



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 | 2 |
| 2 | Présentation de la structure d'accueil | 3 |
| 3 | Présentation de la problématique du projet | 4 |
| 3.1 | Présentation du contexte | 4 |
| 3.2 | Objectifs attendus, problème à résoudre | 6 |
| 4 | Solution technique mise en œuvre | 7 |
| 4.1 | Description de la solution envisagée | 7 |
| 4.1.1 | Interpolation monovariée et tensorisation: | 7 |
| 4.1.2 | Construction hiérarchique de l'opérateur d'interpolation: | 9 |
| 4.1.3 | Interpolation adaptative et séquence de points d'interpolation: | 11 |
| 4.2 | Implémentation de la solution | 15 |
| 5 | Évaluation de l'efficacité de la méthode | 19 |
| 5.1 | Présentation du modèle de décomposition de Tucker | 20 |
| 5.2 | Tests et comparaisons | 20 |
| 5.2.1 | Fonctions analytiques prédéfinies | 20 |
| 5.2.2 | Données réelles | 20 |
| 5.3 | Conclusion et remarques | 25 |
| 6 | Impressions personnelles | 25 |
| 7 | Prise en compte de l'impact environnemental et sociétal | 25 |
| 8 | Conclusion | 26 |
| | References | 27 |

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 «bibliothèque nucléaire» 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 J.-L. 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 permet 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 clés: 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 théoriques; contrôle, optimisation et finance; médecine et biologie; traitement du signal et des données.



Figure 1: Localisation de la structure d'accueil

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 appellée 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, telle 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-hydroliques 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 figure 2).

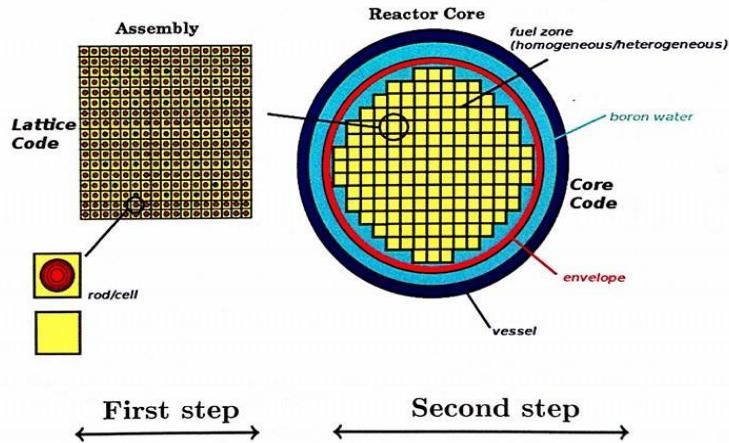


Figure 2: Schéma à 2 étapes basé sur la structure multi-échelle du noyau du réacteur

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

- **Code réseau:** permet de précalculer 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 énergétique (calcul hors ligne). L'information obtenue sur les sections efficaces est stockée dans des fichiers (appelés les **bibliothèques neutroniques**).
- **Code de cœur:** 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 (figure 3) illustre le modèle de reconstruction des sections efficaces.

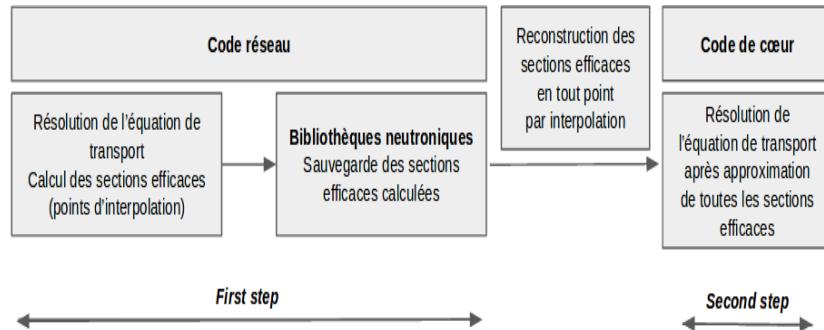


Figure 3: Schéma à 2 étapes pour la simulation du cœur du réacteur

Avec ce modèle, le nombre de points de discrétisation augmente exponentiellement en fonction du nombre de paramètres ou de manière considérable 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 précalculs et ceci en choisissant les points de discrétisation 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 couteux. 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 point $y^i \in \mathcal{P}, i \in 1..m$. Ensuite on évalue $f_i = f(y^i) = f(y_1^i, \dots, y_d^i)$ en faisant m appel au code couteux. Ensuite, 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} coincide 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. Par la suite, on évaluera le coût d'une telle opération et son évolution en fonction du nombre de points d'interpolation et de la dimension de l'espace de phase des paramètres. Ensuite, on présentera une méthode d'interpolation adaptative en grande dimension (moins couteuse) basée sur une construction hiérarchique de l'opérateur d'interpolation.

4.1.1 Interpolation monovariée 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,

$$I_k(f) = \sum_{i=0}^k f(y_i) l_i^k \quad (1)$$

avec $l_i^k(y) = \prod_{j=0, j \neq i}^k \frac{(y-y_j)}{(y_i-y_j)}$, polynômes d'interpolation de Lagrange de degrès $(k-1)$ associé à la séquence $\{y_0, \dots, y_k\}$ (voir figure 4).

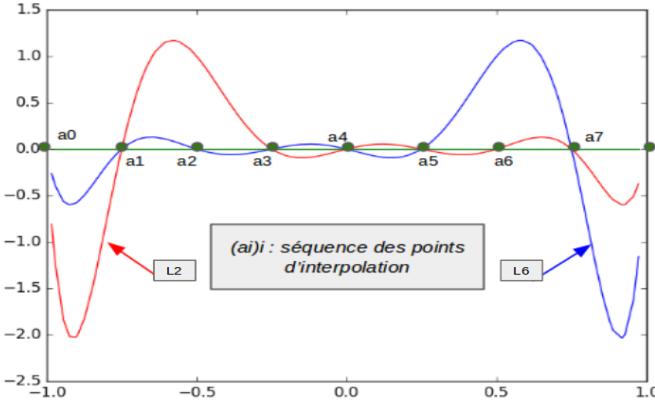


Figure 4: Exemples de polynômes d'interpolation de Lagrange

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

$$\begin{aligned} \tilde{f}(y_1, \dots, y_d) &= \bigotimes_{i=1}^d I_k(f)(y_1, \dots, y_d) \in \mathbb{Q}_{k-1} = \text{vect} \{y \rightarrow y_1^{\alpha_1} \dots y_d^{\alpha_d}\} \\ &= \sum_{k_1=1}^k \dots \sum_{k_d=1}^k (f(y_{k_1}, \dots, y_{k_d}) l_{k_1}(y_1) \dots l_{k_d}(y_d)) \end{aligned} \quad (2)$$

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 couteux, mais aussi, si on souhaite ajouter 1 points suivant la variable i ça revient à ajouter $\prod_{j=1, j \neq i}^d k_j$ nœuds dans la grille (voir figure 5).

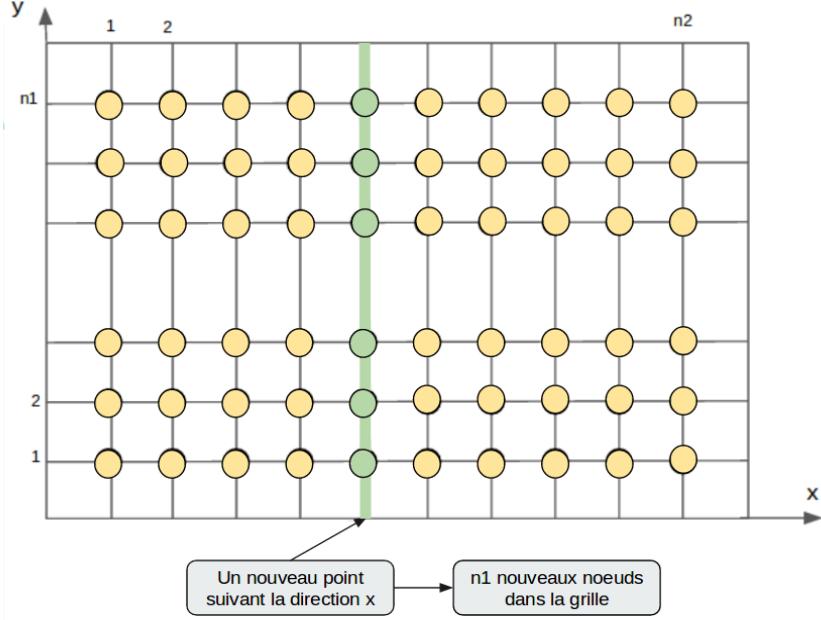


Figure 5: Influence de l'ajout d'un point d'interpolation sur une seule direction

Cette méthode d'interpolation devient très coûteuse lorsqu'il s'agit de fonctions multivariées 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:

$$\Delta_k = I_k - I_{k-1} \quad (3)$$

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,

$$I_n = \sum_{k=0}^n \Delta_k \quad (4)$$

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

$$h_k(y) = \prod_{j=0}^{k-1} \frac{(y - y_j)}{(y_k - y_j)}, \quad k > 0, \text{ et } h_0(y) = 1, \quad (5)$$

On a alors,

$$\Delta_k(f) = \alpha_k(f)h_k, \quad \alpha_k(f) = f(y_k) - I_{k-1}f(y_k) \quad (6)$$

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

$$I_n(f) = \sum_{k=0}^n \alpha_k(f) h_k \quad (7)$$

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

- Cas $d > 1$: On veut effectuer une interpolation polynomiale pour une séquence imbriquée d'ensemble $(\Lambda_n)_{n \geq 1}$ avec $n = \sharp(\Lambda_n)$.

Pour la suite, on définit :

- Le point multivarié \mathbf{y}_ν par $\mathbf{y}_\nu = (y_{\nu_j})_{j \geq 1} \in \mathcal{P}$
- La fonction polynomiale hiérarchique tensorisée \mathbf{H}_ν par $\mathbf{H}_\nu(y) = \prod_{j \geq 1} h_{\nu_j}(y_j)$

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.

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

$$\nu \in \Lambda \text{ et } \mu < \nu \Rightarrow \mu \in \Lambda \quad (8)$$

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

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

$$\begin{aligned} \Delta_{\nu^n} f &= \Delta_{\nu^n} f(\mathbf{y}_{\nu^n}) \mathbf{H}_{\nu^n} \\ &= (I_{\Lambda_n} f(\mathbf{y}_{\nu^n}) - I_{\Lambda_{n-1}} f(\mathbf{y}_{\nu^n})) \mathbf{H}_{\nu^n} \\ &= (f(\mathbf{y}_{\nu^n}) - I_{\Lambda_{n-1}} f(\mathbf{y}_{\nu^n})) \mathbf{H}_{\nu^n} \end{aligned} \quad (9)$$

Donc

$$I_{\Lambda_n} f = I_{\Lambda_{n-1}} f + (f(\mathbf{y}_{\nu^n}) - I_{\Lambda_{n-1}} f(\mathbf{y}_{\nu^n})) \mathbf{H}_{\nu^n} \quad (10)$$

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

$$I_{\Lambda_n} f = \sum_{k=0}^n f_{\nu^k} \mathbf{H}_{\nu^k} \quad (11)$$

avec f_{ν^k} définis récursivement par:

$$f_{\nu^1} = f(y_0), \quad f_{\nu^{k+1}} = f(\mathbf{y}_{\nu^{k+1}}) - I_{\Lambda_k} f(\mathbf{y}_{\nu^{k+1}}) = f(\mathbf{y}_{\nu^{k+1}}) - \sum_{i=1}^k f_{\nu^i} \mathbf{H}_{\nu^i}(\mathbf{y}_{\nu^{k+1}}) \quad (12)$$

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,

$$\mathcal{N}(\Lambda) = \{\nu \notin \Lambda : R_\nu \in \Lambda \cup \{\nu\}\}, R_\nu = \{\mu \in \Lambda : \mu < \nu\} \quad (13)$$

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:

$$\mathbb{L}_\Lambda = \sup_{f \in B(\mathcal{P})} \frac{\|I_\Lambda f\|_{L^\infty(\mathcal{P})}}{\|f\|_{L^\infty(\mathcal{P})}} \quad (14)$$

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 monovariées de Lebesgue

$$\lambda_k = \max_{f \in B(\mathcal{P})} \frac{\|I_k f\|_{L^\infty(\mathcal{P})}}{\|f\|_{L^\infty(\mathcal{P})}} \quad (15)$$

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

$$y_k = \operatorname{argmax}_{y \in P} \prod_{j=0}^{k-1} |y - y_j| \quad (16)$$

Cette séquence $(y_k)_{k \geq 0}$ est appellée 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 intéressant.

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.

Cette figure est divisé 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.

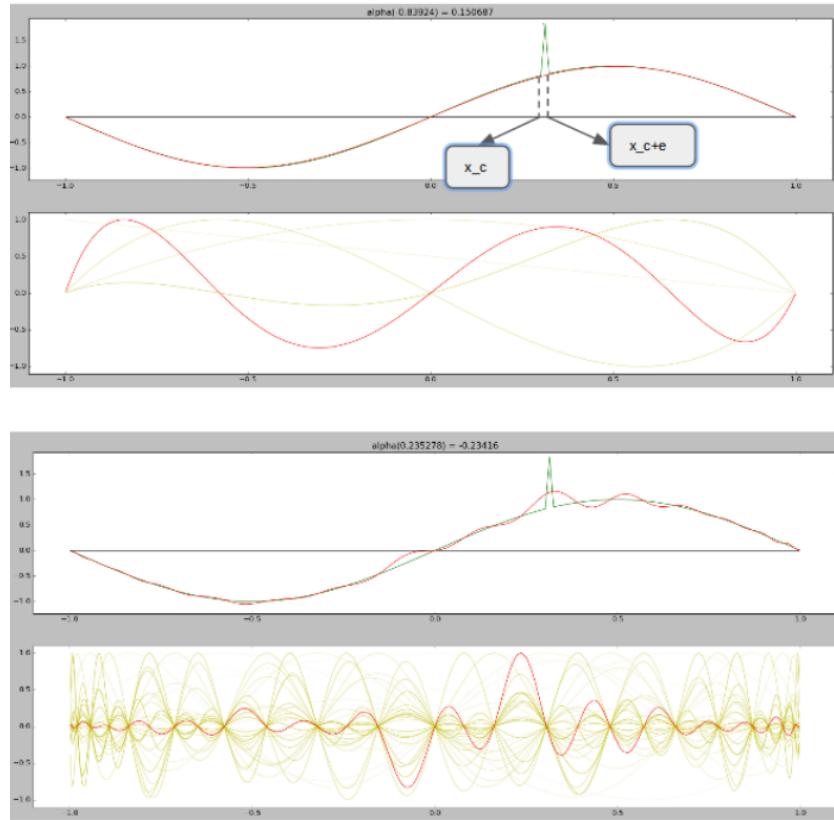


Figure 6: Interpolation utilisant des polynômes de Lagrange définis globalement

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 polynôiales 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 adéquat. La séquence sera formée par un ensemble de points obtenu en discrétilisant 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 .

- **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 forcement à l'ensemble des noeuds descendant directement des points de S dans l'arbre ci-dessous.

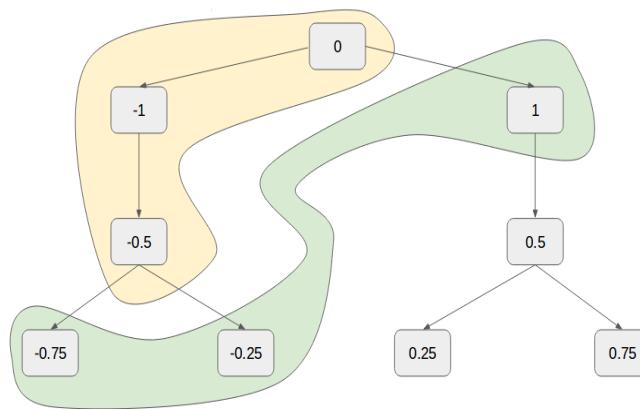


Figure 7: Arbre représentatif des points d'interpolation en 1D

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.

Etant 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. La figure ci-dessous, montre le résultat de l'interpolation d'une fonction de même type que précédemment, cette fois, en utilisant des fonctions quadratique 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.

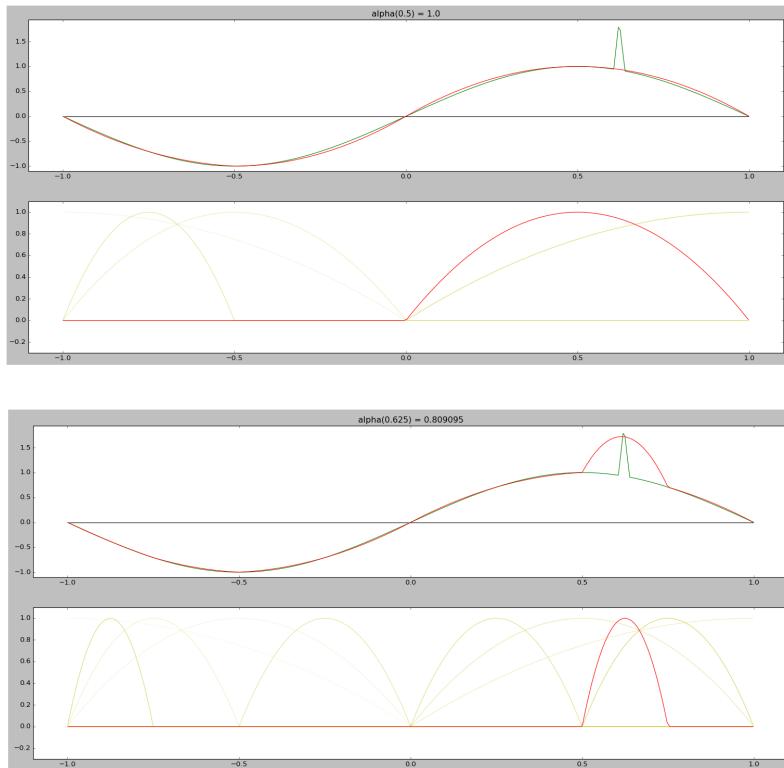


Figure 8: Interpolation utilisant des fonctions quadratiques par morceaux

Remarque:

- Certains nœuds ne peuvent 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 futur 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 évident 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 effet, considérons la fonction $f : (x, y) \rightarrow \sqrt{1 - x^2} \exp(y)$. La meilleure façon d'interpoler f est d'utiliser des fonctions quadratiques par morceaux selon la direction x et des polynômes de Lagrange suivant la direction y.

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).

Cette solution a été implémenté 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 implementés dans la classe *Interpolation < T >*. 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 similarité entre les deux méthodes, d'où le choix d'implémenter une classe abstraite, ensuite de faire deux classes filles qui héritent des fonctionnalités communes.
- **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 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 noeud possède au maximum 2 fils.
- Les points -1.0 et 1.0 (extrémités du P) possèdent chacun exactement un fils.

- Soit n un noeud intermédiaire, on note p son père, g son fils gauche, et d son fils droite, on a alors:

$$g = \begin{cases} -1 & \text{si } n = 0 \\ n - |p - n| / 2 & \text{si } n \neq 0 \end{cases} \quad (17)$$

$$d = \begin{cases} -1 & \text{si } n = 0 \\ n + |p - n| / 2 & \text{si } n \neq 0 \end{cases} \quad (18)$$

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 est ”0110”.

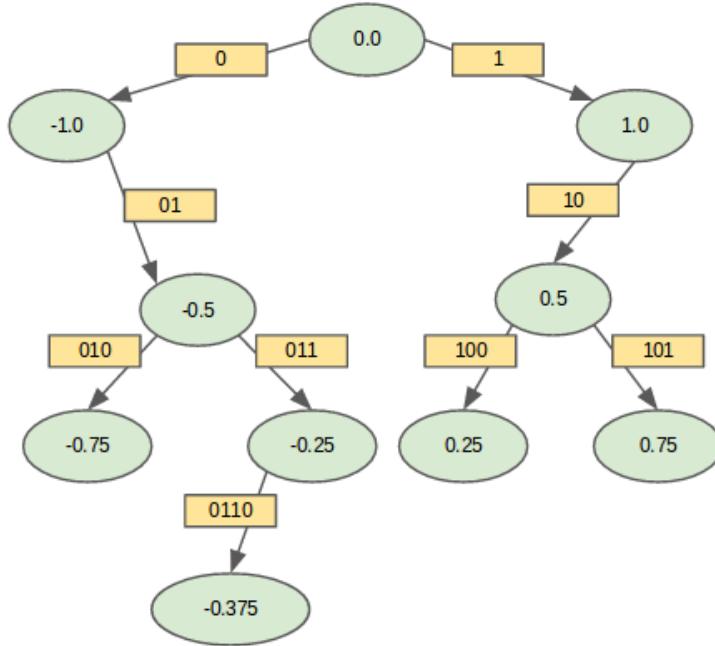


Figure 9: Codage de huffman de chaque nœud de l’arbre

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 nœuds ayant les valeurs les plus proches.

- **Classe MultivariatePoint<T>:** 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 polynômes de Lagrange.
- **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

$$\forall x \in U, |f(x) - L(x)| \leq \sup_{x \in U} \left(\prod_{i=1}^n |x - a_i| \right) \frac{\|f^{(n)}\|_\infty}{n!} \quad (19)$$

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 $a_i = \cos\left(\frac{(2i-1)\pi}{n}\right)$, $i = 1, \dots, n$.

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

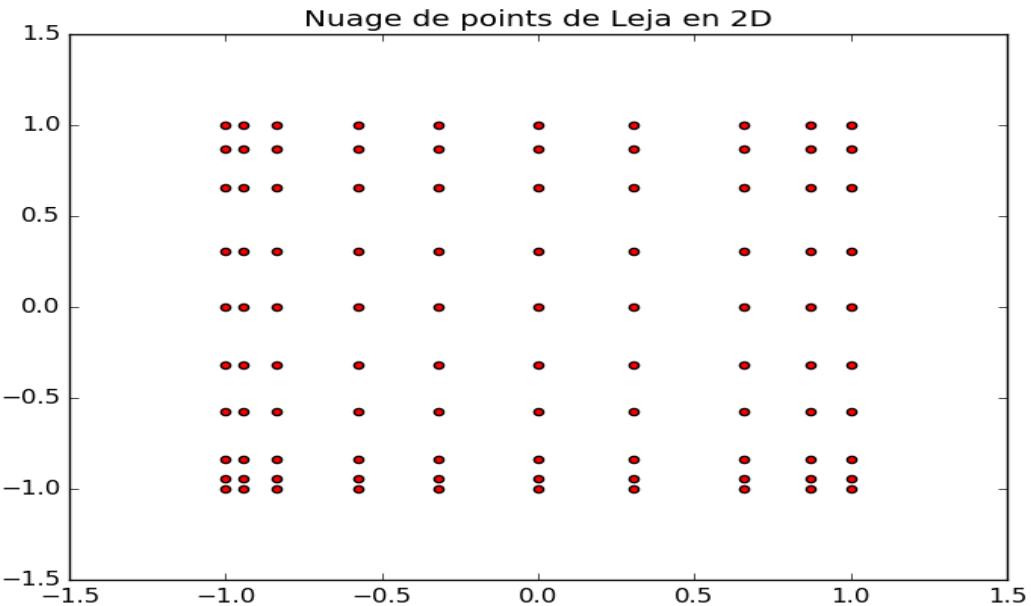


Figure 10: Grille 2D des points de Leja

Recherche du voisin courant: Une première méthode a été de rechercher le nouveau voisin parmi tous les points de la grille. Ce n'est pas évidemment la meilleure idée car la grille contient non seulement les voisins potentiels mais aussi les points déjà sélectionnés et traités. Donc, il est inutile de la parcourir en totalité à chaque étape car ça devient très couteûx.

Une meilleure stratégie a été adopté. En effet, on ne cherche plus le nouveau meilleur voisin parmi tous les points de la grille (100 si on est en dimension 2 et qu'on a choisi 10 points de discrétisation sur chaque direction), mais plutôt parmi un petit ensemble de points correspondant aux points voisins. (voir figure 7: à gauche la version première version et à droite, la seconde. La recherche du nouveau voisin s'effectue dans l'ensemble délimité par un contour bleu).

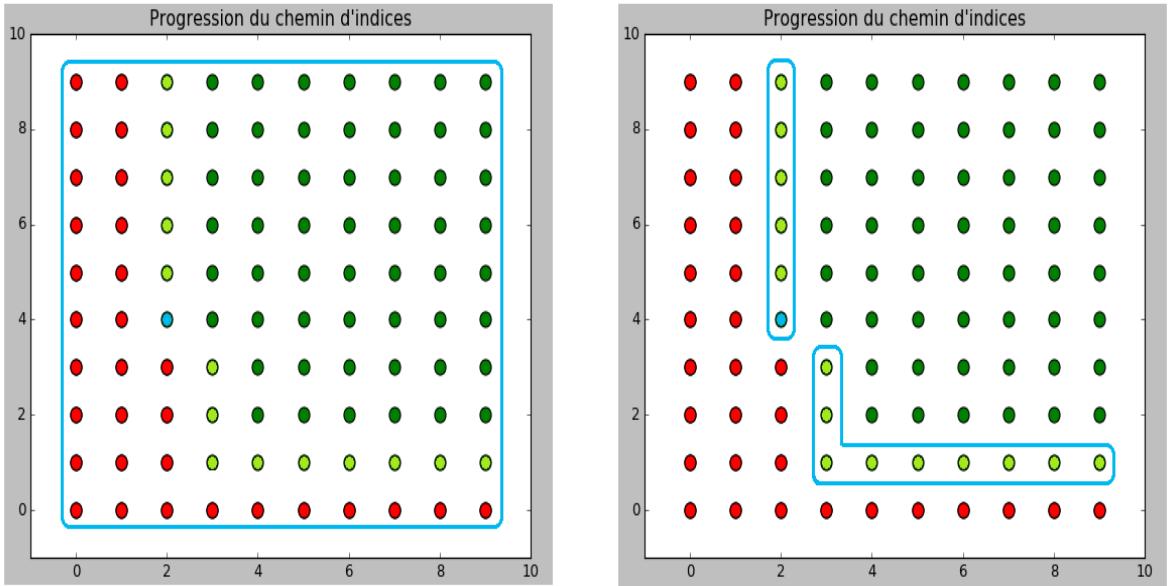


Figure 11: Recherche du voisin courant

Calcul des α_ν (ou f_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 multi-indices 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, couteûx.

Il est possible de procéder d’une manière plus astucieuse. En effet, on peut éviter cette redondance, tout simplement, en initialisant tous les f_ν avec une même valeur qu’on note f_{init} . Ensuite, à chaque fois qu’on a besoin de calculer un f_ν , on le compare d’abord à f_{init} ; si ils sont égaux alors on calcule f_ν (en utilisant (12)), sinon, on ne fait rien.

5 Évaluation de l’efficacité de la méthode

L’algorithme d’Interpolation Adaptative a été testé avec des fonctions analytiques définies à l’avance en des points choisis 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 les deux méthodes sur des données réelles fournies par le département R&D d’EDF et de les comparer.

5.1 Présentation du modèle de décomposition de Tucker

5.2 Tests et comparaisons

5.2.1 Fonctions analytiques prédéfinies

5.2.2 Données réelles

Le cœur d'un réacteur nucléaire posséde 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 appelée 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 appelée 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.

Afin de quantifier les taux des réactions des neutrons, on introduit une notation qui représente la probabilité d'interaction entre un neutron incident et un (des) noyau(x) cible(s). Il s'agit de la notion de sections efficaces en physique des particules.

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 physiques: *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.

| 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 |

Les données réelles fournies par le département R&D d'EDF sont regroupés 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, les vrais valeurs des sections efficaces en ces points (valeurs obtenus par le code 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 suivants: (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. Etant donné qu'on ne peut pas prévoir à l'avance les points où on veut 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 est bien évidemment trop couteux, surtout que le code APOLLO2 ne peut être appellé que sur une grille entière et non pas sur un seul point. On pourrait penser à sauvegarder les points d'interpolations 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.

Etant donné qu'on ne dispose pas du code APPOLLO2 on n'a pas en notre possession le code APPOLLO2, il n'est pas possible d'avoir les "vrais valeurs" des sections

efficaces à n'importe quel point Par contre, on peut avoir le résultat de l'approximation par la méthode de décomposition de Tucker. Donc, une bonne alternative serait de supposer que les valeurs obtenus par la méthode de Tucker correspondent aux vrais valeurs des sections efficaces.

Des tests ont été faits pour chaque type de section efficace. Les résultats obtenus ont été ensuite stocké dans des fichiers et organisées en 12 colonnes correspondants aux valeurs suivantes: **count bu tc density cb xe value_AI value_C value_T value_AI err_C err_T err_AI** où

- **value_AI**: valeurs de section efficace évaluées par la méthode d'Interpolation Adaptative.
- **err_AI**: erreurs relatives commises par la méthode d'Interpolation Adaptative.

Afin de pouvoir bien visualiser les résultats numériques obtenus, ces derniers ont été représenté par des graphes (un graphe par section efficace). Chaque graphe montre 3 nuages de points en $2D$ (l'abscisse de chaque point correspond à l'indice du point de la grille de référence (**count**), quant à l'ordonnée, il correspond à l'erreur relative obtenu en ce point).

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

Les différents graphes sont présentés en annexe.

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. Chaque une 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 nœud 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.

Tout ce qui précéde est décrit dans le tableau ci-dessous. Les informations sur la grille AI concerne la section efficace $\Sigma_t otale_t^0$ de type MOX.

| Direction | Grille Cocagne | Grille Tucker | Grille AI |
|----------------------------|-----------------------|--------------------------------|-----------|
| Burnup | 33 | 25 ($25 * 2^4 = 400$) | 35 |
| Température du combustible | 3 | 25 ($25 * 2^4 = 400$) | 5 |
| Densité du modérateur | 3 | 25 ($25 * 2^4 = 400$) | 13 |
| Concentration en Bore | 3 | 25 ($25 * 2^4 = 400$) | 156 |
| Niveau de xénon | 2 | 25 ($25 * 2^4 = 400$) | 5 |
| Total | $1782 = 33 * 3^3 * 2$ | $720 = 25 * 2^4 + 4 * 5 * 2^4$ | 1 |

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

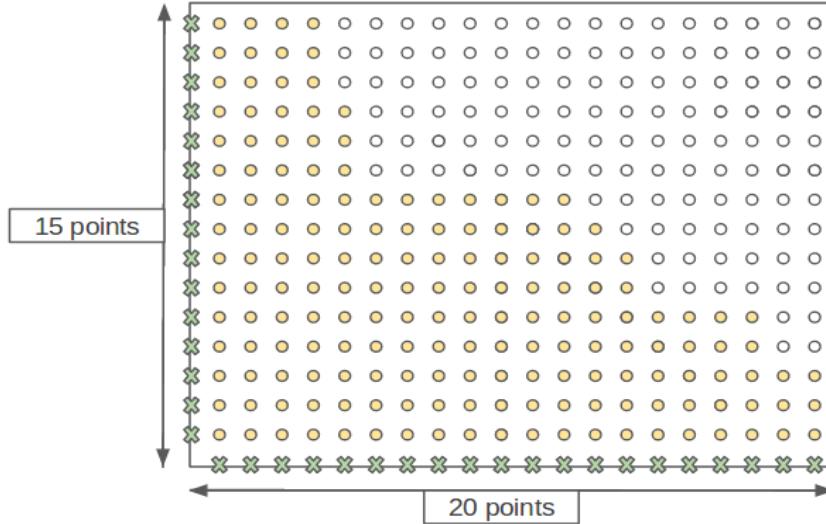


Figure 12: Exemple de points d'interpolation obtenu par l'algorithme AI

Erreurs d'approximation des sections efficaces: On compare la précision des différentes méthodes sur chaque noeud \mathbf{x}_i de la grille de référence. L'erreur d'approximation sur la grille de référence est définie soit par la norme infinie de l'erreuer relative, soit par la moyenne quadratique de l'erreur absolue (**Root Mean Square Error**):

$$e^{(inf)} = \max_{\mathbf{x}_i \in \text{reference grid}} \underbrace{\left| \frac{\tilde{f}(\mathbf{x}_i) - f(\mathbf{x}_i)}{\max |f(\mathbf{x}_i)|} * 10^5 \right|}_{\text{relative error}} \quad (20)$$

$$e^{(RMSE)} = \sqrt{\sum_{i=1}^N \frac{((f(\mathbf{x}_i) - \tilde{f}(\mathbf{x}_i)) * 10^5)^2}{N}} \text{ with } N = \sharp(\text{reference grid}) \quad (21)$$

où $f(\mathbf{x}_i)$ et $\tilde{f}(\mathbf{x}_i)$ sont respectivement la valeur exacte (calculée par APOLLO) 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}^239) absorbe des neutron 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])

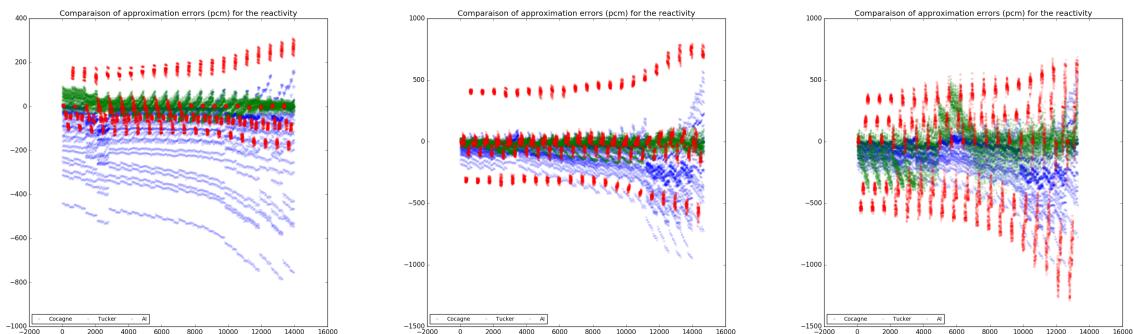
$$reactivity = 1 - \frac{1}{k_{eff}}, \text{ avec } k_{eff} = k_\infty = \frac{\nu \Sigma_f^1 * (\Sigma_t^2 - \Sigma_{s0}^{2 \rightarrow 2}) + \nu \Sigma_f^2 * \Sigma_{s0}^{1 \rightarrow 2}}{(\Sigma_t^1 - \Sigma_{s0}^{1 \rightarrow 1}) * (\Sigma_t^2 - \Sigma_{s0}^{2 \rightarrow 2}) - \Sigma_{s0}^{1 \rightarrow 2} * \Sigma_{s0}^{2 \rightarrow 1}} \quad (22)$$

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

$$\mathbf{e}_{reactivity}(inf) = \max_{\mathbf{x}_i \in \text{reference grid}} \left| \left(\frac{1}{k_\infty(\mathbf{x}_i)} - \frac{1}{\tilde{k}_\infty(\mathbf{x}_i)} \right) * 10^5 \right| \quad (23)$$

Les figurent ci-dessous montrent les erreurs de réactivité obtenues avec chaque méthode et pour chaque type de réacteur (MOX, UOX, UOX-Gd):

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



Comparaison du nombre de points d'évaluation:

| Interpolation Multilinéaire | Décomposition de Tucker | Interpolation Adaptative |
|--------------------------------------------------------------------------------------------------------|-------------------------|--------------------------|
| $\underbrace{720}_{\text{for 5 Tucker grids}} + \underbrace{378}_{\text{for evaluated points}} = 1098$ | 1782 | 780 |

Comparaison du stockage:

Comparaison de la précision:

5.3 Conclusion et remarques

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 jour.

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.

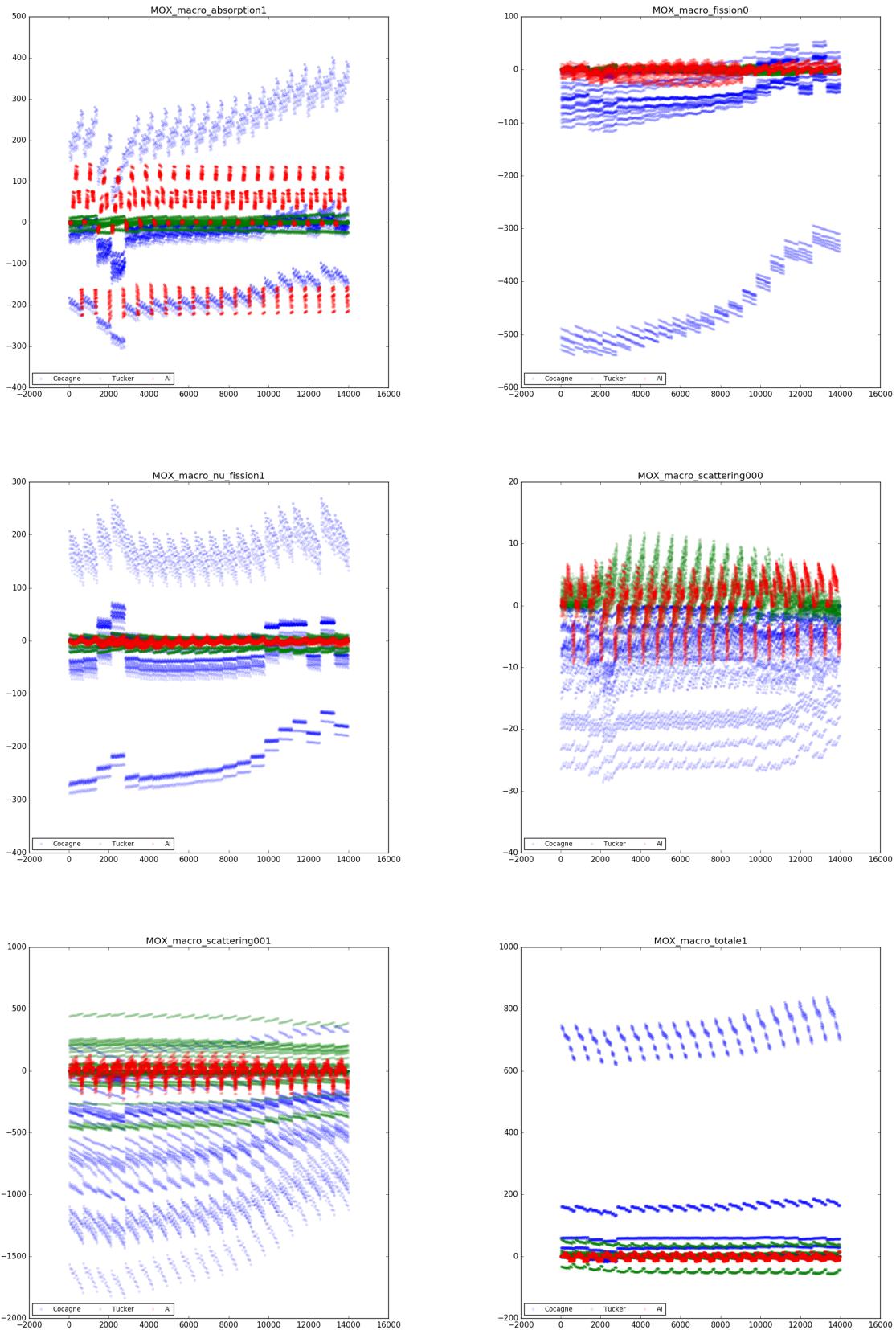


Figure 13: Comparaison des erreurs relatives (pcm) pour les sections efficaces MOX

