. Missing point estimation in models described by proper orthogonal decomposition. *Automatic Control, IEEE Transactions on*, 53(10) :2237–2251, 2008.

[9] Wolfgang Bangerth, Ralf Hartmann, and Guido Kanschat. deal. ii—a general-purpose object-oriented finite element library. *ACM Transactions on Mathematical Software (TOMS)*, 33(4) :24, 2007.

[10] Maxime Barrault, Yvon Maday, Ngoc Cuong Nguyen, and Anthony T Patera. An 'empirical interpolation'method : application to efficient reduced-basis discretization of partial differential equations. *Comptes Rendus Mathematique*, 339(9) :667–672, 2004.

[11] Klaus-Jürgen Bathe. *Finite element procedures*. Klaus-Jurgen Bathe, 2006.

[12] J Besson and R Foerch. Large scale object-oriented finite element code design. *Computer Methods in Applied Mechanics and Engineering*, 142(1) :165–187, 1997.

[13] Peter Binev, Albert Cohen, Wolfgang Dahmen, Ronald DeVore, Guergana Petrova, and Przemyslaw Wojtaszczyk. Convergence rates for greedy algorithms in reduced basis methods. *SIAM Journal on Mathematical Analysis*, 43(3) :1457–1472, 2011.

[14] Joshua Bloch. *Effective java*. Pearson Education India, 2008.

[15] Tan Bui-Thanh, Karen Willcox, and Omar Ghattas. Parametric reduced-order models for probabilistic analysis of unsteady aerodynamic applications. *AIAA journal*, 46(10) :2520–2529, 2008.

[16] Tan Bui-Thanh, Karen Willcox, Omar Ghattas, and B van Bloemen Waanders. Goal-oriented, model-constrained optimization for reduction of large-scale systems. *Journal of Computational Physics*, 224(2) :880–896, 2007.

[17] Kevin Carlberg, Charbel Bou-Mosleh, and Charbel Farhat. Efficient non-linear model reduction via a least-squares petrov–galerkin projection and compressive tensor approximations. *International Journal for Numerical Methods in Engineering*, 86(2) :155–181, 2011.

[18] Kevin Carlberg, Charbel Farhat, Julien Cortial, and David Amsallem. The gnat method for nonlinear model reduction : effective implementation and application to computational fluid dynamics and turbulent flows. *Journal of Computational Physics*, 242 :623–647, 2013.

[19] Saifon Chaturantabut and Danny C Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing*, 32(5) :2737–2764, 2010.

[20] Saifon Chaturantabut and Danny C Sorensen. A state space error estimate for pod-deim nonlinear model reduction. *SIAM Journal on numerical analysis*, 50(1) :46–63, 2012.

[21] Francisco Chinesta, Amine Ammar, and E Cueto. Proper generalized decomposition of multiscale models. *International Journal for Numerical Methods in Engineering*, 83(8-9) :1114–1132, 2010.

[22] Francisco Chinesta, Amine Ammar, Adrien Leygue, and Roland Keunings. An overview of the proper generalized decomposition with applications in computational rheology. *Journal of Non-Newtonian Fluid Mechanics*, 166(11) :578–592, 2011.

[23] Vivien Courtier. *Réduction de modèle et simplification de l'intégration de loi de comportement pour la prévision de la durée de vie*. PhD thesis, Ecole Polytechnique X, 2013.

[24] George J Dvorak. Transformation field analysis of inelastic composite materials. *Proceedings of the Royal Society of London. Series A : Mathematical and Physical Sciences*, 437(1900) :311–327, 1992.

[25] Richard Everson and Lawrence Sirovich. Karhunen–loeve procedure for gappy data. *JOSA A*, 12(8) :1657–1664, 1995.

[26] Dominique Eyheramendy and Thomas Zimmermann. Programmation orientée objet appliquée à la méthode des éléments finis : dérivations symboliques, programmation automatique. *Revue Européenne des éléments finis*, 4(3) :327–360, 1995.

[27] Antonio Falcó and Anthony Nouy. Proper generalized decomposition for nonlinear convex problems in tensor banach spaces. *Numerische Mathematik*, 121(3) :503–530, 2012.

