



Grenoble INP - Ensimag
École nationale supérieure d'informatique et de mathématiques appliquées

Pré Rapport

Laboratoire Jacques-Louis Lions

Interpolation polynomiale en grande dimension pour un problème de construction de bibliothèque neutronique

SIALA Rafik

3A - Specialization MMIS /M2 MSIAM

du 03/04/2017 au 01/09/2017 (22 semaines)

Laboratoire Jacques-Louis Lions
UPMC, 4 place Jussieu
75005 Paris, France

Tuteur entreprise
COHEN Albert, cohen@ann.jussieu.fr
MADAY Yvon, maday@ann.jussieu.fr
Tuteur Ensimag:
MAITRE Emmanuel,
emmanuel.maitre@imag.fr

Contents

1	Introduction	4
2	Présentation de la structure d'accueil	5
3	Présentation de la problématique du projet	5
3.1	Présentation du contexte	5
3.2	Objectifs attendus, problème à résoudre	7
4	Solution technique mise en œuvre	7
4.1	Description de la solution envisagée	8
4.1.1	Interpolation mono-variée et tensorisation:	8
4.1.2	Construction hiérarchique de l'opérateur d'interpolation:	9
4.1.3	Interpolation adaptative et séquence de points d'interpolation:	12
4.2	Implémentation de la solution	16
5	Évaluation de l'efficacité de la méthode	21
5.1	Présentation du modèle de décomposition de Tucker	21
5.2	Résultats des approximations de fonctions analytiques	24
5.3	Résultats des approximations de données réelles	28
5.3.1	Déscription des cas de test	28
5.3.2	Résultats numériques	33
6	Impressions personnelles	38
7	Prise en compte de l'impact environnemental et sociétal	38
8	Conclusion	39
	References	40

List of Figures

1	Schéma à 2 étapes basé sur la structure multi-échelle du noyau du réacteur	6
2	Schéma à 2 étapes pour la simulation du cœur du réacteur	6
3	Exemples de polynômes d'interpolation de Lagrange	8
4	Influence de l'ajout d'un point d'interpolation sur une seule direction	9
5	Interpolation utilisant des polynomes de Lagrange définis globalement	13
6	Arbre représentatif des points d'interpolation en 1D	14
7	Interpolation utilisant des fonctions quadratiques par morceaux	15
8	Codage de huffman de chaque nœud de l'arbre	17
9	Grille 2D des points de Leja	19
11	Construction des fonctions de base directionnelles pour toutes les directions en utilisant une extension du modèle de décomposition de Karhunen-Loève (KL)	22
12	Discrétisation adaptative (grilles de Tucker) utilisées pour une extension de la décomposition de Karhunen-Loève afin de construire les fonctions de base tensorielles directionnelles direction par direction	23
14	Comparaison des erreurs relatives ($\text{pcm} = 10^{-5}$) pour les fonctions f_1 , f_2 , f_3 , et f_4 en plusieurs dimensions.	25
16	Comparaison des temps d'exécutions des deux méthodes pour les fonctions f_1 , f_2 , f_3 et f_4 en plusieurs dimensions.	25
18	Exemple de points d'interpolation obtenu par l'algorithme AI	31
20	Projection des points d'interpolation, obtenus en interpolant Σ_t^1 de type MOX, sur chaque direction.	31
22	Comparaison des erreurs relatives (pcm) pour la section efficace $\nu\Sigma_f^2$	34
23	Comparaison des erreurs relatives (pcm) pour la section efficace $\Sigma_{s0}^{2 \rightarrow 1}$	35
24	Comparaison des erreurs d'approximation (pcm) de la réactivité pour le type d'assemblage MOX	35
25	Courbes de convergence pour chaque section efficace de type MOX	36

List of Tables

1	Comparaison des résultat de l'interpolation de la fonction f_2 en 2D en utilisant différents types de fonctions de bases et différents types de points d'interpolation	26
2	Intervalles des paramètres	28
3	Discrétisation de la grille multilinéaire, de la grille de Tucker et de la grille construite par l'algorithme AI	30
4	Comparaison de la précision des 3 méthodes sur une grille de référence de 6000 points	33
5	Comparaison du nombre de points de calcul entre le modèle de décomposition de Tucker, le modèle multilinéaire et la méthode d'interpolation adaptative	37
6	Comparaison de la taille du stockage entre le modèle de décomposition de Tucker, le modèle multilinéaire et la méthode d'interpolation adaptative	37

Remerciements

J'aimerais remercier mes encadrants de stage, Albert Cohen, Yvon Maday ainsi que Nora Aissiouene pour le suivi régulier de mon travail, les responsabilités qu'ils m'ont confiées et la confiance qu'il ont témoignée à mon égard.

Je tiens également à remercier l'ensemble de l'équipe au laboratoire J-L Lions qui m'ont accueilli et facilité mon intégration. Enfin, je souhaite remercier Emmanuel Maitre, mon tuteur de stage à l'ENSIMAG, pour avoir accepté de suivre le déroulement de mon stage.

1 Introduction

Un nouveau cadre pour les calculs neutroniques de base développé à EDF R&D, dans le département de SINETICS (SImulation NEutronique, Technologie de l'Information, Calcul Scientifique) est basé sur la résolution de l'équation de Boltzmann (ou une approximation) pour les neutrons. Cette équation nécessite en entrée des sections efficaces qui modélisent les interactions entre les neutrons induits par la fission et les noyaux provenant soit du combustible soit du modérateur. Ces sections efficaces (au nombre de plusieurs milliers) dépendent de d paramètres locaux (dits paramètres de rétroaction), tels que la densité de l'eau, la concentration en bore, la température du carburant, le brûlage du combustible, etc. Elles sont stockées dans des « bibliothèques neutroniques » et doivent être consultées régulièrement tout au long de la simulation quand les paramètres de rétroaction évoluent.

Dans le cadre d'une thèse CIFRE [3] qui vient d'être soutenue au Laboratoire Jacques-Louis Lions, une nouvelle méthode [4] basée sur l'utilisation de bases tensorielles a été développé. Ces méthodes utilisent des fonctions directionnelles adaptées aux fonctions qu'on veut représenter et qui permettent de diminuer le stockage, le coût des calculs pour reconstruire les sections efficaces avec une très grande précision.

L'objectif du stage est de comparer cette étude avec une approche alternative (méthodes parcimonieuses) proposée récemment au Laboratoire J.-L. Lions par Albert Cohen [2]. Le stage porte sur la mise en œuvre de ces méthodes parcimonieuses pour le cas particulier de l'approximation des sections efficaces. Il s'agit pour certaines d'entre elles de méthodes non-intrusives (donc basées sur des évaluations obtenues par un code qui peut être vu comme une boîte noire, ce qui est bien adapté à la situation). Elles font appel à des techniques de représentations parcimonieuses dans le but de tirer parti des anisotropies potentielles dans la fonction qu'on veut capturer (certaines variables pouvant en particulier être plus sensibles que d'autres) et des propriétés de régularité en fonction des différentes variables.

Mots clé: sections efficaces, méthodes parcimonieuses, interpolation multivariée ...

2 Présentation de la structure d'accueil

Le laboratoire, créé en 1969, porte le nom de son fondateur Jacques-Louis Lions. Il s'agit maintenant d'une unité de recherche conjointe à l'Université Pierre et Marie Curie, à l'université Paris Diderot et au Centre National de la Recherche Scientifique.

Le Laboratoire Jacques-Louis Lions constitue le plus grand laboratoire de France et l'un des principaux au monde pour la formation et la recherche en mathématiques appliquées.

Il accueille l'activité de deux masters deuxième année ce qui représente une centaine d'étudiants. Ses axes de recherche recouvrent l'analyse, la modélisation et le calcul scientifique haute performance de phénomènes représentés par des équations aux dérivées partielles. Fort d'environ 100 enseignants-chercheurs, chercheurs, ingénieurs, personnels administratifs permanents ou émérites, et d'autant de doctorants ou post-doctorants, le LJLL collabore avec le monde économique et avec d'autres domaines scientifiques à travers un large spectre d'applications: dynamique des fluides ; physique, mécanique et chimie théorique ; contrôle, optimisation et finance ; médecine et biologie ; traitement du signal et des données.

3 Présentation de la problématique du projet

3.1 Présentation du contexte

Afin d'exploiter au mieux son parc nucléaire, la R&D d'EFD est en train de développer une nouvelle chaîne de calcul appelée ANDROMEDE, pour simuler le cœur des réacteurs nucléaires avec des outils à l'état de l'art. L'un des éléments de cette chaîne est le code neutronique COCAGNE, dont l'un des objectifs est de résoudre numériquement l'équation du **transport neutronique** (ou l'une de ses approximations), permettant ainsi d'obtenir des grandeurs physiques d'intérêt pour décrire le comportement du réacteur, tel que: le flux neutronique, le facteur de multiplication effectif, la réactivité ...

La résolution de cette équation nécessite une grande quantité de données physiques, en particulier les **sections efficaces**.

En neutronique, les sections efficaces représentent la probabilité d'interaction d'un neutron incident avec les noyaux cibles, pour différents types d'interaction. Dans une simulation neutronique, les sections efficaces peuvent être représentées comme des fonctions dépendant de plusieurs paramètres physiques. Ces paramètres sont utilisés pour décrire les conditions thermo-hydrauliques et la configuration du cœur du réacteur, tel que: la température du combustible, concentration en bore, niveau de xénon, burnup, ... Ainsi, les sections efficaces sont des fonctions multivariées définies sur un espace appelé **l'espace de phase des paramètres**.

Dans la simulation d'un cœur complet, le nombre de valeurs des sections efficaces est de l'ordre de plusieurs milliards. Leur détermination demande un calcul complexe et lent. Pour résoudre ce problème, un schéma en deux étapes est proposé afin de réduire la complexité des calculs et d'accomplir la simulation du cœur. Ce schéma est basé sur la structure multi-échelle du noyau du réacteur: noyau contenant des assemblages,

assemblages contenant des tiges/cellules (voir 1).

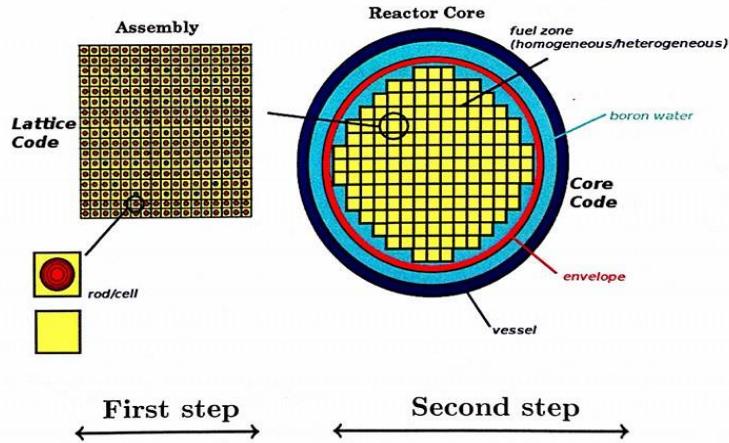


Figure 1: Schéma à 2 étapes basé sur la structure multi-échelle du noyau du réacteur

Par conséquent, 2 étapes sont effectuées séparément par deux codes:

- **Code réseau (APOLLO2):** permet de pré-calculer les sections efficaces sur des points présélectionnés dans l'espace de phase des paramètres (sur chaque assemblage). L'équation du transport neutronique est résolue de manière précise grâce à des discrétisations spatiales et énergétiques (calcul hors ligne). L'information obtenue sur les sections efficaces est stockée dans des fichiers (appelés les **bibliothèques neutroniques**).
- **Code de cœur (COCAGNE):** permet d'évaluer les sections efficaces en n'importe quel point du cœur par interpolation multivariée. Les informations stockées dans les bibliothèques neutroniques sont utilisées comme données (points d'interpolation) pour résoudre l'équation du transport neutronique au niveau du cœur du réacteur.

Le schéma ci-dessous (2) illustre le modèle de reconstruction des sections efficaces.

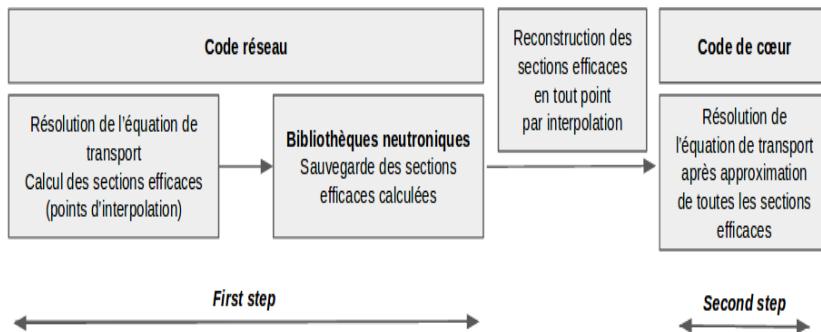


Figure 2: Schéma à 2 étapes pour la simulation du cœur du réacteur

Avec ce modèle, le nombre de points de discrétisation augmente exponentiellement en fonction du nombre de paramètres ou quand on ajoute des points sur un des axes. En effet, si on suppose que les sections efficaces dépendent de d paramètres et que chaque paramètre est discrétisé par n points sur l'axe correspondant, dans ce cas le nombre de calculs hors ligne et la taille des bibliothèques neutroniques sont de l'ordre de $O(n^d)$.

3.2 Objectifs attendus, problème à résoudre

Avec les contraintes industrielles imposées à EDF, le modèle de l'interpolation multi-variée devient très coûteux en mémoire et en temps de calcul pour des situations complexes à cause de l'augmentation rapide et exponentielle du nombre de points de calcul. Ces situations peuvent arriver, par exemple, dans le cas accidentel où de nouveaux paramètres s'ajoutent (température de l'eau, taille des lames d'eau) et/ou quand le domaine de calcul s'étend (avec des domaines de définition plus larges).

Afin de dépasser les limitations de ce modèle, l'objectif de ce stage est de développer un modèle de reconstitution des sections efficaces tout en répondant aux exigences suivantes

- **Calculs hors ligne:**

- utiliser moins de pré-calculs et ceci en choisissant les points de discréétisation de manière astucieuse.
- stocker moins de données pour la reconstruction des sections efficaces.

- **Calculs en ligne:**

- avoir une bonne précision pour l'évaluation des sections efficaces.

4 Solution technique mise en œuvre

Le but de cette partie est de présenter une méthode d'interpolation polynomiale adaptative en grande dimension. Cette méthode permet d'approximer une fonction multivariée avec une forte précision tout en utilisant moins de données.

Dans la section précédente, on a vu que les sections efficaces sont des fonctions multivariées définies sur un espace appelé l'espace de phase des paramètres qu'on note \mathcal{P} .

On peut donc modéliser une section efficace par une fonction f tel que:

$f : \mathcal{P} = P^d \rightarrow V_\Lambda$ avec P un compact de \mathbb{R} (ou \mathbb{C}) et V l'espace des valeurs prises par les sections efficaces.

En pratique, pour $y \in \mathcal{P}$, on peut approcher $f(y)$ avec un code très coûteux. Le but est d'éviter ce lent calcul, tout simplement, en considérant une approximation de f en y , qu'on note $\tilde{f}(y)$. Pour cela, il faut commencer par choisir un certain nombre de points $y^i \in \mathcal{P}, i \in 1..m$. Ensuite on évalue $f_i = f(y^i) = f(y_1^i, \dots, y_d^i)$ en faisant m appel au code coûteux. Finalement, on utilise $y^1, \dots, y^m, f_1, \dots, f_m$ pour fabriquer \tilde{f} par interpolation.

L'objectif est donc de construire un opérateur d'interpolation I_Λ qui permet de reconstruire une fonction f définie sur \mathcal{P} à valeurs dans V_Λ .

Les méthodes d'interpolation polynomiale d'ordre supérieur construisent des approximations de la forme $u_\Lambda(y) = \sum_{\nu \in \Lambda} u_\nu y^\nu$ avec $\Lambda \in \mathcal{F}$ un ensemble fini de multi-indices $\nu = (\nu_j)_{j \geq 1} \in \mathcal{F}$ et $y^\nu = \prod_{j \geq 1} y_j^{\nu_j}$.

Remarque: En dimension finie ($d < \infty$), l'ensemble d'indices \mathcal{F} coïncide avec \mathbb{N}_0^d . Les u_Λ sont choisis dans l'espace V_Λ , défini par $V_\Lambda = \text{vect} \left\{ \sum_{\nu \in \Lambda} v_\nu y^\nu : v_\nu \in V \right\}$

4.1 Description de la solution envisagée

Afin d'approximer f en tout point, on va procéder par interpolation multivariée. Dans cette partie, on va revoir le principe de l'interpolation de Lagrange en $1D$, puis en dimension quelconque. Ensuite, on présentera une méthode d'interpolation adaptative en grande dimension (moins coûteuse) basée sur une construction hiérarchique de l'opérateur d'interpolation.

4.1.1 Interpolation mono-variée et tensorisation:

- **Cas $d = 1$:** Soit $(y_k)_{k \geq 0}$ une séquence de points deux à deux distincts. On note I_k l'opérateur d'interpolation polynomiale associé à la séquence $\{y_0, \dots, y_k\}$. Supposons que la fonction f qu'on veut interpoler est à valeurs dans \mathbb{R} (ou \mathbb{C}). On a alors,

$$I_k(f) = \sum_{i=0}^k f(y_i) l_i^k$$

avec $l_i^k(y) = \prod_{j=0, i \neq j}^k \frac{(y-y_j)}{(y_i-y_j)}$, polynômes d'interpolation de Lagrange de degrès $(k-1)$ associé à la séquence $\{y_0, \dots, y_k\}$ (3).

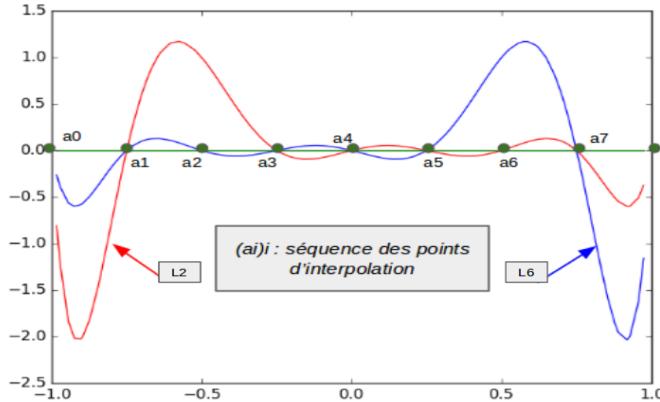


Figure 3: Exemples de polynômes d'interpolation de Lagrange

- **Cas $d > 1$:** On procède par tensorisation.
C'est à dire, pour $\alpha_1, \dots, \alpha_d \in \{0, \dots, k-1\}$

$$\begin{aligned} \tilde{f}(y_1, \dots, y_d) &= \bigotimes_{i=1}^d I_k(f)(y_1, \dots, y_d) \in \mathbb{Q}_{k-1} = \text{vect } \{y \rightarrow y_1^{\alpha_1} \dots y_d^{\alpha_d}\} \\ &= \sum_{k_1=1}^k \dots \sum_{k_d=1}^k (f(y_{k_1}, \dots, y_{k_d}) l_{k_1}(y_1) \dots l_{k_d}(y_d)) \end{aligned}$$

Pour avoir une bonne précision, il est préférable de choisir un nombre relativement grand de points d'interpolation. Si par exemple, on prend k_j points suivant la direction j , on obtient un nombre total de points égal à $\prod_{i=1}^d k_j$. Ceci pose problème, bien évidemment, lorsque d est grand. En effet, non seulement le calcul de \tilde{f} sera coûteux, mais aussi, si on souhaite ajouter 1 point suivant la variable i ça revient à ajouter $\prod_{j=1, j \neq i}^d k_j$ nœuds dans la grille (4).

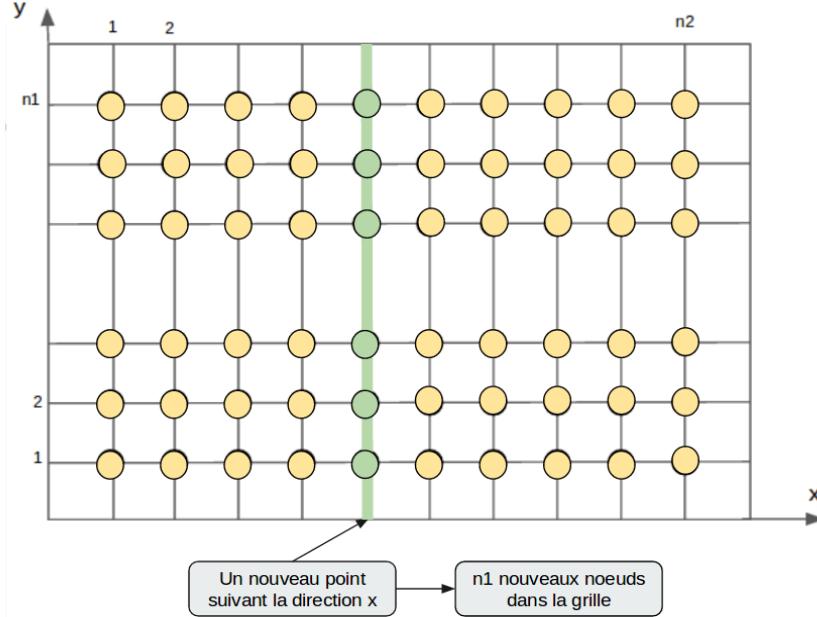


Figure 4: Influence de l'ajout d'un point d'interpolation sur une seule direction

Cette méthode d'interpolation devient très coûteuse lorsqu'il s'agit de fonctions multivariables en grande dimension.

Une solution serait de calculer l'opérateur d'interpolation par une méthode hiérarchique. C'est à dire qu'on va utiliser les approximations de f à l'ordre $j < k$ ($I_j(f)$ obtenue avec j points d'interpolation) pour approcher $I_k(f)$.

4.1.2 Construction hiérarchique de l'opérateur d'interpolation:

- Cas $d = 1$: Soit $k \geq 0$, on définit l'opérateur différence Δ_k par:

$$\Delta_k = I_k - I_{k-1}$$

On suppose, par convention, que I_{-1} est l'opérateur nul. De cette façon, $I_0 = \Delta_0$ correspond à l'opérateur qui à f associe le polynôme constant $f(y_0)$.

On a alors,

$$I_n = \sum_{k=0}^n \Delta_k$$

On définit ensuite, les polynômes hiérarchiques de degré k associés à la séquence $\{y_0, \dots, y_k\}$ par

$$h_k(y) = \prod_{j=0}^{k-1} \frac{(y - y_j)}{(y_k - y_j)}, \quad k > 0, \text{ et } h_0(y) = 1,$$

On a alors,

$$\Delta_k(f) = \alpha_k(f)h_k, \quad \alpha_k(f) = f(y_k) - I_{k-1}f(y_k)$$

On obtient alors, par un processus itératif, la représentation suivante:

$$I_n(f) = \sum_{k=0}^n \alpha_k(f)h_k$$

Ainsi pour calculer l'interpolant de f en utilisant n points, il suffit de connaître les interpolants de f d'ordre i , $i \in \{1, \dots, n-1\}$, et ceci d'une manière relativement rapide.

- **Cas $d > 1$:** Pour un multi-indice $\nu \in \mathcal{F}$, on associe le point multivarié

$$\mathbf{y}_\nu = (y_{\nu_j})_{j \geq 1} \in \mathcal{P},$$

la fonction polynomiale hiérarchique tensorisée

$$\mathbf{H}_\nu(y) = \prod_{j \geq 1} h_{\nu_j}(y_j),$$

et les opérateurs d'interpolation tensorisés

$$I_\nu := \bigotimes_{j \geq 1} I_{\nu_j} \text{ et } \Delta_\nu := \bigotimes_{j \geq 1} \Delta_{\nu_j},$$

Cette tensorisation peut être définie inductivement (voir [2]): pour une fonction f continue et bornée sur \mathcal{P} à valeurs réelles ou complexes,

- si $\nu = 0_{\mathcal{F}}$, alors $\Delta_\nu f = I_\nu f$ est le polynôme constant qui vaut $f(\mathbf{y}_{0_{\mathcal{F}}})$
- si $\nu \neq 0_{\mathcal{F}}$, alors

$$I_\nu f = I_{\nu_1}(t \mapsto I_{\hat{\nu}} f_t) \text{ et } \Delta_\nu f = \Delta_{\nu_1}(t \mapsto \Delta_{\hat{\nu}} f_t)$$

où $\hat{\nu} := (\nu_2, \nu_3, \dots)$ et $\forall t \in P$, la fonction f_t est définie sur $P^{\mathbb{N}}$ par $f_t(\hat{\mathbf{y}}) = g(\mathbf{y})$, $\mathbf{y} = (t, \hat{\mathbf{y}})$

Quand d est fini, la fonction f_t est définie de la même manière mais sur P^{d-1} . Dans ce cas, l'induction ci-dessus s'arrête après exactement d itérations et on obtient

$$I_\nu = \sum_{\mu \leq \nu} \Delta_\mu$$

Définition: Un ensemble $\Lambda \subset \mathcal{F}$, non vide, est dit monotone, si

$$\nu \in \Lambda \text{ et } \mu < \nu \Rightarrow \mu \in \Lambda$$

avec $\mu \leq \nu$ signifie que $\mu_j \leq \nu_j$ pour tout j .

Théorème: Pour tout ensemble monotone $\Lambda \subset \mathcal{F}$, on définit

$$I_\Lambda := \sum_{\nu \in \Lambda} \Delta_\nu, \quad \Gamma_\Lambda := \{\mathbf{y}_\nu : \nu \in \Lambda\}$$

La grille Γ_Λ est unisolvante pour \mathbb{P}_Λ et pour toute fonction f définie sur $\bar{\mathcal{P}}$, l'unique élément dans \mathbb{P}_Λ qui concorde avec f sur \mathbb{P}_Λ est donné par $I_\Lambda f$ (preuve dans [2]).

Dans notre cas, on veut effectuer une interpolation polynomiale pour une séquence imbriquée d'ensemble $(\Lambda_n)_{n \geq 1}$ avec $n = \sharp(\Lambda_n)$.

Pour que la méthode décrite précédemment en 1D fonctionne en dimension $d > 1$, il faut s'assurer que pour tout $n \geq 1$, Λ_n est monotone.

Dans cette configuration, Λ_n peut être vu comme une section $\{\nu^1, \dots, \nu^n\}$ d'une séquence $(\nu^k)_{k \geq 1} \in \Lambda^\mathbb{N}$. Cette observation mène à un algorithme efficace pour le calcul de $I_{\Lambda_n} f$ à partir de $I_{\Lambda_{n-1}} f$ et de la valeur de f au nouveau point \mathbf{y}_{ν^n} . En effet, on remarque que Δ_{ν^n} est multiple de la fonction hiérarchique tensorisée \mathbf{H}_{ν^n} défini précédemment.

Puisque $\mathbf{H}_{\nu^n}(\mathbf{y}_{\nu^n}) = 1$, alors

$$\begin{aligned} \Delta_{\nu^n} f &= \Delta_{\nu^n} f(\mathbf{y}_{\nu^n}) \mathbf{H}_{\nu^n} \\ &= (I_{\Lambda_n} f(\mathbf{y}_{\nu^n}) - I_{\Lambda_{n-1}} f(\mathbf{y}_{\nu^n})) \mathbf{H}_{\nu^n} \\ &= (f(\mathbf{y}_{\nu^n}) - I_{\Lambda_{n-1}} f(\mathbf{y}_{\nu^n})) \mathbf{H}_{\nu^n} \end{aligned}$$

Donc

$$I_{\Lambda_n} f = I_{\Lambda_{n-1}} f + (f(\mathbf{y}_{\nu^n}) - I_{\Lambda_{n-1}} f(\mathbf{y}_{\nu^n})) \mathbf{H}_{\nu^n}$$

Par conséquent, les polynômes $I_{\Lambda_n} f$ sont donnés par:

$$I_{\Lambda_n} f = \sum_{k=0}^n f_{\nu^k} \mathbf{H}_{\nu^k}$$

avec f_{ν^k} définis récursivement par:

$$f_{\nu^1} = f(y_0), \quad f_{\nu^{k+1}} = f(\mathbf{y}_{\nu^{k+1}}) - I_{\Lambda_k} f(\mathbf{y}_{\nu^{k+1}}) = f(\mathbf{y}_{\nu^{k+1}}) - \sum_{i=1}^k f_{\nu^i} \mathbf{H}_{\nu^i}(\mathbf{y}_{\nu^{k+1}})$$

4.1.3 Interpolation adaptative et séquence de points d'interpolation:

Les ensembles Λ_n peuvent être choisis préalablement de sorte qu'ils forment une séquence imbriquée d'ensembles monotones. Ils peuvent aussi être construits d'une manière astucieuse lors du calcul de l'interpolant en construisant un chemin qui correspond à un ordre entre les points d'interpolations.

Soit l'analogie suivante: si $(\mathbf{H}_\nu)_{\nu \in \mathcal{F}}$ est une base orthoromée de $L^2(\mathcal{P})$ alors le choix d'un ensemble d'indices Λ_n qui minimise l'erreur serait de prendre les n plus grands $a_\nu |f_\nu|$ tel que $a_\nu = \|\mathbf{H}_\nu\|_{L^\infty(\mathcal{P})} = \prod_{j \geq 1} \|h_{\nu_j}\|_{L^\infty(\mathcal{P})}$.

Cette stratégie permet d'avoir une séquence imbriquée $(\Lambda_n)_{n \geq 1}$, cependant, il n'est pas certain que les ensembles Λ_n soient monotones.

Afin de résoudre ce problème, on définit la notion de voisins pour n'importe quel ensemble monotone Λ par,

$$\mathcal{N}(\Lambda) = \{\nu \notin \Lambda : R_\nu \in \Lambda \cup \{\nu\}\}, R_\nu = \{\mu \in \Lambda : \mu < \nu\}$$

Ainsi, avec cette définition, l'algorithme ci-dessous mène à une séquence imbriquée d'ensembles monotones.