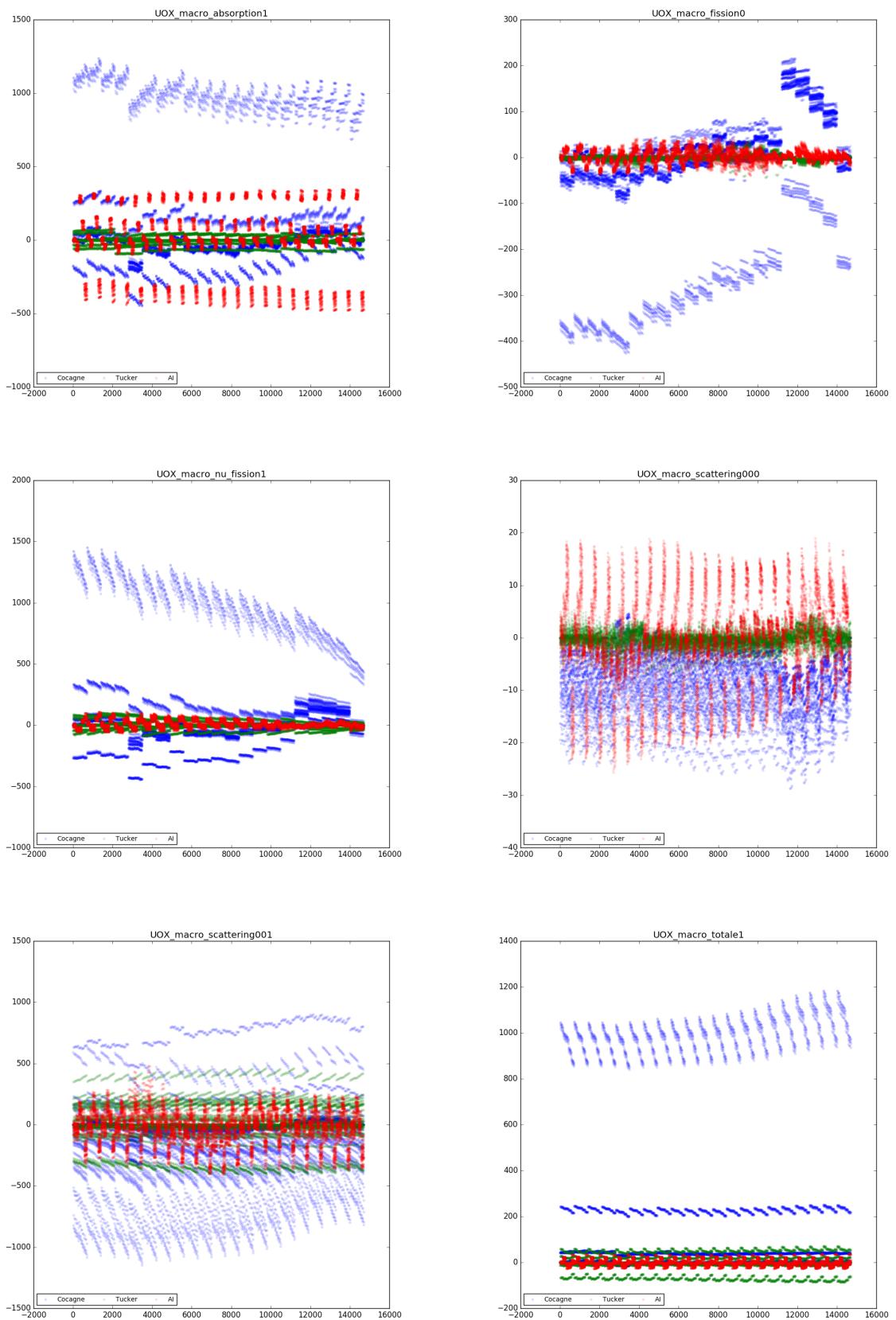


Figure 14: Comparaison des erreurs relatives (pcm) pour les sections efficaces UO₂

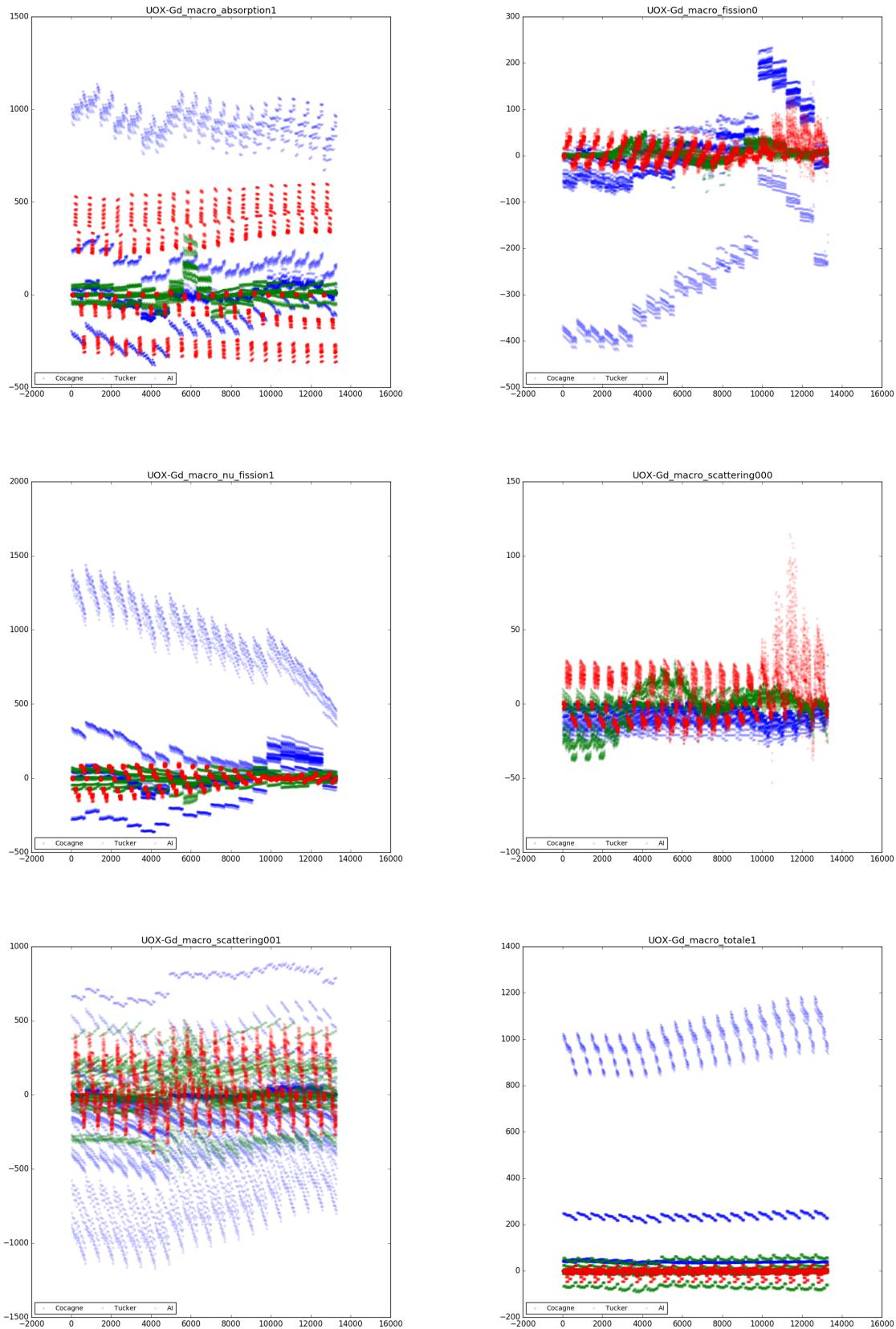


Figure 15: Comparaison des erreurs relatives (pcm) pour les sections efficaces UOX-Gd