[28] F Feyel. Multiscale non linear fe2 analysis of composite structures : Fiber size effects. *Le Journal de Physique IV*, 11(PR5) :Pr5–195, 2001.

[29] Frédéric Feyel. *Application du calcul parallèle aux modèles à grand nombre de variables internes*. PhD thesis, 1999.

[30] Frédéric Feyel. Some new technics regarding the parallelisation of zébulon, an object oriented finite element code for structural mechanics. *ESAIM : Mathematical Modelling and Numerical Analysis*, 36(05) :923–935, 2002.

[31] Frederic Feyel and Jean-Louis Chaboche. Multi-scale non-linear fe2 analysis of composite structures : damage and fiber size effects. *Revue européenne des Eléments Finis : NUMDAM'00 issue*, 10 :449–472, 2001.

[32] Felix Fritzen and Thomas Boehlke. Reduced basis homogenization of viscoelastic composites. *Composites Science and Technology*, 76 :84–91, 2013.

[33] Felix Fritzen and T Böhlke. Three-dimensional finite element implementation of the nonuniform transformation field analysis. *International Journal for Numerical Methods in Engineering*, 84(7) :803–829, 2010.

[34] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns : elements of reusable object-oriented software*. Pearson Education, 1994.

[35] Lionel Gendre. *Approche globale/locale non-intrusive : application aux structures avec plasticité localisée*. PhD thesis, École normale supérieure de Cachan-ENS Cachan, 2009.

[36] Brian Göetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, and Doug Lea. *Java concurrency in practice*. Addison-Wesley, 2006.

[37] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http ://eigen.tuxfamily.org, 2010.

[38] Frédéric Hecht. New development in freefem++. *Journal of Numerical Mathematics*, 20(3-4) :251–266, 2012.

[39] Stephen C Johnson. *Yacc : Yet another compiler-compiler*, volume 32. Bell Laboratories Murray Hill, NJ, 1975.

[40] Pierre Kerfriden, Pierre Gosselet, Sondipon Adhikari, and Stephane Pierre-Alain Bordas. Bridging proper orthogonal decomposition methods and augmented newton–krylov algorithms : an adaptive model order reduction for highly nonlinear mechanical problems. *Computer Methods in Applied Mechanics and Engineering*, 200(5) :850–866, 2011.

[41] Pierre Kerfriden, Juan-José Ródenas, and SP-A Bordas. Certification of projection-based reduced order modelling in computational homogenisation by the constitutive relation error. *International Journal for Numerical Methods in Engineering*, 97(6) :395–422, 2014.

[42] Michael Kirby and Lawrence Sirovich. Application of the karhunen-loeve procedure for the characterization of human faces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(1) :103–108, 1990.

[43] Dirk Klindworth, Martin A Grepl, and Georg Vossen. Certified reduced basis methods for parametrized parabolic partial differential equations with non-affine source terms. *Computer Methods in Applied Mechanics and Engineering*, 209 :144–155, 2012.

[44] Pierre Ladevèze, J-C Passieux, and David Néron. The latin multiscale computational method and the proper generalized decomposition. *Computer Methods in Applied Mechanics and Engineering*, 199(21) :1287–1296, 2010.

[45] Mark Lutz. *Learning python*. " O'Reilly Media, Inc.", 2013.

[46] Scott Meyers. *More effective C++ : 35 new ways to improve your programs and designs*. Pearson Education, 1995.

[47] Scott Meyers. *Effective STL : 50 specific ways to improve your use of the standard template library*. Pearson Education, 2001.

[48] Scott Meyers. *Effective C++ : 55 specific ways to improve your programs and designs*. Pearson Education, 2005.

[49] Jean-Claude Michel and Pierre Suquet. Nonuniform transformation field analysis. *International journal of solids and structures*, 40(25) :6937–6955, 2003.

[50] Anthony Nouy. A priori model reduction through proper generalized decomposition for solving time-dependent partial differential equations. *Computer Methods in Applied Mechanics and Engineering*, 199(23) :1603–1626, 2010.

[51] Goury O. Bridging analytical and computational homogenisation for non linear multiscale problems : a reduced order modelling approach for a damage problem. 2013.

[52] Jean-Charles Passieux. *Approximation radiale et méthode LATIN multiéchelle en temps et en espace*. PhD thesis, École normale supérieure de Cachan-ENS Cachan, 2008.