Algorithme d'Interpolation Adaptative:

- Commencer par $\Lambda_1 = \{0_\Lambda\}$
- En supposant Λ_{n-1} calculé, trouver $\nu^n = \operatorname{argmax} \{a_\nu |f_\nu| : \nu \in \mathcal{N}(\Lambda_{n-1})\}$, et définir $\Lambda_n = \Lambda_{n-1} \cup \{\nu\}$

Il est important de noter que le choix de la séquence de points d'interpolation joue un rôle critique pour la stabilité de l'opérateur d'interpolation multivariée définie par la constante de Lebesgue:

$$\mathbb{L}_\Lambda = \sup_{f \in B(\mathcal{P})-0} \frac{\|I_\Lambda f\|_{L^\infty(\mathcal{P})}}{\|f\|_{L^\infty(\mathcal{P})}}$$

avec $B(\mathcal{P})$ est l'espace des fonctions bornées définies sur \mathcal{P}

L'objectif est de choisir la séquence $(y_k)_{k \geq 0}$ tel que les constantes mono-variées de Lebesgue

$$\lambda_k = \max_{f \in B(\mathcal{P})-0} \frac{\|I_k f\|_{L^\infty(\mathcal{P})}}{\|f\|_{L^\infty(\mathcal{P})}}$$

associées à l'opérateur d'interpolation mono-variée I_k croissent modérément par rapport à k . Une telle séquence peut être construite en fixant $y_0 \in P$ et en définissant inductivement

$$y_k = \operatorname{argmax}_{y \in P} \prod_{j=0}^{k-1} |y - y_j|$$

Cette séquence $(y_k)_{k \geq 0}$ est appelée séquence de Leja [1] sur P . Elle modère la croissance des constantes de Lebesgue et a une implication intéressante sur le choix adaptatif des Λ_n . Voici les 10 premiers points de la séquence de Leja construite dans l'intervalle $[-1, 1]$:

$$\{ 1, -1, 0, -0.577316, 0.658716, -0.83924, 0.870029, 0.305729, -0.321539, -0.942991 \}$$

La précision de l'interpolation dépend fortement du choix des fonctions hiérarchiques de base et de la séquence des points d'interpolation. Quand il s'agit d'approcher des fonctions plates ou polynomiales de n'importe quel ordre, il est intéressant de choisir les polynômes hiérarchiques de Lagrange ainsi que la séquence de Leja.

En effet, non seulement, on obtient l'interpolant au bout d'un nombre d'itérations relativement petit, mais aussi avec une grande précision.

Cependant, quand il s'agit d'approcher des fonctions plus complexes, par exemple des fonctions présentant de fortes variations brusques, ou certaines irrégularités, le choix précédent n'est plus optimal.

On a vu que l'algorithme AI évolue en localisant les points où l'erreur d'interpolation est la plus élevée. Parfois, il est très difficile de détecter un point où l'erreur est grande. Pour mieux voir cela, considérons l'exemple de la figure ci-dessous (5).

Cette figure est divisée en deux parties. Chaque partie correspond à une itération particulière dans l'algorithme AI.

Chaque partie est divisée verticalement en deux graphes:

- Celui en haut montre l'interpolé (en vert) et l'interpolant (en rouge).
- Celui en bas montre les fonctions de bases construites jusqu'à l'itération courante.

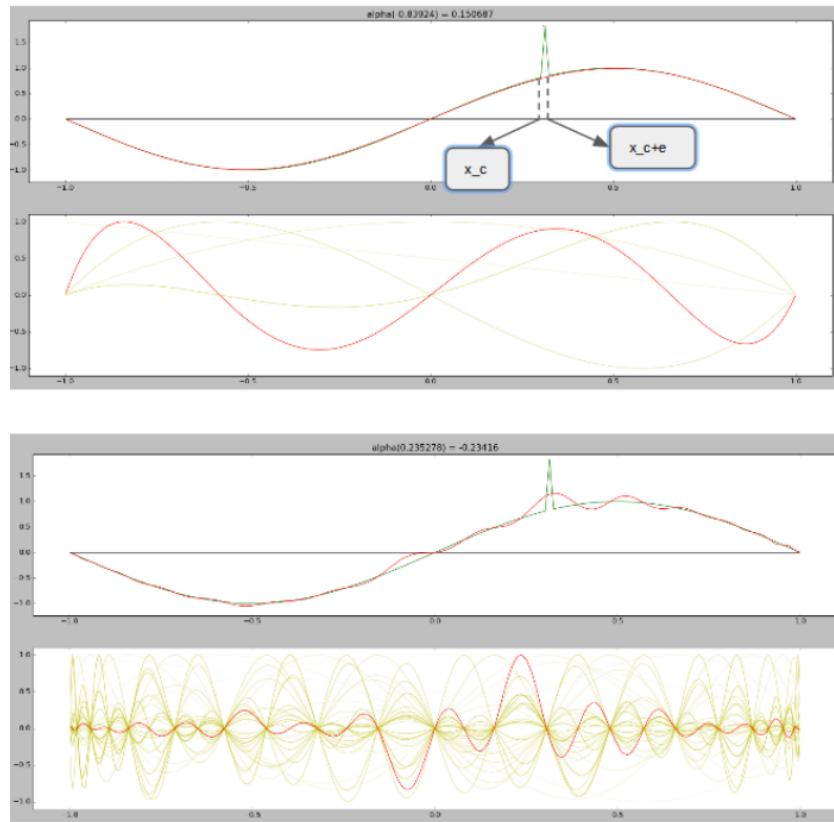


Figure 5: Interpolation utilisant des polynômes de Lagrange définis globalement

Dès qu'un nouveau point d'interpolation est ajouté dans l'intervalle $[x_c, x_c + \epsilon]$, on obtient des perturbations de l'interpolant ailleurs. Ce qui provoque une augmentation considérable de l'erreur. Ceci est dû au caractère global des fonctions hiérarchiques.

Dans de telles situations il est plus judicieux d'utiliser des fonctions polynômiales par morceaux comme fonctions hiérarchiques de base.

De cette façon, chaque nouveau point d'interpolation ajouté, ne provoque que des perturbations locales, ce qui ne dégrade pas trop l'interpolation obtenu jusque-là.

Dans ce cas de figure, on choisit un nouveau type de points d'interpolation qui est mieux adapté. La séquence sera formée par un ensemble de points obtenu en divisant progressivement le domaine par dichotomie.

Voici, un exemple en 1D pour mieux comprendre cela. Supposons que $P = [-1, 1]$ et que le premier point d'interpolation est 0. Alors, la séquence de points d'interpolation évoluera de la façon suivante.

On suppose que S est la séquence de points d'interpolation, et V la liste courante des points voisins de S dans l'algorithme d'interpolation adaptative.

- **Iteration 0:** $S = \{0\}$ et $V = \{-1, 1\}$
- **Iteration 1:** ($S = \{0, -1\}$ et $V = \{1, -0.5\}$) ou
($S = \{0, 1\}$ et $V = \{-1, 0.5\}$)
- **Iteration 2:** ($S = \{0, -1, 1\}$ et $V = \{-0.5, 0.5\}$) ou
($S = \{0, -1, -0.5\}$ et $V = \{1, -0.25, -0.75\}$) ou
($S = \{0, 1, -1\}$ et $V = \{-0.5, 0.5\}$) ou
($S = \{0, 1, 0.5\}$ et $V = \{-1, 0.25, 0.75\}$)

En d'autres termes, le nouveau point d'interpolation sera celui, parmi les voisins de S (courant), au niveau duquel l'erreur d'interpolation est maximale. Ce point appartient forcément à l'ensemble des noeuds descendant directement des points de S dans l'arbre ci-dessous (6).

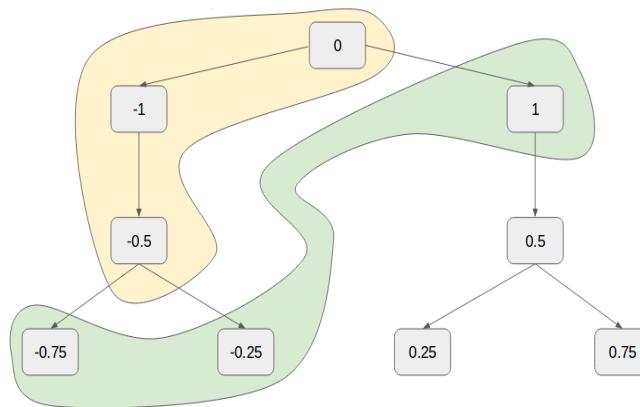


Figure 6: Arbre représentatif des points d'interpolation en 1D

Dans la figure précédente, l'ensemble jaune correspond à la séquence d'interpolation, et celui en vert correspond à l'ensemble courant des points voisins.

Étant donné qu'on a besoin d'une notion d'ordre entre les points d'interpolation, on introduit la relation suivante: un nœud n est d'ordre plus élevé qu'un nœud m si et seulement

si m est un ancêtre de n .

Ainsi, en grande dimension, et avec ce type de séquence, on est capable de définir un ensemble monotone vu qu'on a une relation d'ordre entre les noeuds.

Exemple: soit la séquence monotone de points 3D suivante:

$S = \{(0, 0, 0); (0, 1, 0); (1, 0, 0); (0.5, 0, 0); (0.5, 1, 0)\}$. Le point $p = (0.25, 0, 0)$ est un voisin de cette séquence, car tous les points d'ordre inférieur à celui de p (ancêtres de p) appartiennent à cette séquence. En effet, l'ensemble de tous points d'ordre inférieur à celui de p est $E = \{(0, 0, 0); (1, 0, 0); (0.5, 0, 0)\}$. E est un sous-ensemble de S donc p est bien un voisin de S . Ainsi, $\{p\} \cup S$ reste un ensemble monotone.

La figure ci-dessous (7), montre le résultat de l'interpolation d'une fonction de même type que précédemment, cette fois, en utilisant des fonctions quadratiques par morceaux comme fonctions de base.

On remarque que dès que l'algorithme détecte un point dans la zone critique, il corrige l'interpolant courant sans pour autant provoquer des erreurs ailleurs.

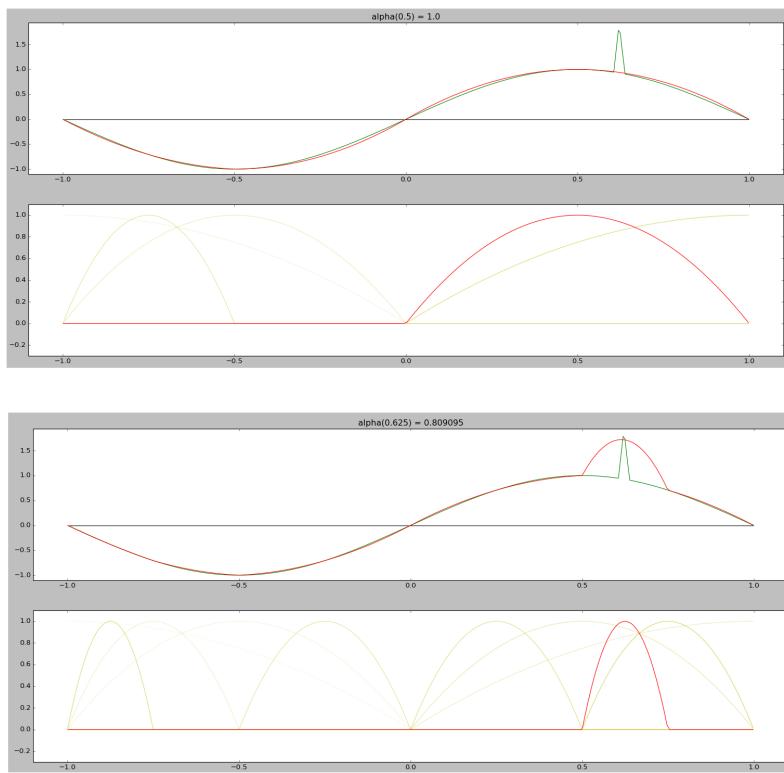


Figure 7: Interpolation utilisant des fonctions quadratiques par morceaux

Remarque:

- Certains noeuds ne peuvent pas être comparés (exemple -1 et 0.5) car l'un n'est ni ancêtre, ni descendant de l'autre.
- En 1D, et pour une itération ($i > 0$) le nombre de voisins (éventuels futurs points d'interpolation) obtenu avec la méthode d'interpolation par polynômes de Lagrange est inférieur à celui obtenu en utilisant des fonctions quadratique par morceaux.

En effet, avec la première méthode, on a toujours un seul voisin possible (qui correspond au point suivant dans la séquence de Leja). Par contre, avec l'autre méthode, on en a plus, car il y a toujours 2 voisins pour chaque point (qui correspondent aux nœuds filles dans l'arbre) sauf pour les extrémités (-1 et 1).

- La méthode d'interpolation par fonctions quadratiques (ou affines) par morceaux n'est pas toujours meilleure que celle par polynômes de Lagrange. Le cas le plus simple est quand l'interpolé est un polynôme. Dans ce cas, on obtient une bonne approximation avec les deux méthodes. Sauf qu'avec la méthode utilisant les polynômes de Lagrange, l'algorithme est beaucoup plus précis et rapide (car il y a toujours moins de voisins à évaluer).
- En dimension quelconque on peut avoir des fonctions ayant une nature différente suivant chaque variable. Dans ce cas, il est possible de combiner les deux méthodes en en choisissant une sur chaque direction.

4.2 Implémentation de la solution

Dans cette partie, on va mettre en pratique cette méthode et on va présenter certains détails de l'implémentation de cette solution (notamment la construction de la séquence des points d'interpolation et le calcul des f_ν).

Cette solution a été implémentée en C++. Le code est constitué principalement des classes suivantes:

- **Classe Interpolation<T>:** Il s'agit d'une classe générique abstraite. Elle implémente les structures de données contenant les différentes quantités qui entrent en jeu dans la construction de l'opérateur d'interpolation, notamment les points d'interpolations, les points de test et la liste des voisins courants dans l'algorithme AI... De plus, l'algorithme AI ainsi que certaines fonctions intermédiaires, (notamment celle responsable de la recherche des voisins dans une grille) sont implémentés dans cette classe. Ici, T fait référence au type de données utilisé pour représenter l'ordre des points d'interpolation. Vu qu'il est différent pour les deux méthodes, on a choisi de procéder par généricité.

Cependant, il y a plusieurs similarités entre les deux méthodes, d'où le choix de faire une classe abstraite, ensuite de faire deux classes filles qui héritent des fonctionnalités communes (création des points de test, calcul des erreurs relatives, construction de l'opérateur d'interpolation une fois tous les f_ν calculés ...).

- **Classe BinaryTree:** Cette classe implémente un arbre binaire qui permet de coder les points cartésiens et les ordonner. Cette classe est utile dans la version de l'algorithme AI qui utilise des fonctions de bases quadratiques par morceaux. En effet, on ne peut pas ordonner les points d'interpolation par des indices comme dans l'autre méthode (chaque indice correspond à l'ordre du réel dans la séquence de Leja).

Cependant, on a besoin de fonctionnalités qui nous permettent de situer un point d'interpolation parmi d'autres ou de retrouver les deux points les plus proches pour pouvoir construire la fonction de base (quadratique par morceaux) correspondante. L'idée ici est de coder chaque point, non par un indice mais par une chaîne de caractère qui n'est autre que le code de Huffman correspondant dans l'arbre. Ainsi, on

est capable de définir une méthode pour comparer l'ordre des points. Chaque arbre est construit de la façon suivante:

- Le point 0.0 est la racine.
- Chaque nœud possède au maximum 2 fils.
- Les points -1.0 et 1.0 (extrémités de P) possèdent chacun exactement un fils.
- Soit n un nœud intermédiaire, on note p son père, g son fils gauche, et d son fils droit, on a alors:

$$g = \begin{cases} -1 & \text{si } n = 0 \\ n - |p - n| / 2 & \text{si } n \neq 0 \end{cases}$$

$$d = \begin{cases} 1 & \text{si } n = 0 \\ n + |p - n| / 2 & \text{si } n \neq 0 \end{cases}$$

Le code de chaque noeud n est construit en parcourant l'arbre de la racine jusqu'au point n . Le code est initialement une chaîne vide. Chaque fois qu'on va à gauche (resp. à droite) ou écrit 0 (resp. 1).

Ainsi, le code de -0.375 dans l'arbre ci-dessous (8) est "0110".