[53] B Patzák and Z Bittnar. Design of object oriented finite element code. *Advances in Engineering Software*, 32(10) :759–767, 2001.

[54] Christophe Prud'homme, Dimitrios V Rovas, Karen Veroy, Luc Machiels, Yvon Maday, Anthony T Patera, and Gabriel Turinici. Reliable real-time solution of parametrized partial differential equations : Reduced-basis output bound methods. *Journal of Fluids Engineering*, 124(1) :70–80, 2002.

[55] Etienne Pruliere, Amine Ammar, Nadia El Kissi, and Francisco Chinesta. Recirculating flows involving short fiber suspensions : numerical difficulties and efficient advanced micro-macro solvers. *Archives of Computational Methods in Engineering*, 16(1) :1–30, 2009.

[56] Sophie Roussette, Jean-Claude Michel, and Pierre Suquet. Nonuniform transformation field analysis of elastic–viscoplastic composites. *Composites Science and Technology*, 69(1) :22–27, 2009.

[57] Gianluigi Rozza, DBP Huynh, and Anthony T Patera. Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations. *Archives of Computational Methods in Engineering*, 15(3) :229–275, 2008.

[58] D Ryckelynck. A priori hyperreduction method : an adaptive approach. *Journal of Computational Physics*, 202(1) :346–366, 2005.

[59] David Ryckelynck and Djamel Missoum Benziane. Multi-level a priori hyper-reduction of mechanical models involving internal variables. *Computer Methods in Applied Mechanics and Engineering*, 199(17) :1134–1142, 2010.

[60] David Ryckelynck, Florence Vincent, and Sabine Cantournet. Multidimensional< i> a priori</i> hyper-reduction of mechanical models involving internal variables. *Computer Methods in Applied Mechanics and Engineering*, 225 :28–43, 2012.

[61] Lawrence Sirovich. Turbulence and the dynamics of coherent structures. part i : Coherent structures. *Quarterly of applied mathematics*, 45(3) :561–571, 1987.

[62] Bjarne Stroustrup. *The C++ programming language*. Pearson Education, 2013.

[63] Vladimir Temlyakov. *Greedy approximation*, volume 20. Cambridge University Press, 2011.

[64] K Veroy and AT Patera. Certified real-time solution of the parametrized steady incompressible navier–stokes equations : rigorous reduced-basis a posteriori error bounds. *International Journal for Numerical Methods in Fluids*, 47(8-9) :773–788, 2005.

[65] Karen Veroy, Dimitrios V Rovas, and Anthony T Patera. A posteriori error estimation for reduced-basis approximation of parametrized elliptic coercive partial differential equations :"convex inverse" bound conditioners. *ESAIM : Control, Optimisation and Calculus of Variations*, 8 :1007–1028, 2002.

[66] Stefan Volkwein. Model reduction using proper orthogonal decomposition. *Lecture Notes, Institute of Mathematics and Scientific Computing, University of Graz. see http ://www. uni-graz. at/imawww/volkwein/POD. pdf*, 2011.

[67] Bin Wen and Nicholas Zabaras. A multiscale approach for model reduction of random microstructures. *Computational Materials Science*, 63 :269–285, 2012.

[68] Stephen J Wright and Jorge Nocedal. *Numerical optimization*, volume 2. Springer New York, 1999.

Méthode EF2 et hyperréduction de modèle : vers des calculs massifs à l'échelle micro