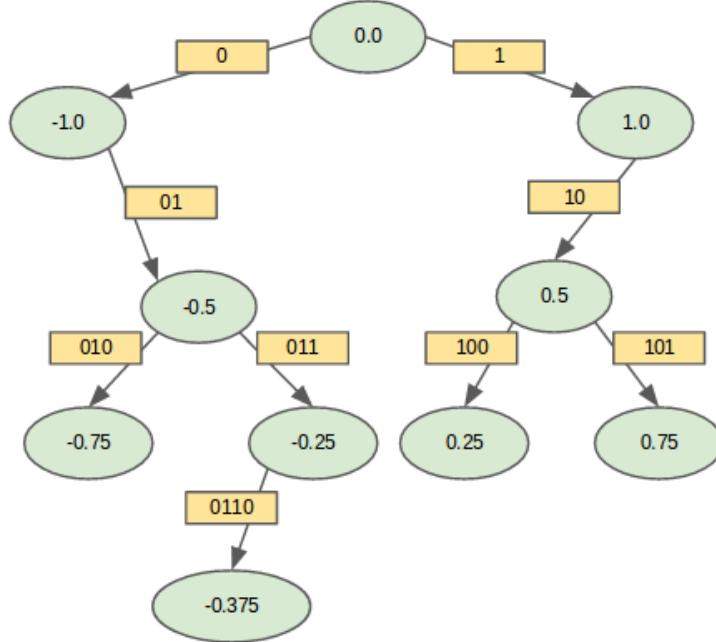


Figure 8: Codage de huffman de chaque nœud de l'arbre

Avec ce codage, on est capable de retrouver facilement, à partir d'un nœud n le code de son parent ou encore les codes des noeuds ayant les valeurs les plus proches.

- **Classe MultivariatePoint:** Il s'agit d'une classe générique qui modélise à la fois les multi-indices, ainsi que les points réels multivariés. C'est à dire qu'une instance de cette classe peut renfermer soit des valeurs réelles, soit des entiers, soit des chaînes de caractères qui correspondent aux codes de Huffman du point réel en question. Les opérateurs d'affectation, d'addition, d'affichage ainsi que de comparaison de points multivariés sont implémentés dans cette classe. Un multi-indice correspond à un ensemble d'ordres. Par exemple: en 2D, le vecteur $(0, 1) \in \mathbb{N}^2$ correspond au point $(1.0, -1.0) \in \mathcal{P}$.

En effet, 1.0 (resp. -1.0) étant le point d'indice 0 (resp. 1) dans la séquence de Leja.

Par contre, quand il s'agit d'utiliser une séquence de points construits par dichotomie, on utilisera des chaînes de caractères. Par exemple : en 2D, le vecteur $("", "01")$ correspond au point $(0, -0.5) \in \mathcal{P}$.

- **Classe LagrangeInterpolation:** Cette classe hérite de **Interpolation<int>** et elle implémente la méthode utilisant les polynômes de Lagrange et les points de Leja.
- **Classe PiecewiseInterpolation:** Cette classe hérite de **Interpolation<string>** et elle implémente la méthode utilisant les fonctions quadratiques par morceaux.
- **Classe MixedInterpolation:** Cette classe hérite de **Interpolation<string>** et elle implémente une combinaison des 2 méthodes. Ici, on utilise des chaînes de caractère car, avec ce type de donnée, on peut à la fois représenter les codes de Huffman ainsi que de simples entiers.
- **Classe Utils:** Il s'agit d'une classe statique renfermant certaines fonctions utiles notamment les fonctions d'affichages, les fonctions qui gèrent la création des séquences d'interpolations (notamment la séquence uniforme, les zéros de Tchebychev ainsi que la séquence de Leja). De plus, cette classe permet de lire des données et d'écrire les résultats dans des fichiers externes.
- **Classe Functions:** Cette classe implémente certaines fonctions analytiques qui permettront de tester l'algorithme d'Interpolation Adaptative. De plus, elle permet de lire, d'organiser et d'intégrer les données réelles fournies par le département R&D d'EDF.

Construction de la séquence de Leja:

L'idée est de discréteriser l'intervalle de définition U de la fonction f en un grand nombre de points $(a_i)_{i=1..n(=100000)}$. Supposons qu'on a construit à l'étape i la séquence $\{y_0, \dots, y_i\}$, alors pour avoir le point y_{i+1} , on compare le produit des distances de chaque point de discréterisation aux points de Leja déjà calculés, puis on choisit le max.

La façon la plus naturelle de choisir les points a_i est de les choisir uniformément répartis dans l'intervalle de définition de f .

On obtient alors une suite de polynômes qui interpole f en de plus en plus de points. On pourrait s'attendre à ce que la suite converge uniformément vers f lorsque le nombre de points d'interpolation augmente.

Malheureusement, ce n'est pas le cas, ce phénomène est connu sous le nom de phénomène de Runge.

Une solution est, étonnamment, de ne pas choisir les points uniformément répartis. Une raison pour cela est l'inégalité suivante : si f est de classe C^N sur l'intervalle U et si L est le polynôme d'interpolation de Lagrange aux points a_1, \dots, a_N , alors on a

$$\forall x \in U, |f(x) - L(x)| \leq \sup_{x \in U} \left(\prod_{i=1}^n |x - a_i| \right) \frac{\|f^{(n)}\|_\infty}{n!}$$

L'idée est donc de choisir les a_i de sorte que $\sup_{x \in U} \prod_{i=1}^n |x - a_i|$ soit le plus petit possible. On démontre que ceci est réalisé lorsque les a_i sont les zéros du polynôme de Tchebychev de degré n , à savoir $a_i = \cos(\frac{i\pi}{n})$, $i = 0, \dots, n$.

La figure 6 montre une grille 2D formée par des points de Leja construits dans le rectangle $[-1, 1] \times [-1, 1]$.

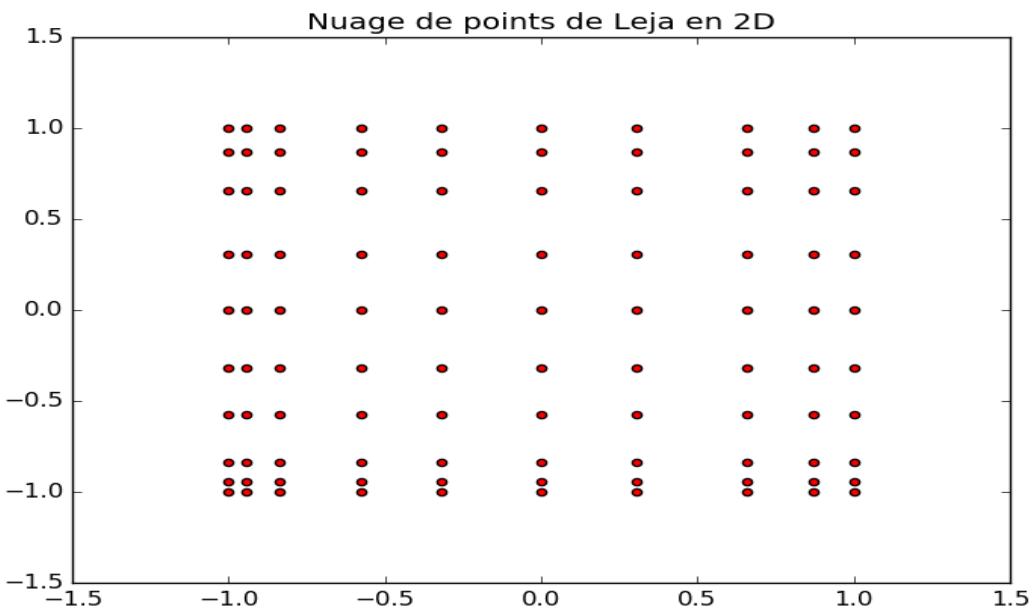


Figure 9: Grille 2D des points de Leja

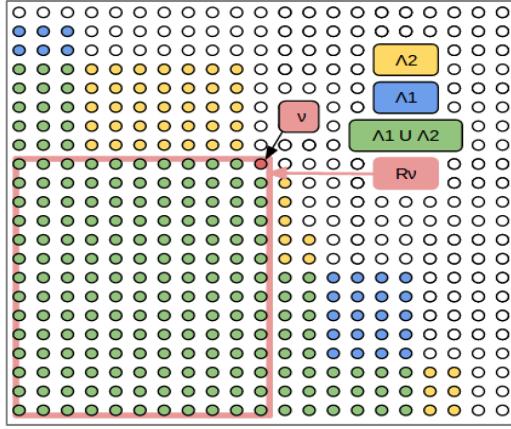
Calcul des f_ν (ou α_ν):

A chaque étape de l'algorithme AI, on construit un sous-ensemble de points qui correspondent à tous les voisins courants. Ensuite, on calcule tous les f_ν (ν étant les vecteurs d'ordre correspondants aux voisins courants) et on choisit le voisin ayant le plus grand f_ν . Il est possible qu'au cours d'une certaine itération (voir toutes les itérations) de recalculer un f_ν déjà calculé aux itérations précédentes. Ceci est, bien évidemment, coûteux.

Il est possible de procéder d'une manière plus astucieuse. En effet, étant donné que f_ν est constant et ne dépend que de ν (justification ci-dessous), on n'a besoin de le calculer qu'une seule fois.

Preuve: Dans la figure ci-dessous (4.2), on considère deux ensemble différents Λ_1 (bleu + vert) et Λ_2 (jaune + vert).

Ces deux ensembles sont choisis de sorte que leur intersection forme l'ensemble (vert) E qui n'est autre que $R_\nu - \{\nu\}$.



L'erreur d'interpolation au point d'ordre ν peut être exprimée de plusieurs manières en fonction de l'ensemble de points d'interpolation Λ courant:

$$\begin{aligned} f_\nu &= f(\mathbf{y}_\nu) - I_{\Lambda_1} f(\mathbf{y}_\nu) \\ f_\nu &= f(\mathbf{y}_\nu) - I_{\Lambda_2} f(\mathbf{y}_\nu) \end{aligned}$$

Montrons que $I_\Lambda f(\mathbf{y}_\nu)$ ne dépend que de ν (càd $I_{\Lambda_1} f(\mathbf{y}_\nu) = I_{\Lambda_2} f(\mathbf{y}_\nu)$). De manière générale,

$$\begin{aligned} I_\Lambda f(\mathbf{y}_\nu) &= \sum_{\tilde{\nu} \in \Lambda} \Delta_{\tilde{\nu}} f \\ \Delta_{\tilde{\nu}} &= \bigotimes_{i=1}^d \Delta_{\tilde{\nu}_i} \end{aligned}$$

or $\Lambda_i, i \in \{1, 2\}$, peut s'écrire de cette façon:

$$\Lambda_i = (R_\nu - \{\nu\}) \cup (\Lambda_i - R_\nu), \text{ car } \nu \notin \Lambda_i$$

Ainsi

$$I_{\Lambda_i} = I_{R_\nu} - \Delta_\nu + \sum_{\tilde{\nu} \in \Lambda_i - R_\nu} \Delta_{\tilde{\nu}}$$

Or $\Delta_{\tilde{\nu}} f(\mathbf{y}_\nu) = 0$ si $\tilde{\nu}$ est non inférieur à ν

$$\begin{aligned} \tilde{\nu} \text{ est non inférieur } \nu &\Leftrightarrow \tilde{\nu} \notin R_\nu \\ &\Leftrightarrow \exists j < d \text{ tel que } \tilde{\nu}_j > \nu_j \\ &\Leftrightarrow \exists j < d \text{ tel que } \Delta_{\tilde{\nu}_j} f(\mathbf{y}_{\nu_j}) = (I_{\tilde{\nu}_j} - I_{\tilde{\nu}_j-1}) f(\mathbf{y}_{\nu_j}) = 0 \\ &\Rightarrow \Delta_{\tilde{\nu}} f(\mathbf{y}_\nu) = 0 \end{aligned}$$

Donc, on en déduit que

$$f_\nu = f(\mathbf{y}_\nu) - I_{R_\nu} f(\mathbf{y}_\nu) + \Delta_\nu f(\mathbf{y}_\nu)$$

Ainsi, f_ν ne dépend que de ν .

5 Évaluation de l'efficacité de la méthode

L'algorithme d'Interpolation Adaptative a été testé sur des fonctions analytiques définies à l'avance, en une grille de référence construite d'une manière aléatoire.

Les résultats obtenus ont été ensuite comparés à ceux obtenus avec une autre méthode d'approximation de fonctions multivariées basée sur le modèle de décomposition de Tucker.

L'étape suivante a été de tester puis de comparer les deux méthodes ainsi que la méthode d'interpolation multilinéaire sur des données réelles fournies par le département R&D d'EDF.

5.1 Présentation du modèle de décomposition de Tucker

D'un point de vue mathématique, le problème de reconstruction des sections efficaces se présente comme l'approximation de plusieurs fonctions multivariées $\{f_k(\mathbf{x})\}_k$, définies sur un domaine Ω , ici $\mathbf{x} = (x_1, \dots, x_d) \in \Omega \subset \mathbb{R}^d$, d est le nombre de paramètres et $\Omega = \Omega_1 \times \dots \times \Omega_d$.

L'objectif est d'acquérir des informations, sauvegarder les données, et proposer une reconstruction de chaque f_k avec un rapport (précision / complexité) élevé.

Décomposition de Karhunen-Loève:

L'intérêt de cette décomposition est qu'elle fournit une approximation optimale de rang r de fonctions en 2D par rapport à la norme L^2 .

Soit $f = f(x, y)$, une fonction de $L^2(\Omega_x \times \Omega_y)$. Par la décomposition de Karhunen-Loève, on peut écrire f comme:

$$f(x, y) = \sum_{n=1}^{+\infty} \sqrt{\lambda_n} \varphi_n(x) \psi_n(y), \quad \forall (x, y) \in \Omega_x \times \Omega_y$$

Avec $\{\varphi(x)\}_{n=1}^{+\infty}$ (*resp.* $\{\psi(y)\}_{n=1}^{+\infty}$) est la base orthonormée de $L^2(\Omega_x)$ (*resp.* la base orthonormée de $L^2(\Omega_y)$). En outre, chaque $(\lambda_n, \varphi_n(x))$ (*resp.* $(\lambda_n, \psi_n(y))$) est un couple de valeur propre - fonction propre de l'opérateur de Hilbert-Schmidt $K_f^{(x)}$ (*resp.* $K_f^{(y)}$) défini par:

$$\begin{aligned} K_f^{(x)} : L^2(\Omega_x) &\rightarrow L^2(\Omega_x) \\ u &\mapsto K_f^{(x)} u(x) = \int_{\Omega_x} \int_{\Omega_y} f(x, y) f(x', y) dy u(x') dx' \\ (\text{resp. } K_f^{(y)} : L^2(\Omega_y) &\rightarrow L^2(\Omega_y) \\ u &\mapsto K_f^{(y)} u(y) = \int_{\Omega_y} \int_{\Omega_x} f(x, y) f(x, y') dx u(y') dy' \end{aligned}$$

Les $\{\lambda_n\}_{n=1}^{+\infty}$ sont toutes positives et triées dans un ordre décroissant: $\lambda_1 \geq \lambda_2 \geq \dots > 0$.

Pour un entier r donné ($r \in \mathbb{N}^*$), si on cherche une approximation de f par un produit tensoriel de rang r , alors la décomposition de Karhunen-Loève fournit la meilleure approximation au sens de la minimisation de l'erreur quadratique moyenne e^{RMSE} .

$$f_r^{best} = \sum_{n=1}^r \sqrt{\lambda_n} \varphi_n(x) \psi_n(y)$$

Dans ce cas:

$$e^{RMSE} = \sqrt{\|f(x, y) - f_r^{best}(x, y)\|_{L^2(\Omega)}^2} = \sqrt{\sum_{n=r+1}^{\infty} \lambda_n}$$

La décomposition de Karhunen-Loève pour une fonction en 2D montre que l'information la plus importante qui permet de représenter une fonction bi-variée peut être trouvée dans les premières fonctions propres (associées aux premières plus grandes valeurs propres). Cette décomposition est valable en 2 dimension, mais n'a pas d'implications directes en dimension plus grande.

Décomposition de Tucker comme éxtension de la décomposition de Karhunen-Loève:

Le modèle de Tucker propose de considérer les fonctions multivariées comme des fonctions bivariées de x et \mathbf{y} puis d'appliquer la décomposition de Karhunen-Loève à cette expression : la première direction correspond à une variable présente dans la configuration multidimensionnelle, i.e. $x = x_j$ pour $j = 1, \dots, d$, l'autre direction correspond au reste des variables condensés dans un vecteur $\mathbf{y} := (x_1, \dots, x_{j-1}, x_{j+1}, \dots, x_d)$. Cette approche, appelée matricialisation, effectuée dans chaque direction, permet de construire des fonctions de base tensorielles directionnelles appropriées.

La décomposition de Karhunen-Loève est utilisée itérativement pour tout i et donne d équations, chacune s'écrit de la façon suivante:

$$\int_{\Omega_x} \int_{\Omega_y} f(x, \mathbf{y}) f(x', \mathbf{y}) dx' d\mathbf{y} = \lambda \varphi(x), \quad \forall x \in \Omega_x$$

Résoudre ces équations permet de déterminer les fonctions de base tensorielles directionnelles $\{\varphi^{(j)}(x_j)\}$ pour tout j ($1 \leq j \leq d$). De cette décomposition, on ne garde que la famille de vecteurs propres dans la variable x_j . Le processus d'extension est illustré dans la figure 11.

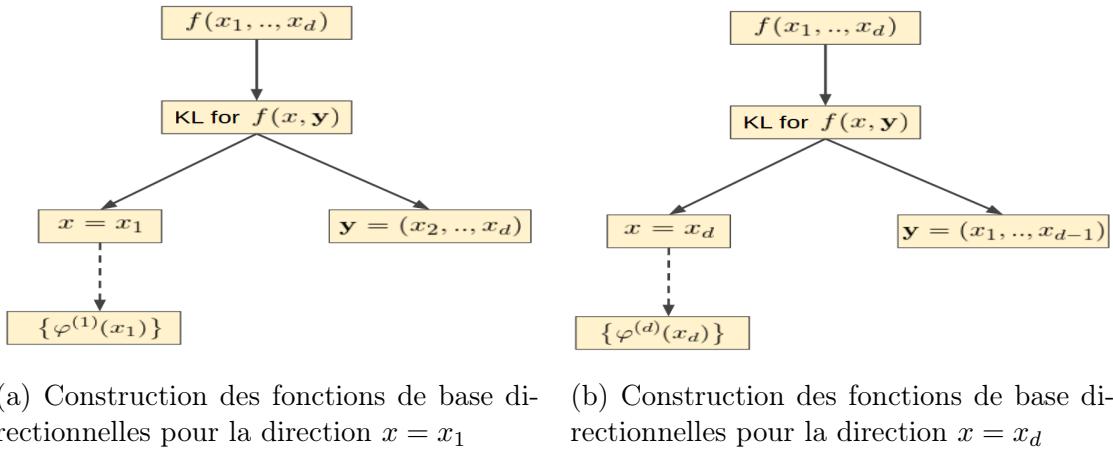


Figure 11: Construction des fonctions de base directionnelles pour toutes les directions en utilisant une extension du modèle de décomposition de Karhunen-Loève (KL)

Afin de résoudre numériquement les équations décrites plus haut, il faut une méthode pour approximer les intégrales. En pratique, pour une fonction d -variée $f = f(x_1, \dots, x_d)$, on a besoin de d formules de quadratures différentes basées sur d grilles. Chacune est utilisée pour construire les fonctions de base tensorielles directionnelles pour seulement une direction j . Une bonne méthode est de discréteriser d'une manière adaptative chaque domaine afin de capturer les informations importantes associées à la direction concernée, mais en utilisant le moins de points que possible. Concrètement, la direction concernée j est finement discréterisée, alors que les autres sont discréterisés grossièrement (2^{d-1} points). Une telle discréterisation est appelée **grille de Tucker**.

Étant donné qu'on a d grilles dans un espace de dimension d , alors dans chaque direction, d discréterisations différentes sont réalisées, mais seulement une est fine. Ces discréterisations sont présentées dans la figure 12.

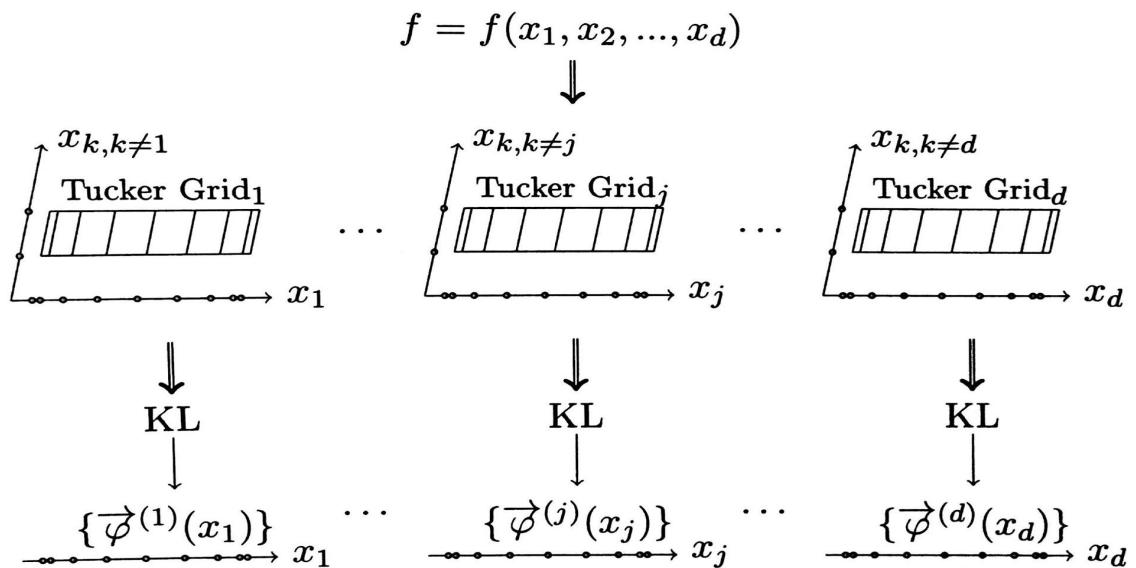


Figure 12: Discréterisation adaptative (grilles de Tucker) utilisées pour une extension de la décomposition de Karhunen-Loève afin de construire les fonctions de base tensorielles directionnelles direction par direction

Une fois le système d'équations, décrit plus haut, est résolu numériquement, on obtient une famille de vecteurs propres $\{\vec{\varphi}^{(j)}(x_j)\}$. Cependant, le but est de proposer une approximation de la fonction écrite comme une décomposition de Tucker

$$f(x_1, \dots, x_d) \approx \tilde{f}(x_1, \dots, x_d) = \sum_{i_1=1}^{r_1} \dots \sum_{i_d=1}^{r_d} a_{i_1 \dots i_d} \prod_{j=1}^d \varphi_{i_j}^{(j)}(x_j) \quad (1)$$

qui prend en compte des fonctions propres dans sa partie droite. Donc il faut une méthode qui transforme les vecteurs propres en fonctions propres, par exemple, le modèle d'interpolation de Lagrange. C'est-à-dire que, pour une direction j , les fonctions propres φ^j sont représentées par des polynômes de Lagrange via les points de cette direction et les valeurs des vecteurs propres en ces points. Les points mentionnés ici sont ceux de la discréterisation fine de la direction concernée j .

Critère de sélection des fonctions de base tensorielles directionnelles dans la décomposition de Tucker:

L'idée est de sélectionner les fonctions de base les plus importantes. Cette sélection prend seulement les fonctions propres associées aux valeurs propres λ_i qui satisfont $\frac{\lambda_i}{\lambda_1} > \epsilon$, avec $\lambda_1 \geq \lambda_2 \geq \dots > 0$ et ϵ , un paramètre choisi arbitrairement (ex 10^{10}).

Détermination des coéficients dans la décomposition de Tucker:

Dans l'application de la décomposition de Tucker (voir équation 1) pour $f = f(x_1, \dots, x_d)$, il faut déterminer $R = \prod_{j=1}^d r_j$ coéficients \mathbf{a}_i , où r_j est le nombre de fonctions de base tensorielles directionnelles dans la direction j : $r_j = \#\left\{\varphi_{i_j}^{(j)}\right\}$.

Il est possible de trouver les meilleurs $\mathbf{a}_i = a_{i_1 \dots i_d} = \langle f, \prod_{j=1}^d \varphi_{i_j}^{(j)} \rangle$. Cependant, cette méthode nécessite un coût de calcul élevé. Une approche plus simple est basée sur l'interpolation en R points bien sélectionnés $\{\mathbf{x}_t\}_{t=1}^R$.

Si ces points sont donnés avec leurs coordonnées $\mathbf{x}_t = (x_{t_1}, \dots, x_{t_d}, \dots, x_{t_d})$, alors les R coéficients $\{a_1, \dots, a_i, \dots, a_R\}$ sont solution du système suivant:

$$\begin{cases} f(\mathbf{x}_1) = \sum_{i=1}^R \mathbf{a}_i \prod_{j=1}^d \varphi_{i_j}^{(j)}(x_{1_j}) \\ \dots \\ f(\mathbf{x}_t) = \sum_{i=1}^R \mathbf{a}_i \prod_{j=1}^d \varphi_{i_j}^{(j)}(x_{t_j}) \\ \dots \\ f(\mathbf{x}_R) = \sum_{i=1}^R \mathbf{a}_i \prod_{j=1}^d \varphi_{i_j}^{(j)}(x_{R_j}) \end{cases} \quad (2)$$

Les points $\{\mathbf{x}_t\}_{t=1}^R$ sont appelés des points d'évaluation de f . Les coordonnées $\{x_{t_j}\}_{t=1}^R$ pour une direction j donnée sont appelés les points d'évaluation des fonctions de base tensorielles directionnelles sur $\left\{\varphi_{i_j}^{(j)}\right\}_{i_j=1}^{r_j}$ la direction j .

5.2 Résultats des approximations de fonctions analytiques

Les tests de comparaisons ont été faits sur quatre fonctions différentes définies sur le carré $[-1, 1]^d$, d étant la dimension de l'espace de départ. Les fonctions prises en compte sont les suivantes:

- $f1$: Polynôme de degré 7 dont les coéficients sont choisis de manière aléatoire.
- $f2$: $x = (x_1, \dots, x_d) \rightarrow \sqrt{1 - x_0^2} \exp(\sum_{i=1}^d x_i)$
- $f3$: $x = (x_1, \dots, x_d) \rightarrow \sin(\|x\|_2)$
- $f4$: $x = (x_1, \dots, x_d) \rightarrow \sin(\sum_{i=1}^d x_i)$

Les figures ci-dessous (14, 16) montre l'évolution de la précision ainsi que le temps de calcul de chaque méthode en fonction du nombre de points d'évaluation.

La courbe verte correspond aux résultats obtenus avec la méthode de Tucker et la courbe rouge correspond à ceux obtenus avec la méthode AI.

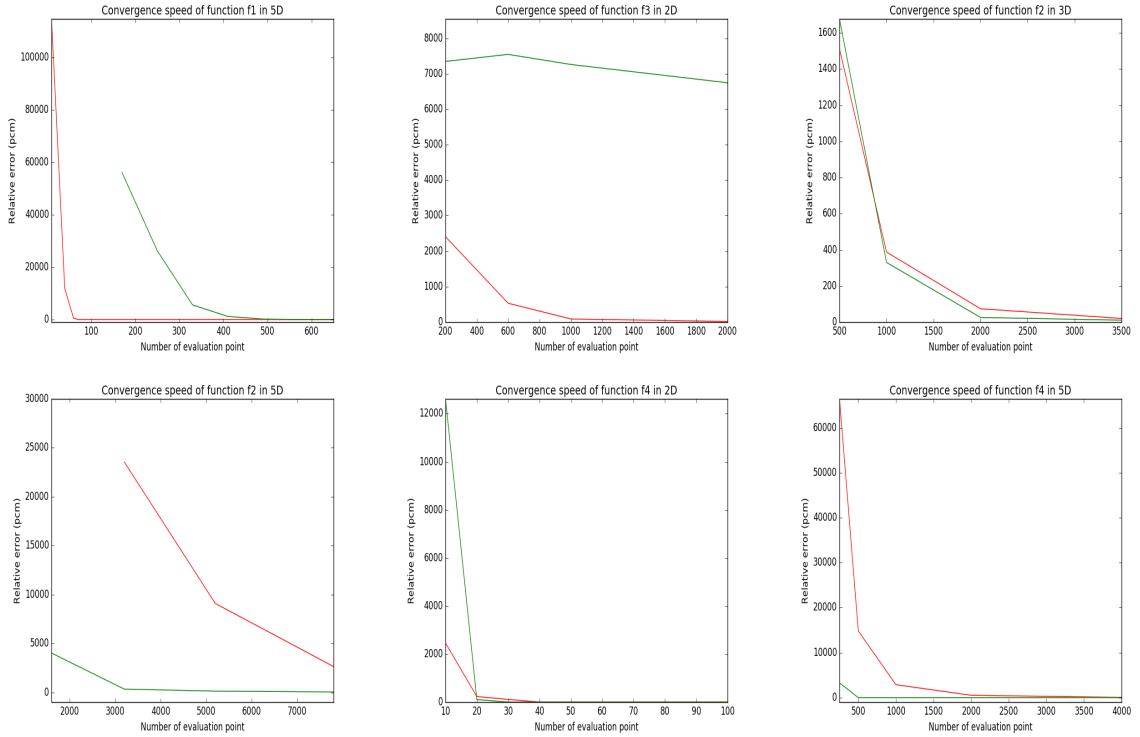


Figure 14: Comparaison des erreurs relatives ($\text{pcm} = 10^{-5}$) pour les fonctions f_1 , f_2 , f_3 , et f_4 en plusieurs dimensions.

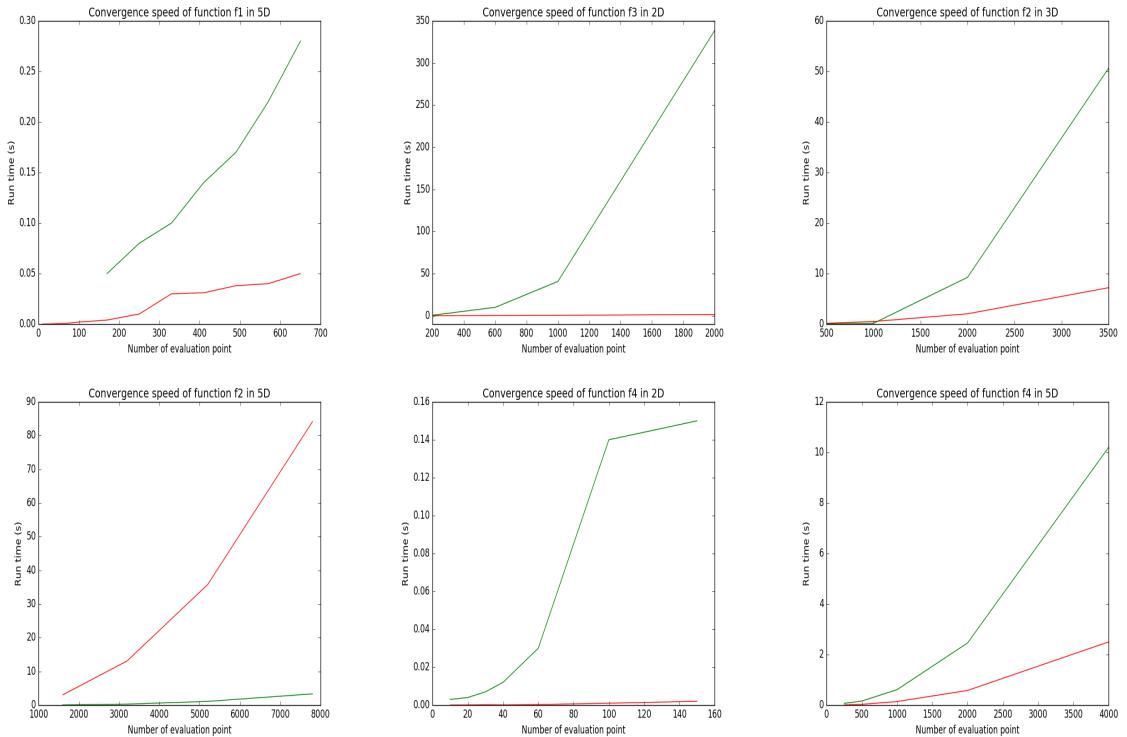


Figure 16: Comparaison des temps d'exécutions des deux méthodes pour les fonctions f_1 , f_2 , f_3 et f_4 en plusieurs dimensions.

Commentaires:

- f_1 : La méthode d'interpolation adaptative donne de meilleurs résultats que celle utilisant la décomposition de Tucker, en terme de vitesse de convergence. Ce résultat reste valable quelle soit la dimension de l'espace de départ.
- f_2 : Si $d = 2$ les 2 méthodes sont équivalentes en terme de précision mais la méthode AI est plus rapide. Cependant, quand $d > 3$ cette dernière converge très lentement et demande un grand nombre de points de calcul (> 7000).
- f_3 : La méthode de décomposition de Tucker ne converge pas (très coûteuse en nombre de points d'évaluation et en temps de calcul). Par contre, la méthode AI converge rapidement, mais demande aussi un grand nombre d'évaluations (> 3000 en 3D)
- f_3 : Si $d = 2$ les 2 méthodes sont équivalentes en terme de précision mais la méthode AI reste toujours plus rapide. Si $d > 3$, cette dernière demande un grand nombre d'évaluations, mais le temps de calcul reste relativement proche de celui obtenu avec la méthode de Tucker

Importance du choix de la séquence de points d'interpolation dans la méthode AI:

Le choix de la séquence de points d'interpolation ainsi que les fonctions de base joue un rôle très important dans la stabilité et la précision de l'opérateur d'interpolation. L'opérateur d'interpolation, dans la méthode AI, associé à la fonction f_2 (en dimension 2D) a été construit en utilisant à chaque fois des types différents de fonctions de bases et de points d'interpolation. Les résultats sont résumés dans le tableau ci-dessous (1) (les tests ont été faits en utilisant 200 itérations dans l'algorithme AI):

Fonctions de bases	Points d'interpolations	Erreur relative (pcm)	Erreur quadratique moyenne (pcm)	Nombre d'évaluations
Polynômes hiérarchiques de Lagrange	Points de Leja	99.8828	58.9808	210
Fonctions quadratiques par morceaux	Construction par dichotomie	230.349	243.573	329
Fonctions quadratiques par morceaux sur la direction 0	Construction par dichotomie sur la direction 0	44.3749	41.9915	310
Polynômes hiérarchiques de Lagrange sur la direction 1	Points de Leja sur la direction 1			

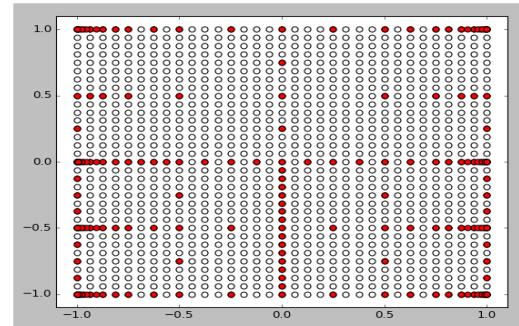
Table 1: Comparaison des résultats de l'interpolation de la fonction f_2 en 2D en utilisant différents types de fonctions de bases et différents types de points d'interpolation

On a vu que l'algorithme AI construit un nouveau point d'interpolation à chaque itération. Cependant, on remarque que le nombre de points de calcul des vraies valeurs de la fonction interpolée diffère du nombre d'itérations. Ceci s'explique par le fait qu'il arrive parfois que l'algorithme ait besoin de la valeur de l'interpolé en un point x sans pour autant l'inclure par la suite dans la séquence de points d'interpolation. En effet, ceci se déroule lors du calcul de l'erreur courante f_ν en chaque candidat, parmi lesquels on ne choisira qu'un seul.

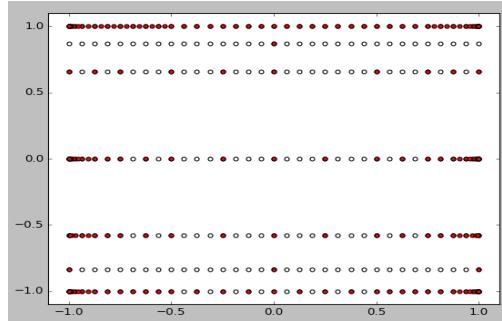
L'utilisation des fonctions quadratiques par morceaux sur la direction 0 permet de mieux corriger l'interpolation aux points voisins des extrémités, là où la variation est la plus grande (pour la fonction 1D, $x \mapsto \sqrt{1 - x^2}$). Les figures, ci-dessous (5.2) montrent les séquences finales de points d'interpolation construites par l'algorithme d'interpolation adaptative. Chaque figure correspond à une ligne du tableau précédent.



(a) Modèle de la ligne 1 de la table 1



(b) Modèle de la ligne 2 de la table 1



(c) Modèle de la ligne 3 de la table 1

Conclusion:

On peut donc conclure que l'algorithme AI donne de très bons résultats en terme de vitesse de convergence, quand il s'agit d'interpoler des fonctions polynomiales quelle que soit la dimension. Quand il s'agit de fonctions plus complexes (moins régulières), l'algorithme réussit généralement à approximer cette fonction avec une bonne précision, cependant dès que la dimension de l'espace de départ devient relativement grande, il devient nécessaire d'utiliser un plus grand nombre de points d'interpolation. Ceci est bien évidemment coûteux dans le cas réel où l'obtention de la valeur d'une fonction en un point quelconque devient coûteuse comme par exemple ce qu'on verra dans la paragraphe suivant.

5.3 Résultats des approximations de données réelles

5.3.1 Description des cas de test

Comme on vu précédemment, les fonctions qui nous intéressent sont les sections efficaces. Ces fonctions dépendent du type du combustible utilisé dans le réacteur nucléaire. Le cœur d'un réacteur nucléaire possède une structure multi-échelle. Il contient entre 150-200 assemblages combustibles disposés en treillis carré et entourés d'eau de bore. Un assemblage typique est souvent constitué de $17 \times 17 = 289$ tiges, dont 264 tiges de carburant et 25 tiges vides. Il existe plusieurs types d'assemblages selon la position et la composition des barres de combustible.

- **Assemblage UOX (Uranium Oxide): UO_2 :** La plupart des réacteurs utilisent l'uranium enrichi dans l'isotope 235 (avec un enrichissement entre 3%-5%) car l'uranium naturel contient seulement 0.7% d'uranium-235. L'assemblage constitué par le carburant d'UOX sera appelé l'assemblage UOX.
- **Assemblage MOX (Mixed Oxide): $UO_2 - PuO_3$:** A l'instar de l'uranium naturel, le plutonium (P_u) est aussi une source fissile de réacteurs nucléaires. Le plutonium peut être extrait de combustibles usés. Ceci nous permet de recycler les carburants utilisés. De plus, MOX, un mélange de combustible uranium et plutonium, est aussi utilisé pour les réacteurs nucléaires. L'assemblage constitué par quelques carburants de MOX sera appelé l'assemblage de MOX
- **Assemblage UOX-Gd (UOX-Gadolinium): $UO_2 - Gd_2O_3$:** Un assemblage UOX-Gd est un assemblage UOX dans lequel certaines positions des barres de combustible UOX sont remplacées par des barres $UO_2 - PuO_3$. Étant donné que la gadolinie Gd_2O_3 est un poison neutronique (absorbe les neutrons), les barres $UO_2 - PuO_3$ sont utilisées comme des barres anti-poison pouvant limiter l'excès de fissions, afin d'atténuer les pics de puissance localisés.

La notion de section efficace en physique des particules permet de quantifier les taux des réactions des neutrons. Elle représente la probabilité d'interaction entre un neutron incident et un (des) noyau(x) cible(s). Les différents types de sections efficaces se distinguent par leur réaction correspondante qu'on note r . Les valeurs que prennent les sections efficaces dépendent de la nature du combustible utilisé (du type d'assemblage). Une section efficace est notée Σ_r^g , (g faisant référence au groupe d'énergie).

Dans notre cas, on peut avoir la *macro totale* – Σ_t^g , la *macro fission* – Σ_f^g , la *macro nu*fission* – $\nu\Sigma_f^g$, la *macro absorption* – $\nu\Sigma_a^g$, la *macro scattering* – $\Sigma_{so}^{g \rightarrow g'}$ (où $g, g' \in \{1, 2\}$, et o est l'ordre d'anisotropie).

Ces sections efficaces dépendent de 5 paramètres physique: *burnup*, *température du combustible*, *densité du modérateur*, *concentration en bore* et *niveau de xénon*. Ces paramètres varient dans les intervalles présentés dans le tableau ci-dessous (2).

Paramètre	Valeur minimale	Valeur maximale
Burnup MWd/t	0.0	80000.0
Température du combustible, C	286.0	1100.0
Densité du modérateur, g/cm3	0.602	0.753.0
Concentration en Bore, ppm	0.0	1800.0
Niveau de xénon, %	0.0	1.0

Table 2: Intervalles des paramètres

Les données réelles fournies par le département R&D d'EDF sont regroupées dans des fichiers texte (un fichier par type de section efficace). On a donc 12 fichiers pour un seul type d'assemblage, soit 36 fichiers en tout.

Ces données comportent les points formant une grille de référence (qui permettra d'évaluer les différentes méthodes), les vraies valeurs des sections efficaces en ces points (valeurs obtenues par un code appelé APOLLO2 développé chez EDF), les approximations obtenues par la méthode COCAGNE (qui n'est autre que la méthode d'interpolation multilinéaire), les résultats obtenus avec la méthode de décomposition de Tucker et les erreurs de précision obtenues avec ces deux méthodes.

Les données sont organisées en 10 colonnes correspondant aux valeurs suivantes: **count** **bu** **tc** **density** **cb** **xe** **value_A** **value_C** **value_T** **err_C** **err_T**

- **count:** l'indice des points que l'on parcourt sur la grille de référence (e.x : la grille contient 14000 points, chaque point a les coordonnées suivantes: (bu, tc, density, cb, xe)).
- **(bu, tc, density, cb, xe):** les valeurs des points de la grille de référence. La colonne i contient le paramètre i
- **value_A:** valeurs de section efficace calculées par APOLLO2 sur les points dans la grille de référence. On les considère comme les valeurs exactes.
- **value_C:** valeurs de section efficace évaluées par le code Cocagne, il s'agit la méthode multilinéaire.
- **value_T:** valeurs de section efficace évaluées par la méthode de Tucker.
- **err_C:** erreurs relatives commises par la méthode Cocagne.
- **err_T:** erreurs relatives commises par la méthode de Tucker.

On a vu que l'algorithme d'interpolation adaptative choisit les points d'interpolation au fur et à mesure qu'il construit l'opérateur d'interpolation. Les points d'interpolations utilisés ainsi que leurs ordres dépendent de la fonction interpolée et du type des fonctions de base. On ne peut donc pas prévoir à l'avance les points où on a besoin d'avoir la valeur exacte des sections efficaces. Il nous faut un moyen d'appeler le code APOLLO2 à chaque fois qu'on veut avoir la valeur d'une section efficace en un point.

Cela n'est évidemment pas intéressant, car le code APOLLO2 est très coûteux et ne peut être appelé que sur une grille tensorielle. On pourrait penser à sauvegarder les points d'interpolation et les évaluer tous en même temps, mais cela n'est pas possible, car pour avoir le nouveau point d'interpolation, l'algorithme AI a besoin de la valeur des sections efficaces au point précédent,

Étant donné qu'on ne dispose pas du code APPOLLO2, il n'est pas possible d'avoir les vraies valeurs des sections efficaces en n'importe quel point. Par contre, on peut avoir le résultat de l'approximation obtenu par la méthode de décomposition de Tucker. Donc l'alternative qu'on a adopté et de supposer que les valeurs obtenues par la méthode de Tucker correspondent aux "vraies valeurs" des sections efficaces.

Des tests ont été faits pour chaque type de section efficace. Les résultats obtenus ont été

ensuite stockés dans des fichiers et organisés en 12 colonnes correspondants aux valeurs suivantes: `count bu tc density cb xe value_A value_C value_T value_AI err_C err_T err_AI_Tucker err_AI_Apollo` où

- **value_AI:** valeurs de section efficace évaluées par la méthode d'Interpolation Adaptive.
- **err_AI_Tucker:** erreurs relatives commises par la méthode d'Interpolation Adaptive par rapport à la méthode Tucker.
- **err_AI_Apollo:** erreurs relatives commises par la méthode d'Interpolation Adaptive par rapport à aux données APOLLO2.

Discrétisation de l'espace de phase des paramètres:

La grille multilinéaire possède un grand nombre de points dans la direction burnup (33 points), et 2 (ou 3) points dans les autres directions. La grille de Tucker contient, quant à elle, 5 grilles correspondant à 5 directions. Chacune comporte 5 points de Clenshaw-Curtis dans la direction étudiée et 2 points dans toutes les autres directions, sauf pour la grille de burnup qui possède 25 points dans la direction burnup. L'algorithme AI, n'exige pas une grille initiale comme pour les deux autres méthodes. En effet, il construit un nouveau point à chaque itération. Ainsi, on obtient la grille finale en sortie de l'algorithme. Cette grille dépend donc du type de fonctions de base utilisé et de la fonction interpolée. Dans tous les tests effectués, les fonctions de bases utilisées sont des fonctions quadratiques par morceaux suivant la direction burnup et des polynômes de Lagrange suivant les autres directions. Tout ce qui précède est décrit dans le tableau ci-dessous 3. Ici, les informations sur la grille AI concernent la section efficace Σ_t^1 de type MOX.

Direction	Grille Cocagne	Grille Tucker	Grille AI
Burnup	33	25 ($25 * 2^4 = 400$)	54
Température du combustible	3	25 ($25 * 2^4 = 400$)	6
Densité du modérateur	3	25 ($25 * 2^4 = 400$)	9
Concentration en Bore	3	25 ($25 * 2^4 = 400$)	5
Niveau de xénon	2	25 ($25 * 2^4 = 400$)	5
Total	$1782 = 33 * 3^3 * 2$	$720 = 25 * 2^4 + 4 * 5 * 2^4$	832

Table 3: Discrétisation de la grille multilinéaire, de la grille de Tucker et de la grille construite par l'algorithme AI

Le nombre total de point d'interpolation dans la méthode AI ne correspond pas au produit des nombres de points sur chaque direction. En effet, en sortie de l'algorithme, on n'obtient pas forcément une grille tensorielle, mais plutôt un ensemble monotone de multi-indices. Un exemple en 2D est présenté dans la figure ci-dessous (18). On a 20 indices (ordres) suivant la direction x et 15 indices suivant la direction y . On a en total 176 couple d'indices utilisés, au lieu de 300.

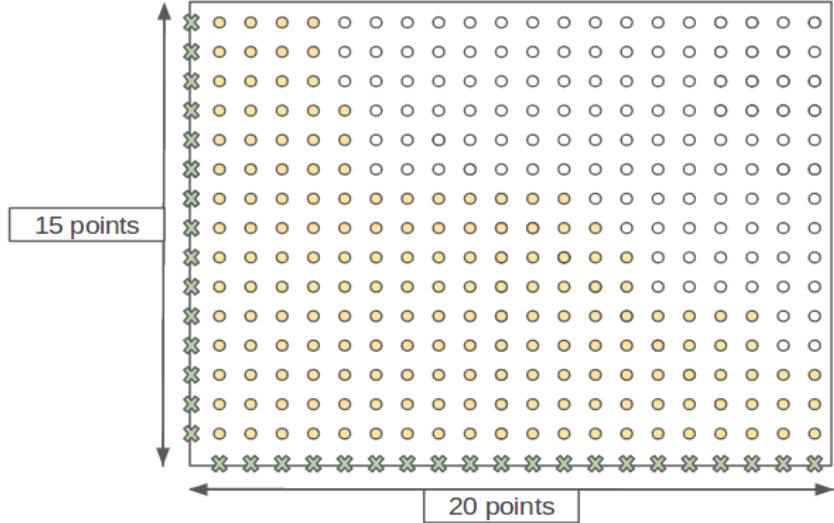


Figure 18: Exemple de points d’interpolation obtenu par l’algorithme AI

L’interpolation de la section efficace Σ_t^1 de type MOX, donne en sortie la séquence de points d’interpolations construits. Ces points en 5 dimensions sont projetés sur chaque direction dans les figures ci-dessous (5.3.1). On remarque l’importance et l’efficacité du caractère adaptatif de l’algorithme AI. En effet, on voit que la majorité des points ont été choisis dans la direction burnup, ce qui est conforme à nos attentes et aux discréétisations faites pour les deux autres méthodes (soit toujours un plus grand nombre de points suivant la variable burnup).

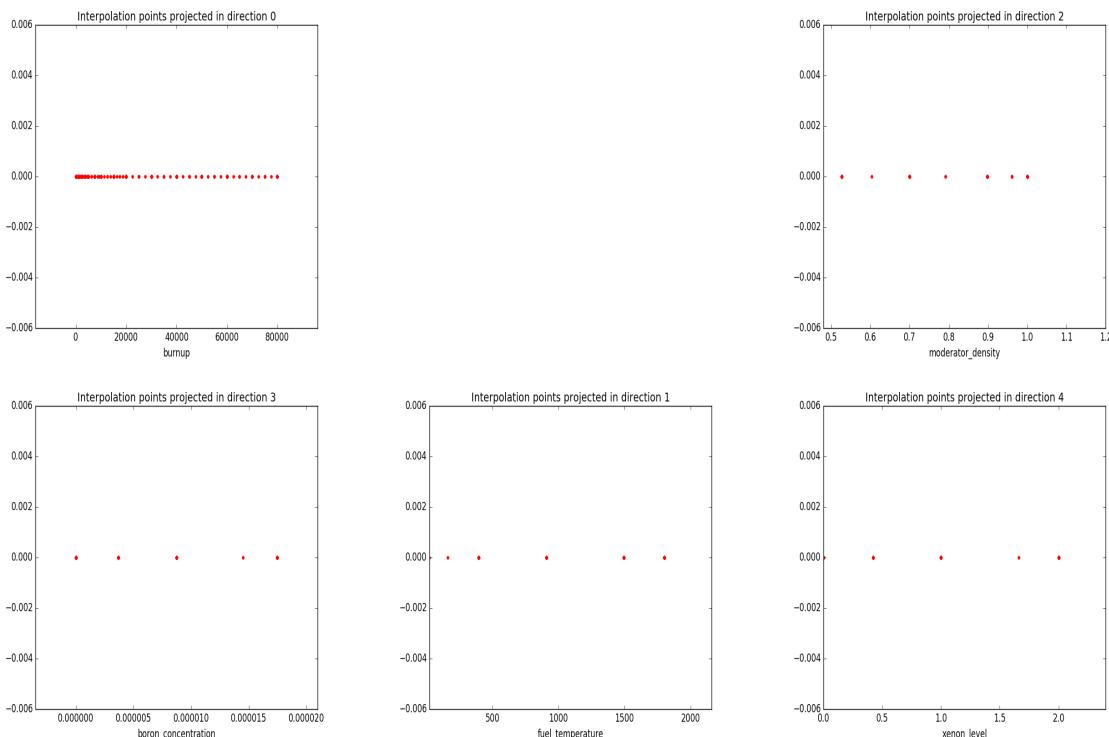


Figure 20: Projection des points d’interpolation, obtenus en interpolant Σ_t^1 de type MOX, sur chaque direction.

Erreurs d'approximation des sections efficaces:

On compare la précision des différentes méthodes sur chaque nœud \mathbf{x}_i de la grille de référence. L'erreur d'approximation sur la grille de référence est définie soit par l'erreur relative $e^{(inf)}$, soit par l'erreur quadratique moyenne (Root Mean Square Error):

$$e^{(inf)} = \max_{\mathbf{x}_i \in \text{reference grid}} \underbrace{\left| \frac{\tilde{f}(\mathbf{x}_i) - f(\mathbf{x}_i)}{\max |\{f(\mathbf{x}_i)\}|} * 10^5 \right|}_{\text{relative error}}$$

$$e^{(RMSE)} = \sqrt{\sum_{i=1}^N \frac{((f(\mathbf{x}_i) - \tilde{f}(\mathbf{x}_i)) * 10^5)^2}{N}} \text{ with } N = \sharp(\text{reference grid})$$

où $f(\mathbf{x}_i)$ et $\tilde{f}(\mathbf{x}_i)$ sont respectivement la valeur exacte et la valeur approximative (obtenue par chaque méthode) de la section efficace au point $\mathbf{x}_i \in \text{reference grid}$. Le facteur 10^5 est ajouté afin d'exprimer les résultats en pcm ($1\text{pcm} = 10^{-5}$)

Erreurs d'approximation de la réactivité:

La réaction de fission est un processus dans lequel un noyau fissile (typiquement Uranium U_{92}^{235} , Plutonium Pu_{94}^2 39) absorbe des neutrons et se divise en noyaux plus légers en produisant de nouveaux neutrons libres et en libérant de l'énergie sous forme de chaleur. Ces nouveaux neutrons entrent ensuite en collision avec d'autres noyaux fissiles ce qui produit une réaction en chaîne.

Le comportement global de la chaîne de fission nucléaire est relié à un facteur appelé facteur de multiplication efficace k_{eff} . Ce facteur est utilisé pour décrire le nombre moyen de neutrons d'une réaction de fission qui causent d'autres fissions.

La valeur de k_{eff} est classée de la façon suivante:

- $k_{eff} > 1$ (*super-criticité*): le nombre de fission croît exponentiellement et la réaction est explosive.
- $k_{eff} = 1$ (*criticité*): le nombre de fission est stable (constant) dans le temps.
- $k_{eff} < 1$ (*sous-criticité*): le nombre de fission décroît exponentiellement et la réaction finit par s'arrêter.

Dans certains cas simplifiés, on peut calculer la réactivité en utilisant la formule suivante (voir pages 1172, 1173, 1221 du livre [5])

$$\text{reactivity} = 1 - \frac{1}{k_{eff}}, \text{ avec } k_{eff} = k_\infty = \frac{\nu \Sigma_f^1 * (\Sigma_t^2 - \Sigma_{s0}^{2 \rightarrow 2}) + \nu \Sigma_f^2 * \Sigma_{s0}^{1 \rightarrow 2}}{(\Sigma_t^1 - \Sigma_{s0}^{1 \rightarrow 1}) * (\Sigma_t^2 - \Sigma_{s0}^{2 \rightarrow 2}) - \Sigma_{s0}^{1 \rightarrow 2} * \Sigma_{s0}^{2 \rightarrow 1}}$$

On mesure l'erreur de la réactivité, en utilisant la formule suivante:

$$\mathbf{e}_{reactivity}(inf) = \max_{\mathbf{x}_i \in \text{reference grid}} \left| \left(\frac{1}{k_\infty(\mathbf{x}_i)} - \frac{1}{\tilde{k}_\infty(\mathbf{x}_i)} \right) * 10^5 \right|$$

5.3.2 Résultats numériques

Comparaison de la précision:

On présente ici les résultats de l'approximation des sections efficaces de type MOX avec chacune des méthodes. Les précisions obtenus avec la méthode d'interpolation adaptative sont comparées au résultats de la décomposition de Tucker ainsi que ceux obtenu par intpolation multilinéaire (Cocagne) sur une grille de référence Contenant 6000 points. Les résultats sont présentés dans le tableau ci-dessous (4). Pour chaque méthode, on donne l'erreur relative absolue ainsi que l'erreur quadratique moyenne obtenue pour chaque type de section efficace ainsi que pour la réactivité.

Section efficace	$e_{Cocagne}^{(inf)}$	$e_{Tucker}^{(inf)}$	$e_{AI}^{(inf)}$	$e_{Cocagne}^{(RMSE)}$	$e_{Tucker}^{(RMSE)}$	$e_{AI}^{(RMSE)}$
Σ_t^1	36.197	15.0578	1.48278	8.56588	2.72007	0.22795
Σ_t^2	839.957	58.1971	6.94451	618.742	52.0822	4.91745
Σ_a^1	751.201	20.2378	10.0386	4.82434	0.114286	0.044465
Σ_a^2	401.499	28.6188	7.90148	39.195	3.07643	0.498357
$\nu\Sigma_f^1$	538.307	11.9292	11.0837	2.24702	0.0472871	0.0336646
$\nu\Sigma_f^2$	288.344	24.8048	3.77013	51.6756	4.33486	0.413668
$\Sigma_{s0}^{1 \rightarrow 1}$	28.5216	12.4517	1.45791	6.37984	2.31958	0.209587
$\Sigma_{s0}^{1 \rightarrow 2}$	1835.02	482.944	27.9257	3.91736	1.25474	0.0440527
$\Sigma_{s0}^{2 \rightarrow 1}$	409.717	11.4509	7.13451	2.67437	0.0718079	0.0327384
$\Sigma_{s0}^{2 \rightarrow 2}$	917.836	62.0243	7.28612	596.271	49.8085	4.50027
Σ_f^1	538.507	11.2092	11.3262	0.773387	0.0154022	0.0118336
Σ_f^2	287.846	25.1117	3.38409	17.8697	1.49996	0.137742
Reactivity	785.077	93.0013	11.1861	164.879	31.879	3.48791

Table 4: Comparaison de la précision des 3 méthodes sur une grille de référence de 6000 points

Afin de pouvoir visualiser les résultats numériques obtenus, ces derniers ont été représenté par des graphes (un graphe par section efficace). Chaque graphe montre des nuages de points en 2D (l'abscisse de chaque point correspond à l'indice du point de la grille de référence (**count**), et l'ordonné correspond à l'erreur relative obtenue en ce point avec une des trois méthodes).

- Le nuage des point bleus correspond aux erreurs relatives obtenus avec la méthode Cocagne.
- Le nuage des point verts correspond aux erreurs relatives obtenus avec la méthode de Tucker.
- Le nuage des point rouges correspond aux erreurs relatives obtenus avec la méthode AI.

Les figures ci-dessous (5.3.2) montrent les résultats obtenus avec 2 exemples de sections efficaces ($\nu\Sigma_f^2$ de type MOX et $\Sigma_{s0}^{2 \rightarrow 1}$ de type UOX).

Dans les figures de gauche, on voit les erreurs relatives commises avec chacune des 3 méthodes par rapport aux résultats de APOLLO2. Afin de pouvoir mieux voir les résultats des 2 méthodes Tucker et AI, les nuages de points correspondant à la méthode Cocagne ont été supprimés dans les figures de droite. Finalement, les figures du bas (milieu) montrent les erreurs relatives commises par la méthode AI par rapport aux résultats de la méthode de Tucker (considérées comme "vraies valeurs" lors de l'interpolation).

On s'intéresse maintenant aux résultats de l'approximation de la réactivité, obtenues avec chaque méthode et pour chaque type de réacteur (MOX, UOX, UOX-Gd). Dans les figures ci-dessous (5.3.2), on garde les même conventions, vus dans le paragraphe précédent, pour les couleurs des nuages de points ainsi que pour la disposition des figures.

On remarque que l'algorithme AI accomplit l'approximation avec une bonne précision, vu que dans la plupart des cas le nuage des points correspondant se confond presque avec celui obtenu par la méthode de Tucker. Cela veut dire que les erreurs obtenues avec la méthode AI sont très proches de ceux obtenus avec la méthode de Tucker. En effet, cela s'explique par le fait que les erreurs relatives obtenues par rapport aux résultats de Tucker (considérées comme "vraies valeurs") sont relativement faibles. On peut confirmer cela grâce aux figures de droites où l'erreur relative absolue est très faible dans la plupart des cas.

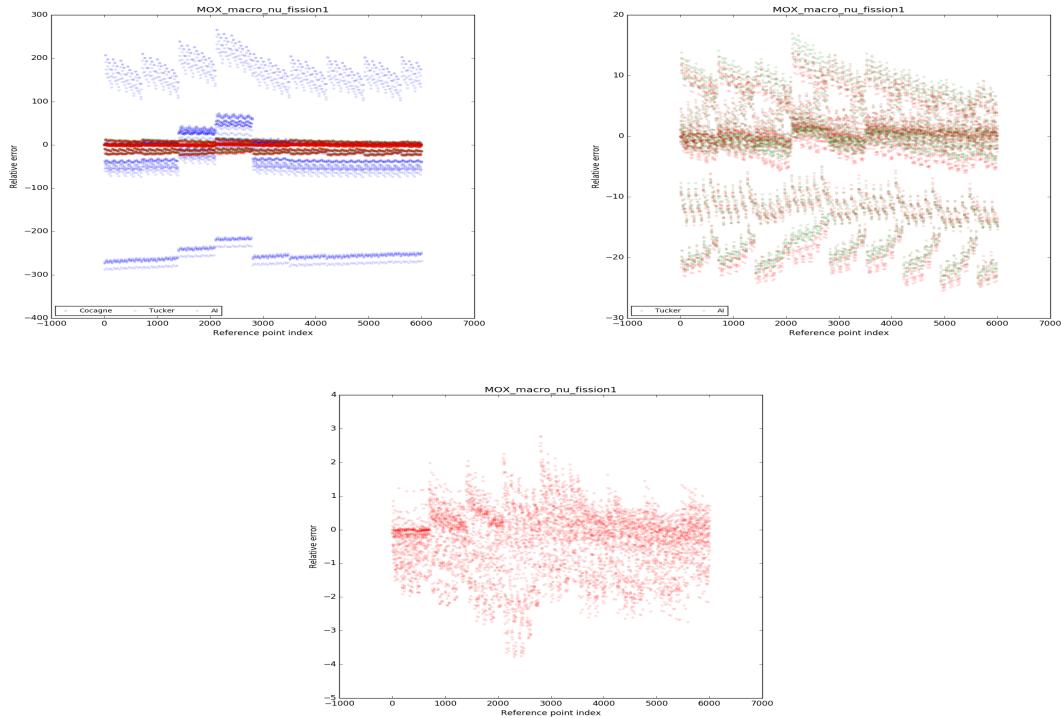


Figure 22: Comparaison des erreurs relatives (pcm) pour la section efficace $\nu\Sigma_f^2$

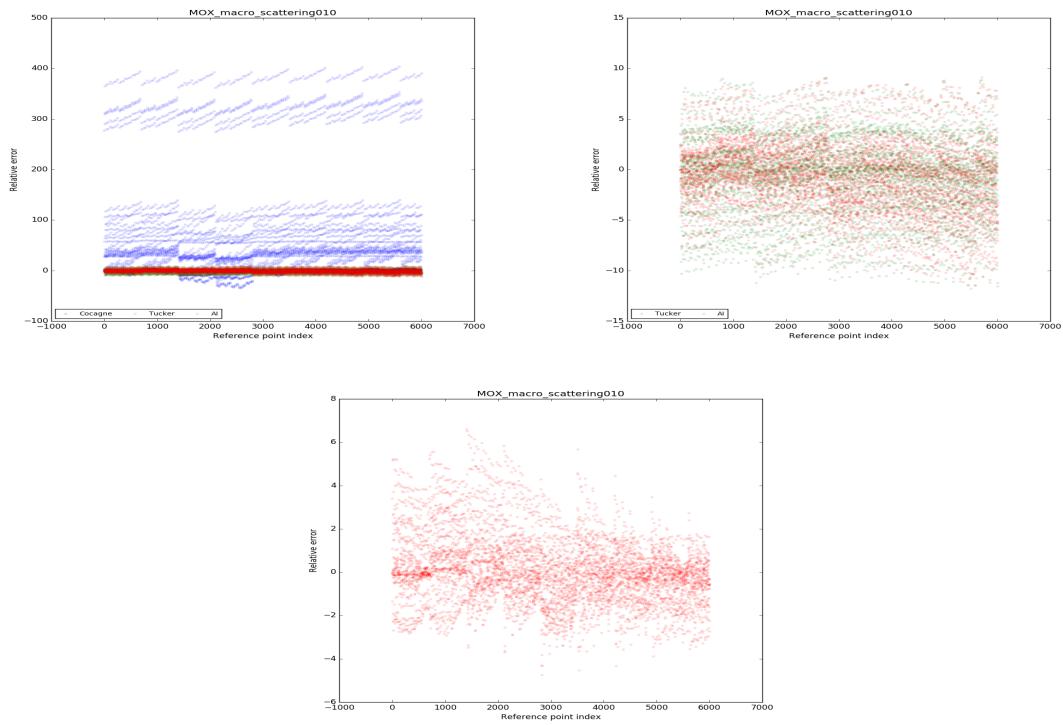


Figure 23: Comparaison des erreurs relatives (pcm) pour la section efficace $\Sigma_{s0}^{2 \rightarrow 1}$

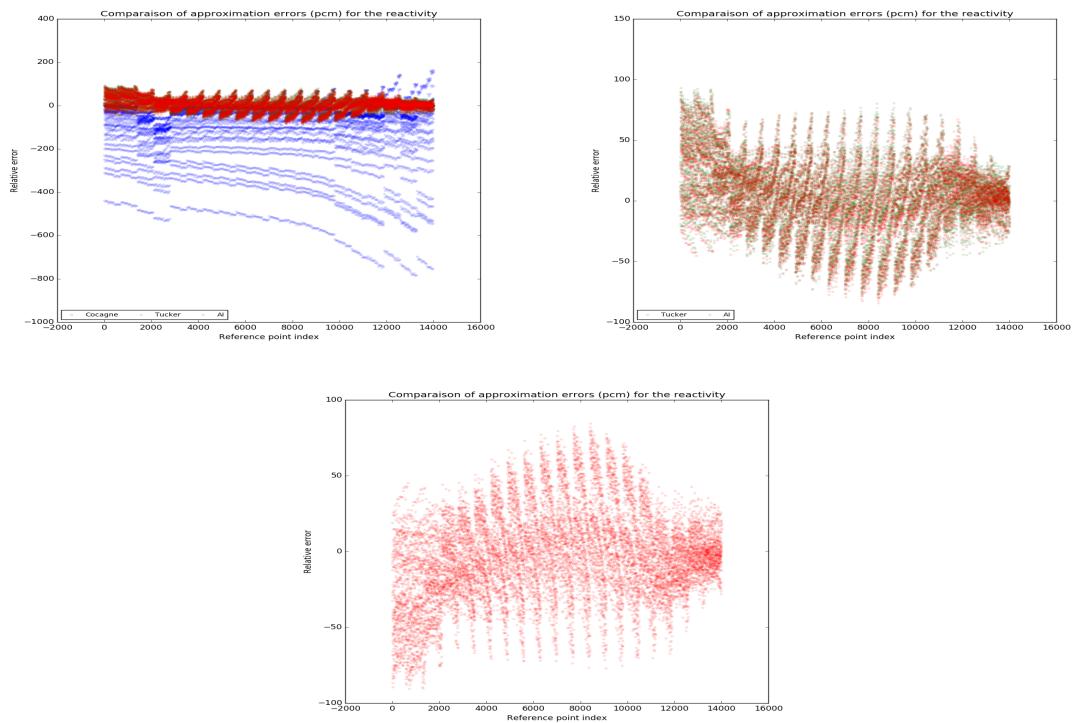


Figure 24: Comparaison des erreurs d'approximation (pcm) de la réactivité pour le type d'assemblage MOX

Vitesse de convergence de la méthode AI:

Les courbes ci-dessous tracent l'erreur relative absolue en fonction du nombre d'itérations nécessaire pour approximer toutes les sections efficaces de type MOX.

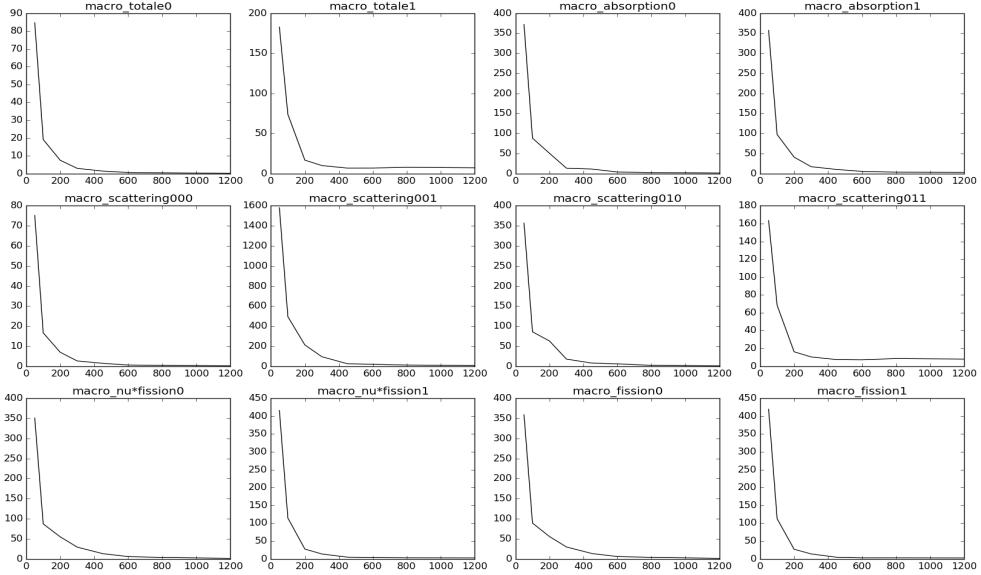


Figure 25: Courbes de convergence pour chaque section efficace de type MOX

On remarque que dans tous les cas, au bout d'un certain nombre d'itérations (~ 500), l'erreur relative absolue se stabilise et atteint un seuil minimal assez faible dépendant de la section efficace. Ce seuil ne dépasse pas les 10 pcm ce qui correspond à une erreur assez faible et une précision assez importante. Donc, on peut considérer qu ce nombre d'itérations est un bon choix pour réaliser les tests de comparaison avec les deux autres méthodes d'approximation en grande dimension.

Comparaison du nombre de points d'évaluation:

Le nombre de points de calcul dans la méthode de Tucker inclut les évaluations sur tous les noeuds de la grille de Tucker (5 grilles pour 5 paramètres avec 720 noeuds au total) et les évaluations faites sur le point d'évaluation x sur le coté gauche de (2). Le nombre de points de calcul pour la méthode Cocagne est égal au nombre de noeuds dans la grille multilinéaire. Par ailleurs, le nombre de points de calcul pour la méthode AI inclut le nombre de points d'interpolation construits et le nombre de candidats non sélectionnés lors de la construction de l'opérateur d'interpolation.

Dans le tableau ci-dessous (5), on voit le nombre total de points de calcul nécessaire pour approximer une section efficace avec chaque méthode. Ce nombre n'est pas fixé à l'avance dans la méthode AI, contrairement aux 2 autres méthodes. Il varie en fonction du type de fonctions de bases et de la nature des points d'interpolation utilisés. Dans la dernière colonne, on affiche le nombre total de points de calculs obtenu lors de l'interpolation des 3 fonction Σ_t^1 , Σ_f^1 et Σ_a^1 de type MOX.

Interpolation Multilinéaire	Décomposition de Tucker	Interpolation Adaptative
$\underbrace{720}_{5 \text{ grilles de Tucker}} + \underbrace{378}_{\text{points d'évaluation}} = 1098$	1782	(500 itérations) 832, 881, 836

Table 5: Comparaison du nombre de points de calcul entre le modèle de décomposition de Tucker, le modèle multilinéaire et la méthode d'interpolation adaptative

Ces résultats montrent qu'en utilisant le modèle de décomposition de Tucker, le nombre de points de calcul diminue de 38% par rapport à la méthode Cocagne. D'un autre côté, en utilisant l'algorithme d'interpolation adaptative, ce nombre diminue de 24% par rapport à la méthode de Tucker, et de 53% par rapport à la méthode Cocagne.

Comparaison du stockage:

La quantité de stockage nécessaire à la décomposition de Tucker pour une section efficace inclut les valeurs de la discréétisation axiale, les vecteurs propres sélectionnés et le coefficient a . La quantité de stockage pour le modèle multilinéaire les valeurs de la discréétisation axiale et les valeurs prise par la section efficace sur les nœuds de la grille de référence. Le nombre de valeurs de la discréétisation axiale ($25 + 4 * 5 = 45$ pour la méthode de Tucker et $33 + 3 + 3 + 3 + 2 = 44$ pour la grille multilinéaire) est le même pour toutes les sections efficaces.

Contrairement à la méthode de Tucker, pour laquelle la taille du stockage dépend de la section efficace considéré, cette taille là reste constante et est égal au nombre de points d'interpolations choisi à l'avance, si on utilise la méthode AI (ou Cocagne). Pour la méthode AI, ce nombre est exactement égal au nombre d'itérations. Finalement, nous obtenons les résultats résumés dans le tableau ci-dessous (6).

Section efficace	Taille du stockage		
	Tucker	AI	Cocagne
Σ_t^1	244	500	1782
Σ_t^2	192	500	1782
Σ_a^1	355	500	1782
Σ_a^2	355	500	1782
$\nu\Sigma_f^1$	388	500	1782
$\nu\Sigma_f^2$	290	500	1782
$\Sigma_{s0}^{1 \rightarrow 1}$	244	500	1782
$\Sigma_{s0}^{1 \rightarrow 2}$	371	500	1782
$\Sigma_{s0}^{2 \rightarrow 1}$	416	500	1782
$\Sigma_{s0}^{2 \rightarrow 2}$	192	500	1782
Σ_f^1	327	500	1782
Σ_f^2	290	500	1782

Table 6: Comparaison de la taille du stockage entre le modèle de décomposition de Tucker, le modèle multilinéaire et la méthode d'interpolation adaptative

Ce résultat montre qu'en utilisant le modèle de décomposition de Tucker, la taille du stockage diminue d'un facteur allant de 4 à 9 (selon la section efficace) comparé au modèle Cocagne, et de 5 à 6 comparé au modèle AI.

6 Impressions personnelles

Au début du stage et notamment au cours des premières présentations du sujet, ce dernier m'a paru plutôt difficile. Il m'a fallu une certaine période de documentation et de recherche avant de pouvoir entamer la partie pratique. Ainsi, au bout des premières semaines de mon stage, j'ai réussi à bien assimiler la partie théorique du sujet. Ce sujet m'a paru alors encore plus intéressant car j'ai pu constater sa richesse en matière de spectre d'applications. De plus j'ai eu l'occasion de beaucoup apprendre notamment sur l'interpolation en grande dimension en général, mais aussi au niveau de la programmation informatique. En effet, j'ai pu améliorer mes connaissances en python et mettre en application tout ce que j'ai appris en algorithmique et en C++.

De plus, étant donnée que mes encadrants ont participé à l'étude et la réalisation de la partie théorique, il m'était facile d'évoluer et d'avancer dans ma compréhension des différents points-clés de mon sujet de stage. En effet, le fait qu'ils soient souvent disponibles pour discuter des points potentiellement imprécis et de l'organisation du travail m'a évité une importante perte de temps.

Par ailleurs, j'ai été agréablement surpris par l'environnement et l'esprit de travail au sein d'un laboratoire de recherche.

J'ai, de plus, apprécié la liberté qu'on m'a laissée pour le choix de l'environnement de travail et l'organisation des différentes tâches et l'ordre de leurs réalisations. Ceci m'a permis non seulement d'apprendre à être autonome mais aussi d'améliorer mes compétences en analyse et en programmation.

7 Prise en compte de l'impact environnemental et sociétal

Dans un monde de plus en plus impacté par l'évolution humaine, l'écologie est un sujet souvent pris en compte de nos jours. On en parle beaucoup dans les domaines de transport, de la nutrition, de l'économie et du marketing, ...

Qu'en est-il alors des nouvelles technologies et de la recherche mathématique?

Actuellement étudiant à l'Ensimag et stagiaire dans un grand laboratoire de mathématiques appliquées, ce sujet m'a intéressé et a aussi suscité l'attention des membres de mon équipe qui m'ont aidé à enquêter là-dessus

Je travaille quotidiennement avec 4 personnes dans un bureau éclairé au néon. Nous utilisons chacun un ordinateur de bureau pendant 7h par jours.

Actuellement, le laboratoire met, à la disposition des employés, de plus en plus de moyens écologiques. Par exemple, l'éclairage s'éteint automatiquement en absence prolongée de mouvement dans la pièce, de plus les déchets papier sont recyclés.

Quant au sujet de mon stage, les méthodes mathématiques qui y sont proposées permettent de réduire considérablement le coût des calculs des sections efficaces en neutronique, à

précision égale, et par conséquent de limiter la consommation énergétique de gros serveurs de calcul.

8 Conclusion

References

- [1] Abdellah Chkifa, Albert Cohen, and Christoph Schwab. Multidimensional interpolation and construction for Leja and RLeja points. 2013.
- [2] Abdellah Chkifa, Albert Cohen, and Christoph Schwab. High-Dimensional Adaptive Sparse Polynomial Interpolation and Applications to Parametric PDEs. *Foundations of Computational Mathematics*, 14(4):601–633, 2014.
- [3] Thi Hieu Luu. *Amélioration du modèle de sections efficaces dans le code de cœur COCAGNE de la chaîne de calculs d'EDF*. Theses, 2017.
- [4] Thi Hieu Luu, Yvon Maday, Matthieu Guillo, and Pierre Guérin. A new method for reconstruction of cross-sections using Tucker decomposition. working paper or preprint, March 2017.
- [5] Marguet Serge. *La physique des réacteurs nucléaires, 2ème édition*. Tec and Doc Lavoisier, 2013.