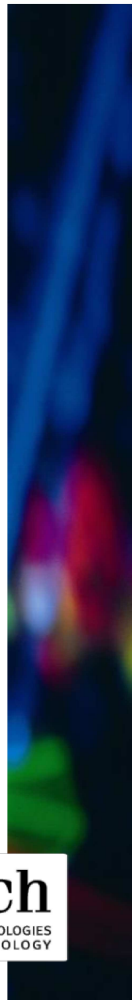
RESUME : Des méthodes de réduction de modèle sont utilisées pour diminuer le coût de calcul associé à des analyses paramétriques de structures qui requièrent un très grand nombre de simulations quasi-identiques. Parmi ces méthodes, l'hyperréduction de modèle est efficace pour attaquer les problèmes de mécanique non-linéaire. Une approche orientée objet de cette méthode dans le cadre d'un code éléments finis modulaire a été développée. L'architecture logicielle s'appuie sur un algorithme qui se déroule en deux étapes : une étape *offline* dans laquelle le modèle réduit est construit à partir d'états du système mécanique et une étape *online* de calcul réduit qui exploite le modèle réduit. La structure du code qui repose sur l'utilisation d'un élément réduit permet d'améliorer la performance, de simplifier la prise en main et de favoriser sa réutilisation dans les développements futurs de la méthode. En outre, la méthode d'hyperréduction est revisitée et améliorée : des bases réduites vectorielles et tensorielles sont mises en oeuvre pour traiter les champs de contraintes et de variables internes des calculs éléments finis non-linéaires. En particulier, l'accent est mis sur la prise en compte des conditions aux limites périodiques et des conditions de bord libre. Dans cette démarche, les conditions aux limites au bord du domaine réduit sont imposées dans l'équation de l'équilibre mécanique réduit. Des exemples d'inclusions élastiques fibre/matrice sont fournis ainsi qu'un calcul complet adaptatif non-linéaire sur plaque perforée. Pour prendre en compte les effets de la microstructure, les méthodes éléments finis au carré (EF2) divisent le problème mécanique en deux échelles. A l'échelle microscopique, les équations de comportement sont intégrées sur le volume élémentaire représentatif (VER) sollicité en condition périodique. Le comportement de la structure macroscopique est déterminé par homogénéisation. Une méthode d'hyperréduction multidimensionnelle est appliquée au problème microscopique constitué de l'ensemble des volumes élémentaires représentatifs. On se sert d'un algorithme de Broyden-Fletcher-Goldfarb-Shanno (BFGS) pour mettre à jour les matrices tangentes macroscopiques en chaque point de Gauss. On parvient ainsi à diminuer le temps de calcul sur des modèles de faible dimension. Cependant, quand le nombre de degrés de liberté augmente, on démontre que l'hyperréduction de modèle multidimensionnelle ne parvient pas à réduire suffisamment les coûts de calcul.

Mots-clés : Réduction de modèle non-linéaire ; Hyperréduction ; Calculs multi-échelles ; Méthodes EF2 ; Architecture logicielle modulaire

FE2 Method and Hyperreduction : towards intensive computations at micro scale

ABSTRACT : Model Order Reduction (MOR) methods are used to cope with high computational costs typically involved in parametric analysis of structures requiring a huge number of almost similar simulations. Among them, a so-called hyperreduction method suitable for non-linear mechanical finite element (FE) problems is studied. An objected-oriented approach to deal with it in the framework of a FE software is carried out. The software design takes advantage of a two-level process : a so-called offline computation step in which the reduced model is set up based on collected snapshots of mechanical system states and an online high-speed reduced computation which runs the reduced model. The code design relying on a reduced element is expected to enhance performance, to give a clearer view over the process and to favour code reuse in subsequent developments of the method. Futhermore, the hyperreduction method is reviewed and is deeply improved : vector and tensor bases are introduced to deal with non-scalar fields which arise in non-linear mechanical FE problems and the mechanical balance is ensured in the extrapolation phase. A particular emphasis is placed on the treatment of free and periodic boundary conditions. In this approach, the boundary conditions at the edge of the reduced integration domain are enforced in the reduced balance equations. Numerical toy examples of elasticity fiber/matrix inclusions as well as a full adaptative non-linear simluation are provided to show the capabilities of the implementation. To take into account microstructural behaviors, FE2 methods consist in splitting the computation into two scales. At the micro scale the material constitutive equations are integrated over periodic RVEs. The behavior of the macro structure is carried out by a homogeneized process. A multidimensional hyperreduction method is applied to the massive micro problem composed of the set of the periodic RVEs. A BFGS algorithm is used to update the macro tangent matrices at each integration Gauss point. Some speed-ups are recorded for low dimensional models. However, as the number of degrees of freedom increases, the multidimensional hyperreduction method is proved to be far less efficient to cut computational costs down.

Keywords : Non-linear model reduction ; Hyperreduction ; Multiscale simulations ; FE2 ; Object-Oriented software design



MINES ParisTech

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY