Improvement of cross section model in COCAGNE code of the calculation chain of EDF

Thi Hieu Luu

▶ **To cite this version:**

Thi Hieu Luu. Improvement of cross section model in COCAGNE code of the calculation chain of EDF. Mathematical Physics [math-ph]. Université Pierre et Marie Curie - Paris VI, 2017. English. NNT : 2017PA066120 . tel-01630268

HAL Id: tel-01630268

https://theses.hal.science/tel-01630268

Submitted on 7 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Thèse de doctorat de l'Université Pierre et Marie Curie

Spécialité

Mathématiques Appliquées

présentée par

Thi Hieu LUU

pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Amélioration du modèle de sections efficaces dans le code de cœur COCAGNE de la chaîne de calculs d'EDF

Soutenue le 17/02/2017

Directeur de thèse :	M. Yvon MADAY	Professeur, Université Pierre et Marie Curie
Encadrants de thèse :	M. Matthieu GUILLO	Ingénieur - Chercheur, EDF - R&D
	M. Pierre GUÉRIN	Ingénieur - Chercheur, EDF - R&D
Rapporteurs :	M. Anthony NOUY	Professeur, École Centrale de Nantes
	M. Lars GRASEDYCK	Professeur, RWTH Aachen University
Examinateurs :	M. Albert COHEN	Professeur, Université Pierre et Marie Curie
	M. Daniele TOMATIS	Chercheur, CEA
	M. Andrea ZOIA	Chercheur, CEA

Thèse effectuée aux :

Laboratoire Jacques-Louis Lions, UMR 7598

Adresse géographique :

Laboratoire Jacques Louis Lions

3ème étage, tour 15-16, 15-25, 16-26

4 place Jussieu

75005 Paris, France

+33 (0)1 44 27 42 98 (Tél.)

Adresse postale :

Laboratoire Jacques-Louis Lions

Université Pierre et Marie Curie

Boîte courrier 187

75252 Paris Cedex 05 France

Département SINETICS, EDF - R&D

Adresse :

Département SINETICS

EDF Lab Paris - Saclay

7, Boulevard Gaspard Monge

92120 Palaiseau, France

Résumé

Afin d'exploiter au mieux son parc nucléaire, la R&D d'EDF est en train de développer une nouvelle chaîne de calcul pour simuler le cœur des réacteurs nucléaires avec des outils à l'état de l'art. Ces calculs nécessitent une grande quantité de données physiques, en particulier les sections efficaces.

Dans la simulation d'un cœur complet, le nombre de valeurs des sections efficaces est de l'ordre de plusieurs milliards. Ces sections efficaces peuvent être représentées comme des fonctions multivariées dépendant de plusieurs paramètres physiques. La détermination des sections efficaces étant un calcul complexe et long, nous pouvons donc les précalculer en certaines valeurs des paramètres (caluls hors ligne) puis les évaluer en tous points par une interpolation (calculs en ligne). Ce processus demande un modèle de reconstruction des sections efficaces entre les deux étapes.

Pour réaliser une simulation plus fidèle du cœur dans la nouvelle chaîne d'EDF, les sections efficaces nécessitent d'être mieux représentées en prenant en compte de nouveaux paramètres. Par ailleurs, la nouvelle chaîne se doit d'être en mesure de calculer le réacteur dans des situations plus larges qu'actuellement. Le modèle d'interpolation multilinéaire pour reconstruire les sections efficaces est celui actuellement utilisé pour répondre à ces objectifs. Néanmoins, 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. Par conséquence, le nombre et le temps des calculs hors ligne ainsi que la taille du stockage des données deviennent problématique.

L'objectif de cette thèse est donc de trouver un nouveau modèle pour répondre aux demandes suivantes : (i)-(hors ligne) réduire le nombre de précalculs, (ii)-(hors ligne) réduire le stockage de données pour la reconstruction et (iii)-(en ligne) tout en conservant (ou améliorant) la précision obtenue par l'interpolation multilinéaire.

D'un point de vue mathématique, ce problème consiste à approcher des fonctions multivariées à partir de leurs valeurs précalculées. Nous nous sommes basés sur le format de Tucker - une *approximation de tenseurs de faible rang* afin de proposer un nouveau modèle appelé *la décomposition de Tucker*. Avec ce modèle, une fonction multivariée est approchée par une combinaison linéaire de produits tensoriels de fonctions d'une variable. Ces fonctions d'une variable sont construites grâce à une technique dite de *décomposition en valeurs singulières d'ordre supérieur* (une "matricization" combinée à une extension de la décomposition de Karhunen-Loève). L'algorithme dit *glouton* est utilisé pour constituer les points liés à la résolution des coefficients dans la combinaison de la décomposition de Tucker.

Les résultats obtenus montrent que notre modèle satisfait les critères exigés sur la réduction de données ainsi que sur la précision. Avec ce modèle, nous pouvons aussi éliminer *a posteriori* et *a priori* les coefficients dans la décomposition de Tucker. Cela nous permet de réduire encore le stockage de données dans les étapes hors ligne sans réduire significativement la précision.

Mots-clés : sections efficaces, décomposition de Tucker, approximation de tenseurs de faible rang, décomposition en valeurs singulières d'ordre supérieur, algorithme glouton, neutronique, réduction de modèle, sparse grids

Abstract

In order to optimize the operation of its nuclear power plants, the EDF's R&D department is currently developing a new calculation chain to simulate the nuclear reactors core with state of the art tools. These calculations require a large amount of physical data, especially the cross-sections.

In the full core simulation, the number of cross-section values is of the order of several billions. These cross-sections can be represented as multivariate functions depending on several physical parameters. The determination of cross-sections is a long and complex calculation, we can therefore pre-compute them in some values of parameters (offline calculations), then evaluate them at all desired points by an interpolation (online calculations). This process requires a model of cross-section reconstruction between the two steps.

In order to perform a more faithful core simulation in the new EDF's chain, the cross-sections need to be better represented by taking into account new parameters. Moreover, the new chain must be able to calculate the reactor in more extensive situations than the current one. The multilinear interpolation is currently used to reconstruct cross-sections and to meet these goals. However, with this model, the number of points in its discretization increases exponentially as a function of the number of parameters, or significantly when adding points to one of the axes. Consequently, the number and time of offline calculations as well as the storage size for this data become problematic.

The goal of this thesis is therefore to find a new model in order to respond to the following requirements: (i)-(offline) reduce the number of pre-calculations, (ii)-(offline) reduce stored data size for the reconstruction and (iii)-(online) maintain (or improve) the accuracy obtained by multilinear interpolation.

From a mathematical point of view, this problem involves approaching multivariate functions from their pre-calculated values. We based our research on the Tucker format - *a low-rank tensor approximation* in order to propose a new model called *the Tucker decomposition*. With this model, a multivariate function is approximated by a linear combination of tensor products of one-variate functions. These one-variate functions are constructed by a technique called *higher-order singular values decomposition* (a "matricization" combined with an extension of the Karhunen-Loeve decomposition). The so-called *greedy* algorithm is used to constitute the points related to the resolution of the coefficients in the combination of the Tucker decomposition.

The results obtained show that our model satisfies the criteria required for the reduction of the data as well as the accuracy. With this model, we can eliminate *a posteriori* and *a priori* the coefficients in the Tucker decomposition in order to further reduce the data storage in offline steps but without reducing significantly the accuracy.

Keywords: cross-sections, Tucker decomposition, low-rank tensor approximation , higher-order singular value decomposition, greedy algorithm, neutronics, reduced model, sparse grids

Acknowledgements

Time always goes by so fast and the moment has come after three years for this thesis to end. I am writing these lines to express my acknowledgment to the people who helped me and accompanied me throughout of this memorable period. Without their support, my work could not have been accomplished.

First, let me express my deepest gratitude to my supervisor, Professor Yvon MADAY, at the Laboratoire Jacques Louis Lions (LJLL). Yvon, you are the person who led me to this thesis (by chance) and you are also the person who guided me through the important progress in my work. Without your patience, your enthusiasm and above all, your extensive scientific knowledge, I would not have got the results I have today. Although very busy, Yvon still reserved his precious time to follow my work and to help me when I needed it. Thank you so much for what you did for me.

The most sincerest and deepest thanks I also give to my two industrial supervisors: Matthieu GUILLO and Pierre GUÉRIN, at EDF - R&D. I was very lucky to have two very competent industrial supervisors in two complementary domains: Matthieu for neutronic physics and Pierre for mathematics. Thank you both for trusting me in this work despite all my defects that you knew. Matthieu and Pierre, you have both always been my references for any technical and administrative problem. Your guidance and your advice have helped me enormously in the organization of my work and in the progress of this thesis. Your availability, your patience and your time reserved for my work, all are immeasurable.

I would like to thank all the members of the jury: Professor Anthony NOUY and Professor Lars GRASEDYCK for having accepted to be the reviewers of this thesis, for having conscientiously read my work and having spent your time to write the constructive reports. My sincere thanks also goes to Professor Albert COHEN, researcher Daniele TOMATIS and researcher Andrea ZOIA for your examination as well as your interest in my work.

This research was done in EDF's SINETICS department and the LJLL laboratory of Université Pierre et Marie Curie. I would like to thank these two establishments for hosting me during the entire research period. In particular, I would like to express my gratitude to the group leader of I27 at EDF: Philippe MAGAT, and the head of the SIMUCOEUR2 project, Coline BROSSELARD.

Thank you very much for facilitating my working conditions and allowing me to participate in many conferences as well as training courses.

A very big thanks is expressed to Nadine SCHWARTZ who has helped me a lot with the installation and the utilization of GAB. It would have been hard to accomplish this work without your guidance. Nadine also provided me with many documents on her research and methods related to the problem in my thesis, which gave me a better overview of my work.

I would also like to thank Serge MARGUET for your enthusiasm and patience in enlightening me with knowledge about neutron equations. Also, many thanks to Angelique PONCOT for your documents on Areva's model.

Special thanks to Ansar CALLOO for your discussions as well as your advice about writing the first journal paper in a PhD thesis. Thanks to Nathalie GULER for the times you picked me up to go to work. Thanks to Hélène MONDAIN for your interest in Vietnam, for your kindness. Thanks to Florian HAURAIS for sharing with me our office.

And sincere thanks to the two groups I27, I28, who always created a friendly environment full of humor with many delicious homemade cakes and picnic meetings.

I now think about my husband, Thanh-Hà and my two sons Hugo and Louis. A lot of emotion when I write about my small family. A person once asked me how I can do this thesis with my two little sons beside me? It would not be true if I said that there was no difficulty. But above all, my husband and my two sons are the ones who are always beside me, encourage me, balance my emotions. My husband is not only the one who shares with me all the troubles in my life but also the one who guides me with patience on scientific research. There is no word to express the importance of my family in my heart.

Finally, a huge thanks and deep gratitude to all members of my family in Vietnam. Thanks to my parents who always supported me before, during and even after this thesis. The presence of my parents-in-law, mẹ Thanh and bố Dân, at this moment with their help in my daily work is unmeasurable for this final stage. The support of "em Lân" is also very important so the word "thank you" is never enough. For my sisters: Hằng, Nga, Tha'o, this PhD thesis is a gift I dedicate to you.

Contents

Introduction (Version française)

Motivation

EDF est le premier électricien du monde, grâce notamment à son parc français composé de 58 réacteurs nucléaires à eau pressurisée (REP). Afin de piloter et contrôler le fonctionnement de ces réacteurs, EDF a développé des chaînes de calcul qui simulent le comportement du cœur d'un réacteur. Face aux demandes de plus en plus exigeantes de l'ingénierie en termes de temps de calcul et de précision, une nouvelle chaîne de calcul de cœur, appelée ANDROMEDE, est en cours de développement à EDF-R&D.

L'un des éléments de la chain est le code neutronique, appelé COCAGNE, dont l'un des objectifs est de résoudre numériquement l'équation du transport neutronique (ou l'une de ses approximations), qui nous permet d'obtenir des grandeurs physiques d'intérêt pour décrire le comportement du réacteur, telles que : le flux neutronique, le facteur de multiplication effectif, la réactivité... La résolution de cette équation a besoin d'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, tels que : température du combustible, densité du modérateur, 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*. Le nombre de valeurs des sections efficaces dans la simulation d'un cœur complet est de l'ordre de plusieurs milliards à cause de la discrétisation du cœur en des centaines de milliers de cellules et les valeurs des sections efficaces sont différentes dans chaque cellule.

Afin de simuler le cœur d'un réacteur dans un temps de calcul raisonnable, un schéma de calcul industriel comprenant les deux étapes suivantes est utilisé:

— Calculs hors ligne : les sections efficaces sont précalculées par un code dit *code réseau* sur

des points fixes et présélectionnés dans l'espace de phase des paramètres. L'équation du transport neutronique est résolue de manière précise grâce à des discrétisations spatiales très fines sur des motifs plus petits que le cœur (typiquement un assemblage combustible) et une discrétisation en énergétique elle-même très fine (plusieurs centaines de groupes). Le flux neutronique sorti de cette résolution est utilisé pour l'homonégisation spatiale et la condensation énergétique des sections efficaces. L'information obtenue sur les sections efficaces est stockée dans des fichiers, appelés les *bibliothèques neutroniques*.

— Calculs en ligne : les sections efficaces sont évaluées en n'importe quel point du cœur par un *code de cœur* et grâce à une méthode d'évaluation basée sur l'information stockée dans les bibliothèques neutroniques. Ces valeurs sont utilisées comme données pour la résolution de l'équation du transport neutronique au niveau du cœur du réacteur. La résolution est simplifiée sur la géométrie complète, par exemple en utilisant une approximation de diffusion. Les discrétisations du cœur sont beaucoup plus grossière que celle employée par le code réseau.

Ce schéma montre qu'entre les deux étapes, un modèle de reconstruction des sections efficaces est nécessaire pour évaluer les sections efficaces à partir de données précalculées et stockées dans les bibliothèques neutroniques.

Dans la nouvelle chaîne de calcul ANDROMEDE, le code APOLLO2 développé au CEA est utilisé comme code réseau et le code COCAGNE développé à EDF-R&D est utilisé comme code de cœur. Actuellement, l'interpolation multilinéaire est utilisée pour la reconstruction des sections efficaces dans le code COCAGNE.

Pour le modèle d'interpolation multilinéaire, les sections efficaces sont pré-calculées sur une grille tensorielle à partir de valeurs discrétisées sur les axes des paramètres. Toutes les valeurs pré-calculées par APOLLO2 sur cette grille sont ensuite stockées directement dans les bibliothèques neutroniques. Avec ce modèle, le nombre de calculs APOLLO2 ainsi que la taille des bibliothèques neutroniques sont de l'ordre de $\mathcal{O}(n^d)$ si on suppose que les sections efficaces dépendent de d paramètres et que chaque paramètre est discrétisé par n points sur l'axe correspondant. Avec les contraintes industrielles imposées à EDF, ce modèle devient trop 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 du modèle actuel, l'objectif de cette thèse est de développer un nouveau modèle de reconstruction des sections efficaces pour répondre aux exigences suivantes :

(i) Calculs hors ligne : utiliser moins de pré-calculs effectués par le code réseau APOLLO2,

(ii) Calculs hors ligne : stocker moins de données pour la reconstruction des sections efficaces et,

(iii) Calculs en ligne : avoir une bonne précision (de l'ordre du pcm (10^{-5})) pour l'évaluation des sections efficaces.

Une autre contrainte est qu'un calcul APOLLO2 en un point donné (de l'espace de phase des paramètres) fournit toutes les valeurs des sections efficaces en même temps. La raison est que toutes ces sections efficaces dépendent du même flux neutronique calculé par APOLLO2 et ce flux est coûteux en temps de calcul. Par conséquence, la reconstruction de chaque section efficace doit être optimisée avec les autres afin de limiter le nombre de calculs APOLLO2.

Face à ce problème, nous avons étudié et proposé un modèle appelé la *décomposition de Tucker*. Ce modèle est basé sur le format de Tucker - une approximation de tenseur de faible rang. Pour *une section efficace représentée comme une fonction multivariée*, la décomposition de Tucker se réalise comme suit:

(i) Calculs hors ligne :
- Nous construisons dans chaque direction, des fonctions d'une variable dites *fonctions de base tensorielle directionnelle* grâce à des techniques comme la *"matricization"* et *la décomposition de Karhunen-Loève*. Ces techniques sont utilisées pour acquérir et reconstruire, direction par direction, l'information à partir de valeurs des sections efficaces précalculées par le code APOLLO2. Les fonctions de base tensorielle directionnelle sont parfois appelées simplement *fonctions de base*. En pratique, ces fonctions sont construites à partir de vecteurs propres de la décomposition de Karhunen-Loève.
- La section efficace considérée est approchée par une combinaison linéaire des produits tensoriels des fonctions de base.
- Les coefficients de la combinaison dans la décomposition de Tucker sont déterminés par un système d'équations linéaires. Ce système dépend de points dans l'espace de phase des paramètres (sur ces points nous devons utiliser les calculs APOLLO2). Dans notre travail, nous proposons de choisir ces points par une technique basée sur un algorithme dit "*glouton*" qui sera détaillée dans ce manuscrit.

(ii) Calculs hors ligne : Nous stockons dans la bibliothèque neutronique les vecteurs qui représentent les valeurs des fonctions de base tensorielle directionnelle aux points de calcul présélectionnés et les coefficients dans la décomposition de Karhunen-Loève.

(iii) Calculs en ligne : Nous utilisons l'interpolation de Lagrange pour évaluer les fonctions de base en un point donné et finalement, évaluer les sections efficaces en n'importe quel point de l'espace de phase des paramètres par la décomposition de Tucker.

Nous avons testé et validé notre modèle sur deux domaines de calcul : le domaine dit *standard* (les valeurs des paramètres sont proches d'un point particulier dit *point nominal*, qui détermine le fonctionnement nominal du réacteur) et le domaine dit *étendu* où les domaines de définition des

paramètres sont éloignés par rapport au cas standard. Les résultats obtenus montrent la performance de notre méthode : alors que le nombre de calculs APOLLO2 et le stockage dans la bibliothèque neutronique sont significativement réduits dans les étapes hors lignes, la précision de notre modèle reste meilleure ou égale à l'interpolation multilinéaire dans l'étape en ligne. Le creusage (a posteriori et a priori) pour les coefficients dans la décomposition de Tucker permet de réduire encore le stockage, ainsi que le temps de calculs en ligne, tout en gardant une précision similaire à notre approche initiale.

Présentation des chapitres

Chapitre 1

Ce chapitre présente succinctement les concepts neutroniques (réactions en chaîne, sections efficaces, flux neutronique, équation du transport des neutrons,...) ainsi que le contexte de la simulation (structure multi-échelles du cœur d'un réacteur, schéma de calcul en deux étapes,...). L'objectif principal est d'aider le lecteur à mieux comprendre le contexte de notre travail.

Chapitre 2

Ce chapitre est dédié à décrire différents formats de tenseur de faible rang, tel que : format en r-termes, format en sous-espace tensoriel (le format de Tucker), format hiérarchique. Nous expliquons ensuite pourquoi nous avons choisi la décomposition de Tucker (basée sur le format de Tucker) pour le problème de reconstruction des sections efficaces.

Chapitre 3

Ce chapitre, basé sur notre premier article, détaille la méthode que nous proposons : la décomposition de Tucker. Les sections efficaces sont approchées par une combinaison linéaire des produits tensoriels des fonctions de base tensorielle directionnelle. Nous décrivons étape par étape les problèmes suivants: la construction des fonctions de base tensorielle directionnelle (par une extension de la décomposition de Karhunen-Loève), la détermination des coefficients dans la combinaison (par un système d'équations linéaires en utilisant l'algorithme glouton). Un premier benchmark est aussi proposé pour comparer notre modèle avec l'interpolation multilinéaire.

Chapitre 4

Dans ce chapitre, basé sur notre deuxième article, nous proposons des benchmarks physiques sur les deux domaines de calcul : standard et étendu. Des analyses statistiques sont proposées qui nous permettent de déterminer les facteurs principaux pour améliorer la précision de la reconstruction

: augmenter le nombre de fonctions de base tensorielle directionnelle ou discrétiser plus finement certaines zones de valeurs des paramètres trouvées par l'analyse.

Chapitre 5

Ce chapitre, basé sur notre troisième article, présente la possibilité de creuser les coefficients dans la décomposition de Tucker. Deux techniques sont proposées : le creusage *a posteriori* et le creusage *a priori*. Le creusage *a posteriori* est effectué quand les coefficients sont déjà calculés et nous pouvons donc éliminer les plus petits coefficients. Le creusage a priori est basé sur une prédiction de valeurs des coefficients grâce à un ordre proposé pour les fonctions de base tensorielle directionnelle. Ces méthodes nous permettent de réduire significativement le nombre de coefficients pour une précision équivalente et ouvrent une possibilité de réduire le nombre de calculs APOLLO2.

Chapitre Conclusion

Ce chapitre est réservé à la conclusion et aux perspectives de notre modèle.

Introduction (English version)

Motivation

EDF is the first electricity utility in the world, thanks in particular to its nuclear power plants in France with 58 pressurized water reactors (PWR). In order to pilot and control the operation of these reactors, EDF has developed calculation chains that simulate the behavior of a reactor core. To face more and more exigent engineering demands in terms of calculation time and accuracy, a new core calculation chain, named ANDROMEDE, is being developed at EDF-R&D.

One of the elements of the chain is the neutron code, called COCAGNE, one goal of which is to solve numerically the neutron transport equation (or one of its approximations), which allows us to obtain physical quantities of interest to describe the behavior of the reactor, such as: the neutron flux, the effective multiplication factor, reactivity... The resolution of this equation requires a large amount of physical data, especially cross-sections.

In neutronics, cross-sections represent the interaction probability of an incident neutron with target nuclei, for different types of interaction. In a neutron simulation, the cross-sections can be represented as functions depending on several physical parameters. These parameters are used to describe the thermo-hydraulic conditions and the configuration of the reactor core, such as: fuel temperature, moderator density, boron concentration, xenon level, burnup,... Thus, the cross-sections are multivariate functions defined on a space called *the parameters-phase space*. The number of cross-section values in the simulation of a full core is in the order of several billion due to the core discretization in hundreds of thousands of cells, with different cross-section values in each cell.

In order to simulate the reactor core in a reasonable calculation time, an industrial calculation scheme consisting in the two following steps is used:

— Offline calculations: the cross-sections are pre-calculated by a code named *lattice code* on fixed and preselected points in the parameters-phase space. The neutron transport equation is solved precisely through very fine spatial discretizations on smaller patterns than the reactor core (typically a fuel assembly) and an energy discretization itself very fine (hundreds of groups). The neutron flux which is the output of this resolution is used for spatial

homogenization and energy condensation of cross-sections. The obtained information on cross-sections is stored in files, called *neutron libraries*.

— Online calculations: the cross-sections are evaluated at any point of the core by a *core code* and through an evaluation method based on the information stored in neutron libraries. These values are used as inputs for the resolution of the neutron transport equation at the reactor core level. The resolution is simplified on the full geometry, for example by using an approximation of diffusion. The core discretizations are much coarser than those used by the lattice code.

This scheme shows that between the two steps, a model of cross-section reconstruction is required to evaluate the cross-sections from pre-calculated and stored data in the neutron libraries. In the new calculation chain ANDROMEDE, the code APOLLO2 developed at CEA is used as lattice code and the core code COCAGNE developed at EDF-R&D is used as core code. Currently, multilinear interpolation is used for the reconstruction of cross-sections in the code COCAGNE .

For the multilinear interpolation model, cross-sections are pre-calculated on a tensorized grid from discretized values on the parameters axes. All pre-calculated values by APOLLO2 on this grid are then stored directly in neutron libraries. With this model, the number of APOLLO2 calculations and the size of neutron libraries are in the order of $\mathcal{O}(n^d)$ if we assume that the cross-sections depend on d parameters and that each parameter is discretized by n points on the corresponding axis. With the industrial constraints imposed at EDF, this model becomes too expensive in memory and computation time for complex situations due to the rapid and exponential increase of calculation points. Such situations can happen, for example, in the incidental case where new parameters are added (water temperature, water blades) and/or when the calculation domain extends (with larger definition domains for the parameters).

In order to overcome the limitations of the current model, the aim of this thesis is to develop a new model of cross-sections reconstruction to respond to the following requirements:

(i) Offline calculations: use fewer pre-calculations performed by the lattice code APOLLO2.

(ii) Offline calculations: store less data for the reconstruction of cross-sections and,

(iii) Online calculations: get a high accuracy (in the order of pcm (10^{-5})) for cross-section evaluation.

Another constraint is that an APOLLO2 calculation at a given point (in the parameters-phase space) provides all the values of the cross-sections at the same time. The reason is that all these cross-sections depend on the same neutron flux computed by APOLLO2 and this flux is a time-consuming calculation. Therefore, the reconstruction of each cross-section must be optimized with the others in order to limit the number of APOLLO2 calculations.

To deal with this problem, we studied and proposed a model called *the Tucker decomposition*.

This is based on the Tucker format - a low-rank tensor approximation. For *a cross-section represented as a multivariate function*, Tucker decomposition is performed as follows:

(i) Offline calculations:

— We construct in each direction, the one-variate functions called *tensor directional basis functions* through techniques like the "*matricization*" and *the Karhunen-Loève decomposition*. These techniques are used to acquire and reconstruct, direction by direction, the information from cross-section values pre-calculated by APOLLO2. The tensor directional basis functions are sometimes simply called *basis functions*. In practice, these functions are constructed from eigenvectors of the Karhunen-Loève decomposition.

— The considered cross-section is approximated by a linear combination of tensor products of basis functions.

— The coefficients of the combination in the Tucker decomposition are determined by a system of linear equations. This system depends on points in the parameters-phase space (on these points we need to use APOLLO2 calculations). In our work, we propose to select these points with a technique based on an algorithm called "*greedy*" which will be detailed in this manuscript.

(ii) Offline calculations: We store in neutron libraries the vectors that represent the values of tensor directional basis functions at pre-selected calculation points, and the coefficients in the Karhunen-Loève decomposition.

(iii) Online calculations: We use the Lagrange interpolation to evaluate basis functions at a given point and finally, evaluate cross-sections at any point in the parameters-phase space by the Tucker decomposition.

We have tested and validated our model on two calculation domains: the domain named *standard* (parameters values are close to a particular point called *nominal point*, which determines the nominal operation of the reactor) and the domain called *extended* where definition domains of parameters are far from the standard case. The results obtained show the performance of our method: while the number of APOLLO2 calculations and storage in the neutron libraries are significantly reduced in the offline steps, the accuracy of our model is better or equal to multilinear interpolation in the online step. The sparse representation (a posteriori and a priori) for the coefficients in the Tucker decomposition allows us to further reduce storage, as well as the online calculation time, while maintaining a similar accuracy to our initial approach.

Presentation of chapters

Chapter 1

This chapter briefly presents neutron notions (reaction chain, cross-sections, neutron flux, neutron transport equation,...) as well as the simulation context (the multi-scale structure of the reactor core, the calculation scheme in two steps, ...). The main goal is to help readers reach a better understanding of our problem.

Chapter 2

This chapter is dedicated to the description of different formats of low-rank tensors, such as: r-term format, subspace tensor format (Tucker format), hierarchical format. We then explain why we chose the Tucker decomposition (based on the Tucker format) for the cross-section reconstruction problem.

Chapter 3

This chapter, based on our first article, details our proposed model: the Tucker decomposition. The cross-sections are approached by a linear combination of the tensor products of the tensor directional basis functions. We describe step by step the following problems: the construction of tensor directional basis functions (by an extension of the Karhunen-Loève decomposition) and the determination of the coefficients in the combination (by a system of linear equations using the greedy algorithm). The first benchmark is also proposed to compare our model with the multilinear interpolation.

Chapter 4

In this chapter, based on our second article, we propose benchmarks on two calculation domains: standard and extended. Statistical analyzes are proposed that allow us to find the major factors to improve the reconstruction accuracy: increasing the number of tensor directional basis functions or discretizing finer some parameter values zones found by the analysis.

Chapter 5

This chapter, based on our third article, presents the possibility of a sparse representation of data (the coefficients in the Tucker decomposition) in our model. Two techniques are proposed: *a posteriori* sparse representation and *a priori* sparse representation. *A posteriori* sparse representation is performed when the coefficients are already calculated and we can therefore remove small

coefficients. *A priori* sparse representation uses a prediction of coefficients values based on an order proposed for tensor directional basis functions. These methods allow us to significantly reduce the number of coefficient for equivalent accuracy and open the possibility to reduce the number of APOLLO2 calculations.

Chapter Conclusion

This chapter is reserved for the conclusion and perspectives of our model.

Part I

Physical context and mathematical background

Chapter 1

Physical context of the neutron cross-section reconstruction

The purpose of this chapter is to present the physical context of neutron cross-section reconstruction. Some notions of neutronic physics related to our problem will be introduced. We refer the reader to the books [Lewis and Miller, 1984], [Reuss, 2003] and [Marguet, 2013] for more details of neutronic physics and nuclear reactors.

1.1 Nuclear reactions

1.1.1 Atomic structure and isotopes

We recall here some basic notions of the structure of an atom in order to get a better comprehension for the next introduction. The atom is the unit component of matter, it consists of smaller particles (sub-atomic), such as: neutron (n), proton (p) and electron (e^-) , as described in figure 1.1. The neutrons and the protons of an atom are called nucleons and they constitute the so-called a nucleus. The term nuclei is used to designate many nucleus. We often denote by the letter Z the number of protons and by the letter A the number of nucleons (protons and neutrons).

A chemical element is identified by the number Z. The isotopes (also called nuclide) of a chemical element are determined by an atom that has a fixed number of protons but different number of neutrons. For example, uranium has the following isotopes: uranium 234 ($^{234}_{92}U$), uranium 235 ($^{235}_{92}U$) and uranium 238 ($^{238}_{92}U$), they have the same number of protons (92 protons) while the number of neutrons are respectively: 142, 143 and 146. Due to this difference, the isotopes of an element can have different properties, e.g: one can be fissible while others are not. An important property of fissile isotopes is to have uneven number of nucleons (A), e.g. $^{233}_{92}U$, $^{235}_{92}U$, $^{239}_{94}Pu$.

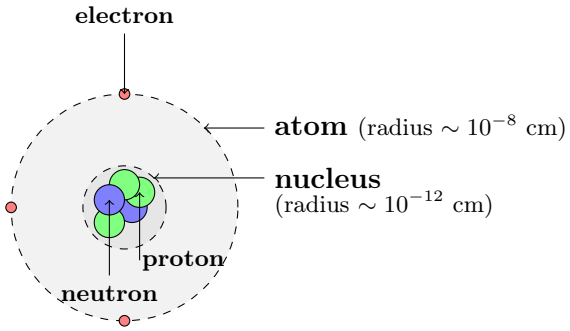


Figure 1.1 – Structure of an atom.

1.1.2 Principal reaction kinds

In table 1.1, we present the principal reaction kinds of nuclear reactions: scattering, fission and capture.

Reaction kind		Formula	Note
Scattering	Elastic scattering	$n + {}^{A}_{Z}X \rightarrow {}^{A}_{Z}X + n$	Kinetic energy conservation
	Inelastic scattering	$n + {}^{A}_{Z}X \rightarrow {}^{A}_{Z}X^{*} + n$; ${}^{A}_{Z}X^{*} \rightarrow {}^{A}_{Z}X + \gamma$	Kinetic energy is not conserved
			* Excited state
Fission		$n + {}^{A}_{Z}X \rightarrow {}^{A+1}_{Z}X^{*} \rightarrow {}^{A_1}_{Z_1}X_1 + {}^{A_2}_{Z_2}X_2 + \nu n$	ν: number of neutrons
			X_1, X_2: heavy elements ($A_1, A_2 \sim 100$)
Capture	Radiative capture	$n + {}^{A}_{Z}X \rightarrow {}^{A+1}_{Z}X + \gamma$	
	Particle ejection	$n + {}^{A}_{Z}X \rightarrow {}^{A_1}_{Z_1}X_1 +$ light particle	$A_1 \sim A$. Light particle: p, ${}^{4}_{2}He$,...

Table 1.1 – Principal reaction kinds of nuclear reactions.

1.1.3 Nuclear fission chain reaction

A fission is a process in which a fissile nucleus (typically, Uranium U^{235}_{92}, Plutonium Pu^{239}_{94}) absorbs neutrons and splits into lighter nuclei, for example:

$$ {}^{235}_{92}U + n \longrightarrow {}^{236}_{92}U^{*} \longrightarrow {}^{92}_{36}Kr + {}^{141}_{56}Ba + 3n \tag{1.1} $$

This process produces new free neutrons (3 in example (1.1)) and releases a large amount of energy in the form of heat (about 200 MeV per fission of U235). Again, part of this newly produced free neutrons collide with other fissile nucleus (while the others can be absorbed by the non-fissile material or leak out of the material), which generates more heat, more neutrons and cause a reaction chain, see figure 1.2.

The global behavior of the nuclear fission chain is related to a factor, called *effective multiplication factor* and denoted by k_{eff}. The k_{eff} is used to describe the average number of neutrons from one

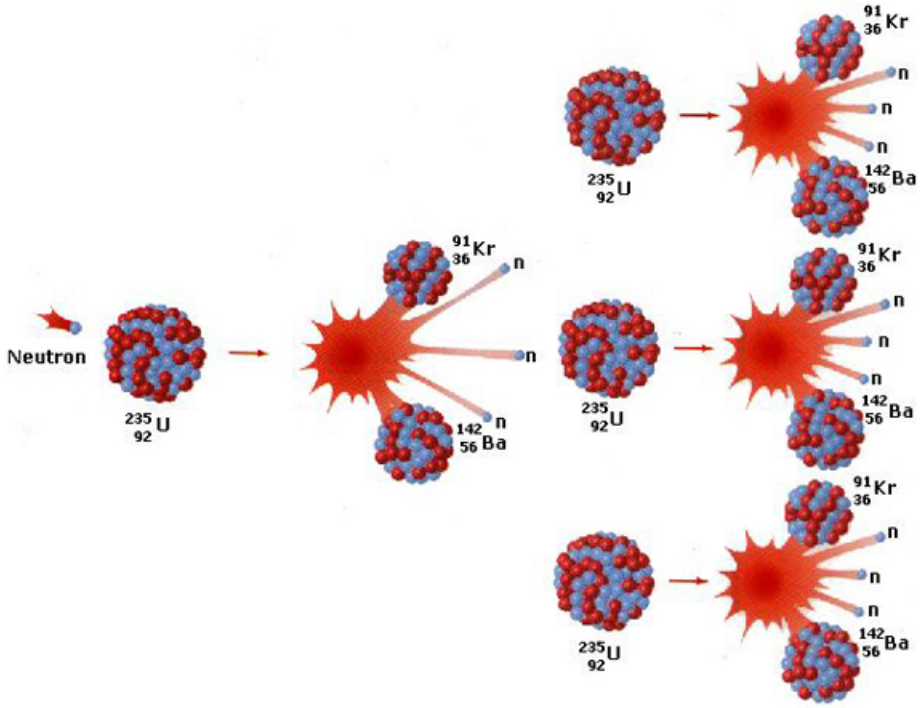


Figure 1.2 – Nuclear fission chain reaction. [1]

fission that cause other fissions. Therefore, this factor determines the evolution of the reaction chain, i.e. N fissions lead to the next ones: Nk_{eff}, Nk_{eff}^2, Nk_{eff}^3, ..., fissions.

The value of k_{eff} is classified as follows:

— $k_{eff} > 1$ (*super-criticality*): number of fissions increases exponentially and the reaction is explosive.

— $k_{eff} = 1$ (*criticality*): number of fissions is stable (constant) in time.

— $k_{eff} < 1$ (*sub-criticality*): number of fissions decreases exponentially and finally the reaction stops.

Fission rate (number of fissions per a unit time) depends on the population of neutrons. Thus, in order to reduce the number of neutrons which were born in the reactor core, *absorbent substances* (also called *neutron poisons* or *neutron absorbers*) are used, such as: boron, gadolinium, etc. The reaction chains themselves produce neutron poisons in their fission products, of which the most important substance is xenon-135 ($^{135}_{54}Xe$). The concentration of xenon increases quickly when the reactor starts (which may cause the reactor to shutdown at this stage) and gradually becomes more stable, see figure 1.3.

The fission rate also depends on the energy E or the velocity $v = ||\overrightarrow{v}||$ of the incident neutron (since $E = mv^2/2$). In general, the slow neutrons (corresponding to the low energy) are captured

1. Source of figure: http://physics.tutorvista.com/modern-physics/fission.html

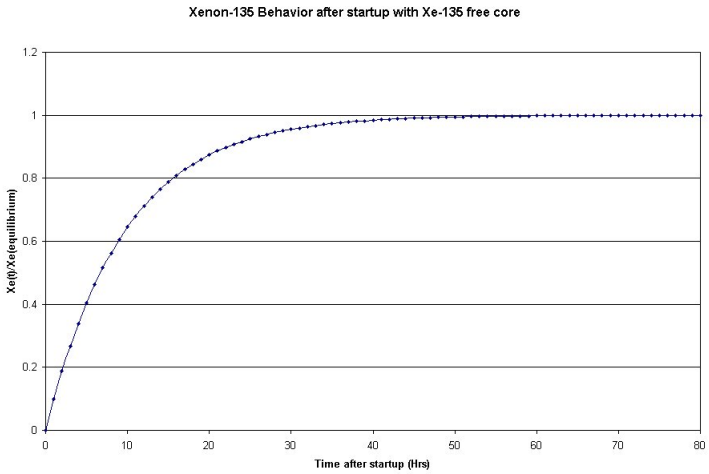


Figure 1.3 – Xenon behavior in reactor core. [2]

more easily in the collision than the fast ones (corresponding to the high energy), meaning that the slow neutrons are more favorable for fission reactions. Therefore, a moderator medium which absorbs very few neutrons and slows down the fast neutrons is employed to increase the fission rate. A good moderator is regular water (H_2O), a better one is heavy water (D_2O).

1.2 Nuclear reactor

1.2.1 General description

A nuclear reactor is a system designed to initiate and control nuclear chain reactions. Depending on its purpose, it can be classified as the research reactor, the military reactor or the power reactor. Here, we are more interested in power reactors because they are used in the nuclear power plants to generate electricity.

The principal mechanism of nuclear reactors is that they produce thermal energy from the heat of nuclear chain reactions which is convert into mechanical or electrical form. Some important components in a nuclear reactor are:

— *The core* that contains nuclear fuels, control systems, and structural materials. The nuclear reactions take place inside the nuclear fuels and produce heat;

— *The coolant* that is a fluid circulating through the core and transferring the heat from the fuel to a turbine. It could be liquid (e.g. water), gas (e.g. hydrogen, helium), liquid gas (e.g. carbon dioxide);

— *The turbine* that converts the heat into electricity.

2. Source of figure: http://www.nucleartourist.com/basics/xe135-1.htm

1.2.2 Pressurised Water Reactors (PWR)

Pressurised Water Reactor (PWR) is the most widely used type of reactors for nuclear power plants in the world as well as in France. In a PWR, the heat is created by the fissions inside the fuels of the core. The core is put in a reactor vessel. Water (pressurized at about 155 atm to avoid boiling) circulates in a primary coolant loop and carries the heat (about $330\,℃$) to a steam generator. Inside the steam generator which has lower pressure, the water is vaporized and conducted through a secondary loop, spinning the turbine. The turbine is connected to a generator which produces electricity. The steam is then condensed back into water inside the condenser (figure 1.4). We note that between the primary and the secondary loop, there is only heat exchange but not water exchange. In the PWR core, water (used in the primary coolant loop) is also employed as moderator to slow down neutrons.

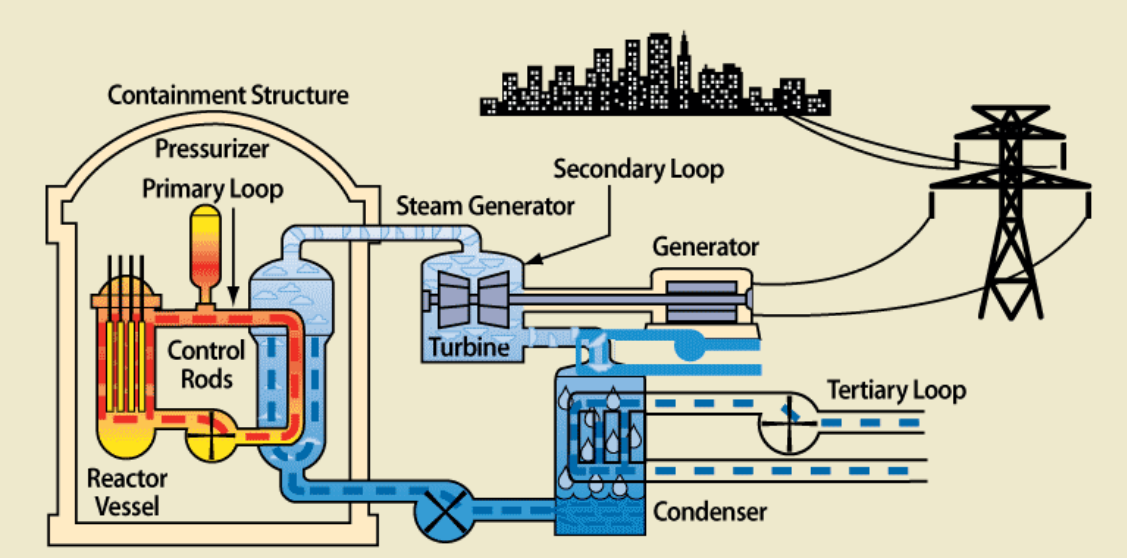


Figure 1.4 – Diagram of a pressurized water reactor. [3]

The PWR core has a multi-scale structure (figure 1.5). It contains between 150-200 fuel assemblies arranged as a square lattice and surrounded by boron water . A typical assembly is often made of $17 \times 17 = 289$ rods, consisting of 264 fuel rods and 25 vacant rods (figure 1.6a). The fuel rods ($\sim 4\,m$ in height) are filled with the individual pellets ($\sim 1\,cm$ in height) (see figure 1.6b). The vacant rods are dedicated for the vertical insertion of the absorbent rod in the guide tubes or the instrument.

3. Source of figure: https://en.wikipedia.org/wiki/Pressurized_water_reactor

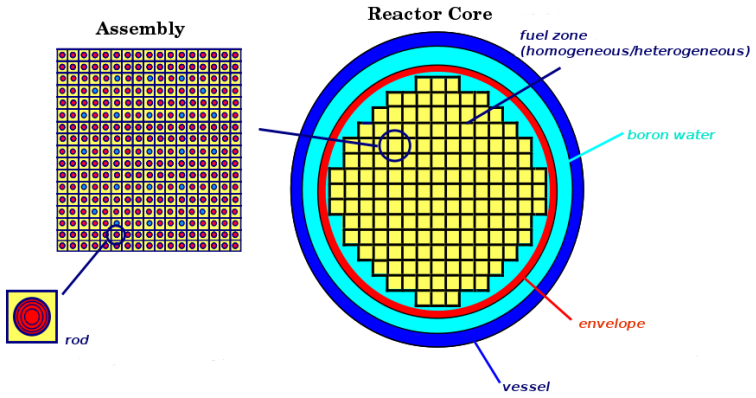
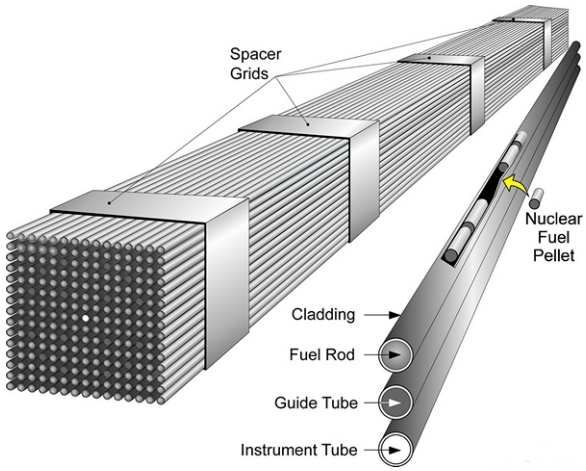
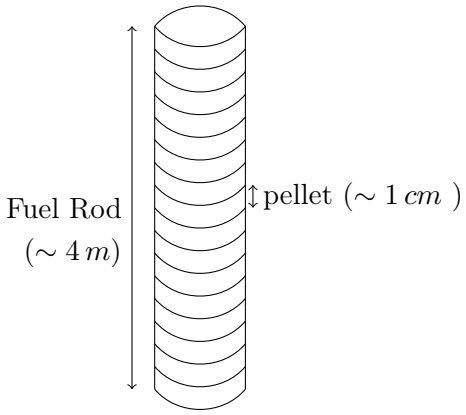


Figure 1.5 – PWR - multi-scale structure: core containing many assemblies, assembly containing many rods. [4]



(a) Fuel assembly of PWR with about 289 rods. [5]



(b) Fuel rod.

Figure 1.6 – Fuel assembly of PWR and fuel rod.

1.2.3 Some assembly types: UOX, MOX, UOX-Gd

As described in previous section, an assembly contains different fuels and vacant rods put in 289 positions. Depending on the fuel rod composition and position, we have different types of assemblies. We present here some assemblies related to our work.

4. Source of figure: Ph.D Thesis of Pierre GUÉRIN (Méthodes de décomposition de domaine pour la formulation mixte duale du problème critique de la diffusion des neutrons)

5. Source of figure: http://www.nuclear-power.net/nuclear-power-plant/nuclear-fuel

1.2.3.1 UOX (Uranium Oxide: UO_2) assembly

Most reactors use uranium enriched in the isotope 235 with the enrichment is between 3%-5% because the natural uranium contains only 0.7% of uranium-235. The assembly constituted by the UOX fuel will be called the UOX assembly.

1.2.3.2 MOX (Mixed Oxide: UO_2-PuO_2) assembly

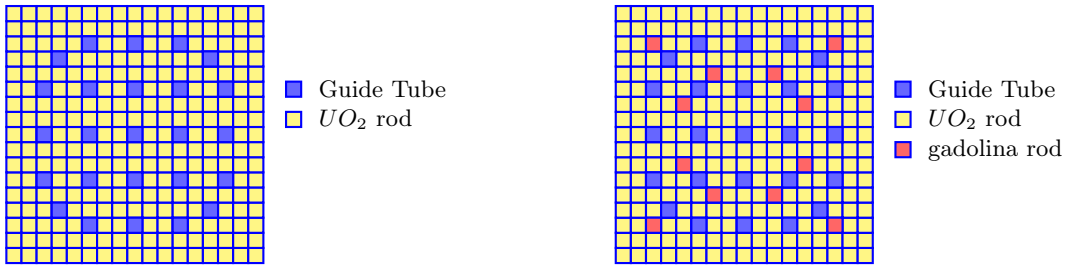
Plutonium (Pu) is also a fissile source for nuclear reactors beside the natural uranium. It is produced inside the nuclear reactors by capturing the neutrons as the following reaction:

$$^{238}_{92}U + n \longrightarrow ^{239}_{92}U \longrightarrow ^{239}_{93}Np + e^- \longrightarrow ^{239}_{94}Pu + e^- \tag{1.2}$$

Moreover, plutonium can be extracted from spent fuels. This allows us to recycle the used fuels. Therefore, MOX, an uranium fuel mixed with plutonium, is also used for the nuclear reactors. The assembly constituted by some MOX fuels will be called the MOX assembly.

1.2.3.3 UOX-Gd (UOX-Gadolinium: UO_2-Gd_2O_3) assembly

An UOX-Gd assembly is an UOX assembly where some positions of the UOX fuel rods are replaced by UO_2-Gd_2O_3 rods (gadolinia rods), see figure 1.7. Since the gadolinium Gd_2O_3 is a neutron poison (absorbing neutrons), the UO_2-Gd_2O_3 rods are used as burnable poison rods to limit excess of fissions, hence to mitigate localized power peaking.



(a) UOX assembly. (b) UOX-Gd assembly.

Figure 1.7 – UOX and UOX-Gd assemblies.

1.3 Neutronic

1.3.1 Neutron cross-section

In order to quantify the neutron reaction rates, a notion which represents interaction probability between an incident neutron and a target nucleus or nuclei is required. This notion is referred to

as *cross-section* in particle physics. A simple explanation of cross-section notion is illustrated by figure 1.8. In this figure, an incident neutron moves with a velocity $\overrightarrow{v}$ toward a target nucleus. The incident neutron and the target nucleus are supposed to have spherical forms with the radius r and R respectively. In this figure, a collision only happens if the mass center of the incident neutron is inside the cylinder of the radius $R + r$ and of axis paralleling to the velocity $\overrightarrow{v}$. Thus, the surface (or the cross-section) perpendicular to the axis of the cylinder represents the interaction likelihood of the incident neutron and the target nucleus. Neutron cross-section notion are therefore illustrated by the area of this surface. This explains why magnitude's neutron cross-sections is in the order of $10^{-24}\,cm^2$ (since the area $\sim \pi R^2$ and $R \sim 10^{-12}\,cm$). Cross-sections are measured in the unit *barn*:

$$1 \text{ barn} = 10^{-24}\,cm^2 = 10^{-28}\,m^2$$

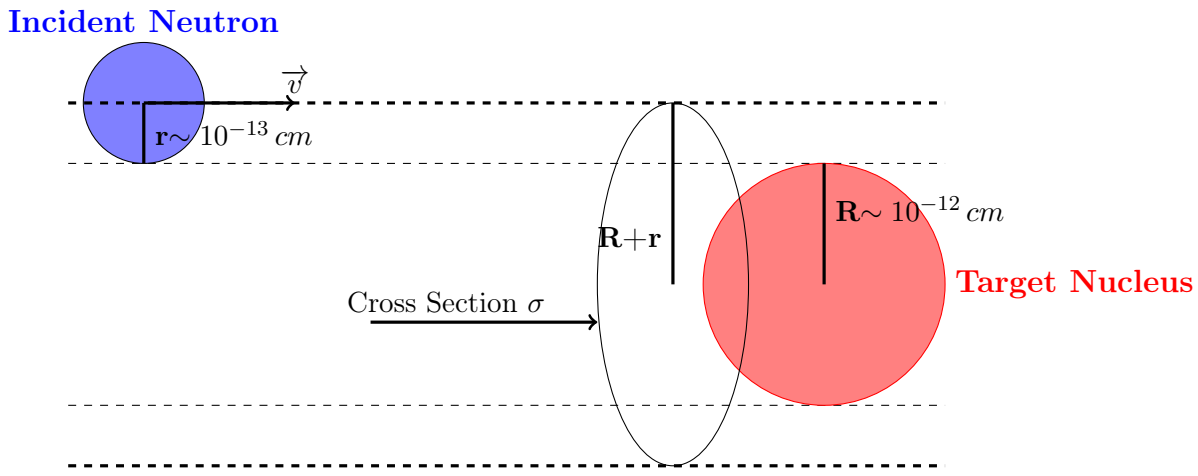


Figure 1.8 – Illustration for cross-section notion.

A cross-section may be microscopic, denoted by σ, that characterizes an individual target nucleus, or macroscopic, denoted by Σ, that describes the material interaction characteristic with a large number of target nuclei. Microscopic cross-sections depend on energy E of the incident neutron: $\sigma = \sigma(E)$ whereas macroscopic cross-sections depend on spatial position $\overrightarrow{r} = \overrightarrow{r}(x, y, z)$, energy E of the incident neutron at a given instant t: $\Sigma = \Sigma(\overrightarrow{r}, E, t)$. The relation between the macroscopic and microscopic cross-sections is defined by:

$$\Sigma(\overrightarrow{r}, E, t) = N(\overrightarrow{r}, t)\sigma(E)$$

where $N(\overrightarrow{r}, t)$ is the density of the target nucleus per a volume unit at the moment t. This relation implies that macroscopic cross-sections are expressed in inverse of length unit: $[\Sigma] = cm^{-1}$.

The different cross-sections kinds are distinguished by their corresponding *reaction* which are denoted by the index r in (σ_r, Σ_r). Here, r could be: f (fission), s (scattering), c (capture), a

(absorption), t (total), ... The relations between these cross-sections are defined as follows:

$$\sigma_a = \sigma_f + \sigma_c; \Sigma_a = \Sigma_f + \Sigma_c$$

$$\sigma_t = \sigma_s + \sigma_a;\ \Sigma_t = \Sigma_s + \Sigma_a$$

Using the definition of the cross-sections, we deduce that the larger the cross-section is, the more likely interaction between the nucleus and the incident neutron is. Moreover, cross-sections depend on the isotope type. For example, at low neutron energy, the fissile isotope U_{92}^{235} has a large fission cross-section (σ_f, Σ_f) while for the isotope U_{92}^{239} this cross-section is small.

Until now, we considered only cross-sections for an isotope kind i. In reality, most material is composite. Therefore, we need to determine a notion corresponding to the *global macroscopic cross-section* for the composite material. With a given reaction kind r, this macroscopic cross-section is the sum of the macroscopic cross-sections of all elements:

$$\begin{aligned}\Sigma_r =& \Sigma_{r,1} + \ldots + \Sigma_{r,I} \\ =& c_1\sigma_{r,1} + \ldots + c_I\sigma_{r,I} \\ =& \sum_{i=1}^{I} c_i\sigma_{r,i}\end{aligned} \tag{1.3}$$

where I is the number of isotopes i included in the composite material and c_i is the concentration of the isotope i.

1.3.2 Neutron flux

Cross-sections provide us information about interaction probability but this is not sufficient for studying the chain reactions. We also need information about the population of free neutrons in the medium inside a nuclear reactor. This population is sufficiently large ($\sim 10^8$ neutrons/cm^3) to use the "neutron density" concept for simulating the variation of this population. Neutron density, denoted by n, is the number of neutrons per time unit t and per volume unit of a *phase space* Ω . Here, the phase space Ω for a neutron of mass m is spanned by its spatial position $\overrightarrow{r} = (r_x, r_y, r_z)$ and its velocity $\overrightarrow{v}$. In neutronic, we prefer to replace $\overrightarrow{v}$ by $(E, \overrightarrow{\Omega} = \dfrac{\overrightarrow{v}}{v})$ since $E = \dfrac{1}{2}mv^2$ with $v := ||\overrightarrow{v}||$. The direction of motion $\overrightarrow{\Omega}$ is called *solid angle* which is defined in a polar coordinate system by a polar angle θ and an azimuthal angle α, i.e. $\overrightarrow{\Omega} = \overrightarrow{\Omega}(\theta, \alpha)$. Thus, we can denote the phase space $\Omega = \Omega(\overrightarrow{r}, E, \overrightarrow{\Omega})$ and the neutron density $n = n(\overrightarrow{r}, E, \overrightarrow{\Omega}, t)$. The neutron population is then represented by the so-called *angular flux* ψ with the following definition:

$$\psi(\overrightarrow{r}, E, \overrightarrow{\Omega}, t) = n(\overrightarrow{r}, E, \overrightarrow{\Omega}, t)v \tag{1.4}$$

Here, $v = \sqrt{2E/m}$.

The integral of the angular flux over whole solid angle gives us the so-called *scalar flux* ϕ with the following definition:

$$\phi(\overrightarrow{r}, E, t) = \int_{4\pi} \psi(\overrightarrow{r}, E, \overrightarrow{\Omega}, t) d^2\Omega \tag{1.5}$$

The formula (1.4) means that the angular flux $\psi(\overrightarrow{r}, E, \overrightarrow{\Omega}, t)$ takes into account only the neutrons having the energy E and moving in the fixed direction $\overrightarrow{\Omega}$. The scalar flux $\phi(\overrightarrow{r}, E, t)$ in (1.5) takes into account all neutrons having the energy E and moving in any direction with the condition that they are in a same volume $d^3\overrightarrow{r}$.

It can be noticed that the notion of "flux" in neutronics is completely different from which of classical fluid mechanics. The equivalent would be current $\overrightarrow{j} = n\overrightarrow{v}$.

1.3.3 Neutron transport equation

The neutron transport equation is based on the Boltzmann equation. The Boltzmann equation is proposed by Ludwig Boltzmann [Boltzmann, 1970] to describe the statistical behavior of mono-atomic gas. The properties of the neutron population in a reactor core are similar to this gas since the neutron density is very low compared with that of atoms. Therefore, this equation can be applied to the neutron population (by neglecting the neutron-neutron and neutron-electron interactions), to predict the global variation of this population.

The neutron transport equation can be written under different forms: either integral or integral-differential. These two forms are mathematically equivalent but the integral-differential one is often used for deterministic methods (to which this work is related). The integral-differential transport equation is expressed as:

$$\begin{aligned}\frac{1}{v}\frac{\partial\psi(\overrightarrow{r}, E, \overrightarrow{\Omega}, t)}{\partial t} =& -\overrightarrow{\nabla}.[\overrightarrow{\Omega}\psi(\overrightarrow{r}, E, \overrightarrow{\Omega}, t)] - \Sigma_t(\overrightarrow{r}, E, t)\psi(\overrightarrow{r}, E, \overrightarrow{\Omega}, t) \\ &+ \int_0^\infty dE' \int_{4\pi} d^2\Omega' \Sigma_s(\overrightarrow{r}, E' \to E, \overrightarrow{\Omega'} \to \overrightarrow{\Omega}, t)\psi(\overrightarrow{r}, E', \overrightarrow{\Omega'}, t) \\ &+ \frac{1}{4\pi}\int_0^\infty dE' \chi(E' \to E)\nu\Sigma_f(\overrightarrow{r}, E', t)\phi(\overrightarrow{r}, E', t) + S_{ext}(\overrightarrow{r}, E, \overrightarrow{\Omega}, t)\end{aligned} \tag{1.6}$$

Where:

— $\overrightarrow{\nabla}.$ is the divergence operator.

— $\chi(E' \to E)$ is the energy distribution spectra.

— S_{ext} is the external source term.

In order to reach the critical stationary state (where time-dependence terms disappear in (1.6)), we introduce a parameter to balance the appearance and disappearance terms. This parameter is considered as the effective multiplication factor k_{eff} (see the section 1.1.3) and determined as the

eigenvalue of the following equation:

$$\begin{aligned}\overrightarrow{\nabla}.[\overrightarrow{\Omega}\psi(\overrightarrow{r},E,\overrightarrow{\Omega})]+\Sigma_t(\overrightarrow{r},E)\psi(\overrightarrow{r},E,\overrightarrow{\Omega})&\\ =\int_0^\infty dE'\int_{4\pi}d^2\Omega'\Sigma_s(\overrightarrow{r},E'\to E,\overrightarrow{\Omega'}\to\overrightarrow{\Omega})\psi(\overrightarrow{r},E',\overrightarrow{\Omega'})&\\ +\frac{1}{4\pi\mathbf{k_{eff}}}\int_0^\infty dE'\chi(E'\to E)\nu\Sigma_f(\overrightarrow{r},E')\phi(\overrightarrow{r},E')+S_{ext}(\overrightarrow{r},E,\overrightarrow{\Omega})&\end{aligned}\tag{1.7}$$

This equation requires inputs as different macroscopic cross-sections: Σ_t, $\nu\Sigma_f$, Σ_s and provides us outputs as neutron flux $\psi(\overrightarrow{r},E,\overrightarrow{\Omega})$ and k_{eff}.

A simplified equation of the equation (1.7) is the diffusion equation:

$$\begin{aligned}-\overrightarrow{\nabla}.[D(\overrightarrow{r},E)\overrightarrow{\nabla}\phi(\overrightarrow{r},E)]+[\Sigma_a(\overrightarrow{r},E)+\int_{E_{min}}^{E_{max}}\Sigma_s(\overrightarrow{r},E\to E')dE']\phi(\overrightarrow{r},E)&\\ =\int_{E_{min}}^{E_{max}}\Sigma_s(\overrightarrow{r},E'\to E)\phi(\overrightarrow{r},E')dE'+\frac{\chi(E)}{k_{eff}}\int_{E_{min}}^{E_{max}}\nu(E')\Sigma_f(r,E')\phi(\overrightarrow{r},E')dE'&\end{aligned}\tag{1.8}$$

Where D is the diffusion coefficient.

With the two-group energy theory (the energy E is discretized into two groups $g=1$ and $g=2$), the diffusion equation (1.8) becomes the two-group diffusion equations:

$$\begin{cases}-D_1\triangle\phi_1+(\Sigma_a^1+\Sigma_s^{1\to2})\phi_1 &= \dfrac{\nu\Sigma_f^1\phi_1+\nu\Sigma_f^2\phi_2}{k_{eff}}+\Sigma_s^{2\to1}\phi_2\\ -D_2\triangle\phi_2+(\Sigma_a^2+\Sigma_s^{2\to1})\phi_2 &= \Sigma_s^{1\to2}\phi_1\end{cases}\tag{1.9}$$

1.3.4 Reactivity in the infinite medium with two-group diffusion theory

The k_{eff} value is used to describe the behavior of nuclear reactors. In normal operating conditions of reactor, k_{eff} is equal to 1 (criticality value). When k_{eff} varies, a small deviation from the criticality value can result a significant change in reactor power. Therefore, for the practical purpose, the "reactivity" is more useful with the following definition:

$$\text{reactivity}=1-\frac{1}{k_{eff}}\tag{1.10}$$

We consider here a simplified configuration: reactor core is assumed as a homogeneous infinite medium and the 2 two-group diffusion equations (1.9) are solved over this medium. Using this hypothesis, the reactivity can be determined by an analytic formula which takes into account some macroscopic cross-sections.

The infinite medium means that all assemblies have the same configuration and they are infinitely arranged together. In this medium, neutrons can not leak out of the system and the effective

multiplication factor k_{eff} becomes the infinite multiplication factor k_∞: $k_{eff} = k_\infty$. With the two-group diffusion theory, we obtain the following analytic formula (see page 1172, 1173, 1221 of the book [Marguet, 2013]):

$$\text{reactivity} = 1 - \frac{1}{k_\infty}, \text{with } k_\infty = \frac{\nu\Sigma_f^1 * (\Sigma_t^2 - \Sigma_{so}^{2\to2}) + \nu\Sigma_f^2 * \Sigma_{so}^{1\to2}}{(\Sigma_t^1 - \Sigma_{so}^{1\to1}) * (\Sigma_t^2 - \Sigma_{so}^{2\to2}) - \Sigma_{so}^{1\to2} * \Sigma_{so}^{2\to1}} \tag{1.11}$$

This formula takes into account some macroscopic cross-sections (not all), such as: the `macro totale`-Σ_t^g, the `macro fission`-Σ_f^g, the `macro nu*fission`-$\nu\Sigma_f^g$ and the `macro scattering`-$\Sigma_{so}^{g\to g'}$, where the energy group $g \in \{1, 2\}$, $g' \in \{1, 2\}$ and the index o in Σ_{so} is the anisotropy order (order for Legendre polynomial expansion for variable "angle").

1.4 Reactor core simulation and process of cross-section reconstruction

In the following descriptions, the neutron transport equation is always considered in the stationary state (time-independence). This equation is also called transport equation in this work.

1.4.1 Numerical methods

Solving the neutron transport equation plays a fundamental role in reactor core simulation. Deterministic methods [Lewis and Miller, 1984] are used to solve numerically this equation (often under the integral-differential form). There are six parameters involved in such resolutions: three for $\overrightarrow{r} = (r_x, r_y, r_z)$, two for $\overrightarrow{\Omega} = (\theta, \alpha)$ and one for E. The resolution is based on some discretization techniques, for instance:

- For space $\overrightarrow{r}$: spatial discretization with finite difference methods [LeVeque, 2007] or finite element methods [Madenci and Guven, 2015].
- For angle $\overrightarrow{\Omega}$: angular discretization with discrete ordinates methods S_N, spherical harmonics methods P_N [Abramowitz and Stegun, 1964], or simplified spherical harmonics methods SP_N [Pomraning, 1993], [McClarren, 2010].
- For energy E: energy discretization with multi-group formalism [Hébert, 2010], often expressed as follows:

$$E = [E_{max}, E_{min}] = \underbrace{[E_0, E_1]}_{\text{group } g=1} \cup \underbrace{[E_1, E_2]}_{\text{group } g=2} \cup \ldots \cup \underbrace{[E_{G-1}, E_G]}_{\text{group } g=G} \tag{1.12}$$

  (the energy groups are indexed by the index g in the decreasing order of energy because neutron energy decreases in time when colliding with moderator (water).)

In figure 1.9, we summarize the principal numerical methods for solving the neutron transport equation.

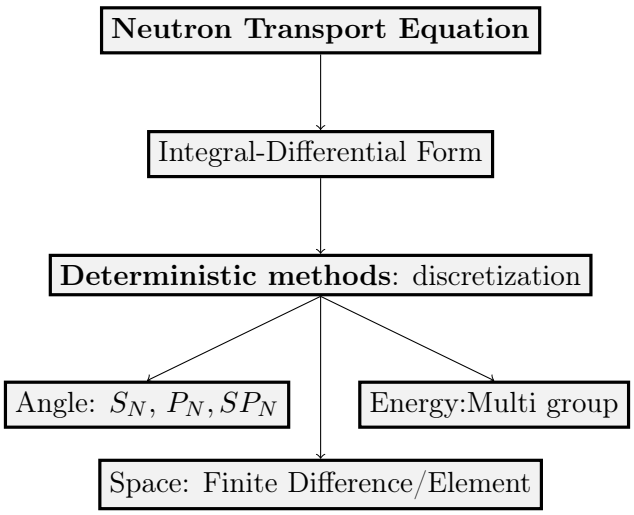


Figure 1.9 – Numerical methods to solve the neutron transport equation.

1.4.2 Calculation scheme with two steps

The neutron transport equation is well understood nowadays but its numerical resolution for the full core simulation is still a challenge, due to the requirement of memory storage and computational time. Indeed, for a real geometry modeling of core (core in 3D), we need to solve the neutron transport equation with a huge number of unknowns ($\sim \mathcal{O}(10^9)$).

To deal with this problem, a two-step calculation scheme is proposed in order to reduce complexity calculations and accomplish the core simulation. This scheme is based on the multi-scale structure of the reactor core: core containing assemblies, assembly containing rods/cells. Therefore, the two steps are separately performed by two codes: first, *lattice code* for all assembly types and then *core code* for the whole core, as described in figure 1.10.

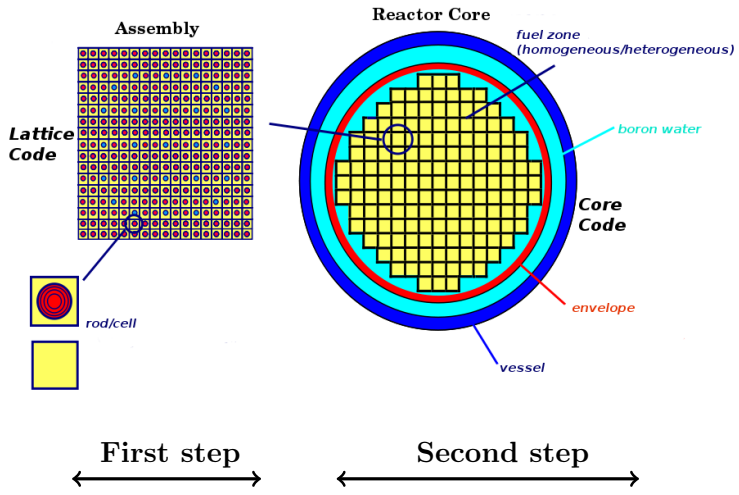


Figure 1.10 – Two-step calculation scheme based on the multi-scale structure of reactor core.

These two steps are respectively named *assembly calculation* and *core calculation* with the following goals:

— *Assembly calculation*: the lattice code solves the neutron transport equation on each assembly type. These assemblies are assumed in a 2D infinite medium. The resolution is performed on a very fine discretization for space $\overrightarrow{r}$ and for energy E. Finally, we obtain the angular neutron flux $\psi(\overrightarrow{r}, E, \overrightarrow{\Omega})$ which is used in the energy condensation and the spatial average for cross-sections:

$$\Sigma_r^g(cell_a) := \overline{\Sigma}_r^g(cell_a) = \frac{\int\limits_{\overrightarrow{r}\in cell_a}\int\limits_{E\in g} \Sigma_r(\overrightarrow{r}, E)\psi(\overrightarrow{r}, E, \overrightarrow{\Omega})d\overrightarrow{r}dE}{\int\limits_{\overrightarrow{r}\in cell_a}\int\limits_{E\in g} \psi(\overrightarrow{r}, E, \overrightarrow{\Omega})d\overrightarrow{r}dE} \tag{1.13}$$

where r is the reaction kind, g is the energy group and $cell_a$ is a cell in a spatial discretization of the assembly a. All these results are gathered inside hierarchical library files which are named *neutron libraries*.

— *Core calculation*: the core code solves a simplified neutron transport equation for the full core in 3D. This resolution needs cross-section values at any point in the core. However, we can not calculate on the fly these values due to its huge cardinal ($\sim \mathcal{O}(10^9)$). We therefore replace required values by their interpolation values based on a reconstruction process. This reconstruction process relies on the pre-computed values of Σ_r^g determined by (1.13).

We illustrate in figure 1.11 the two-step calculation scheme for the core simulation.

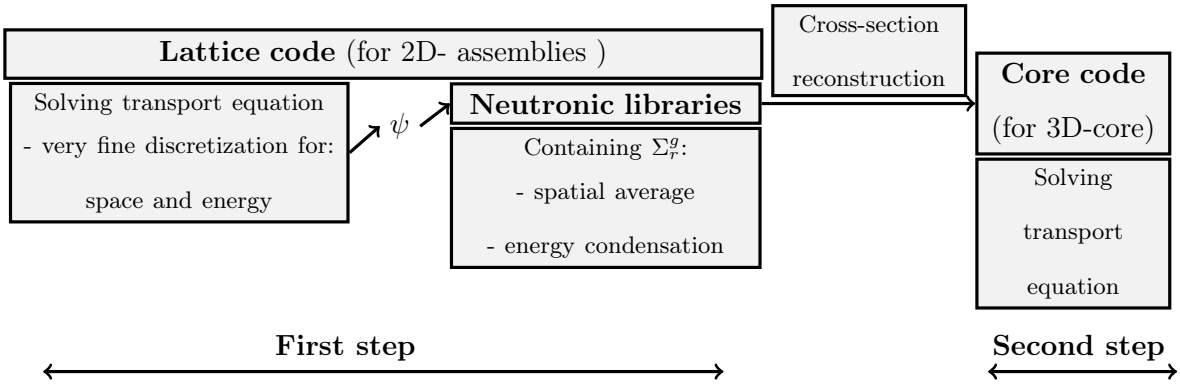


Figure 1.11 – Two-step calculation scheme for the core simulation.

It should be noted that averaged and condensed cross-sections in (1.13) use the angular flux as weight functions in order to preserve the global reaction rate τ between the two steps: $\tau = \Sigma_r^g \int\limits_{\overrightarrow{r}}\int\limits_{E\in g} \psi = \int\limits_{\overrightarrow{r}}\int\limits_{E\in g} \Sigma_r\psi$.

1.4.3 Reconstruction of cross-sections in each calculation step

Cross-section values are required in the whole core simulation but we only have a limited number of these values pre-calculated on assemblies because the two-step calculation scheme is used. Therefore, the reconstruction of cross-sections needs to be performed in order to connect the two calculation steps. This process can be briefly described as follows:

(i) Offline: pre-computing cross-sections on assemblies by a lattice code.

(ii) Offline: storing information about these pre-computed values in neutron libraries.

(iii) Online: using the stored information to *evaluate* cross-sections in the core.

In order to avoid the confusion between the reconstruction notion used in the offline and online step, we distinguish here the two sub-processes:

— *Reconstructing cross-sections:* this is done before storing reconstruction information in the neutron libraries. Such sub-process is only explicit in the case where we need to convert pre-computed cross-section values into equivalent *reconstruction information*. If all pre-computed cross-section values are kept and stored, no convertation process is required and performed.

— *Evaluating cross-sections:* this is done for any point in the core. This sub-process uses reconstruction information in the previous sub-process to evaluate cross-sections by an interpolation method.

We call in general these two sub-processes by the *reconstruction of cross-sections*.

1.4.4 Reconstruction of cross-sections from 3D-space in the reactor core to parameter-phase space in the assemblies

In the nuclear reactor core simulations, cross-sections can be represented as multivariate functions: $\sigma = \sigma(x_1, \ldots, x_d)$ and $\Sigma = \Sigma(x_1, \ldots, x_d)$, where d is the number of physical parameters on which cross-sections depend.

Indeed, cross-sections depend on various parameters involved with different physical conditions and configurations, for instance: *i) burnup-bu (MWd/t), ii) fuel temperature-t_f ($^\circ C$), iii) moderator density-ρ_m (g/cm^3), iv) boron concentration-b_c (ppm), v) xenon level-xe (%), ...* Here, the *burnup* parameter is used to measure how much the nuclear fuel is consumed, that is expressed by the fraction of the actual energy released per initial mass of fuel (gigawatt-days/ton). The other parameters (t_f, ρ_m, b_c, xe) have already been introduced in previous sections. All these parameters vary in a physical space named here *parameter-phase space*. (This is not the phase space presented in the section 1.3.2 about neutron flux).

In the reactor core simulation, the core is modeled in 3D. Therefore, the core's geometry is meshed by cells in a full three-dimensional coordinates $Oxyz$. In ordre to solve the transport

equation, cross-section values are required at any cell in the full core. Moreover, each cell in the core corresponds to a specific physical condition, meaning that there exists a mapping $\mathcal{P}$ from $\mathbb{R}^3$-space to the parameter-phase space such that:

$$\mathcal{P} : \text{cell}_k \in \mathbb{R}^3 \mapsto \mathbf{x}(\text{cell}_k) = (x_1 \ldots, x_d) \in \text{parameter-phase space} \tag{1.14}$$

Therefore, cross-sections used in the 3D-core simulation are now defined as functions of parameters in the parameter-phase space as described in figure 1.12.

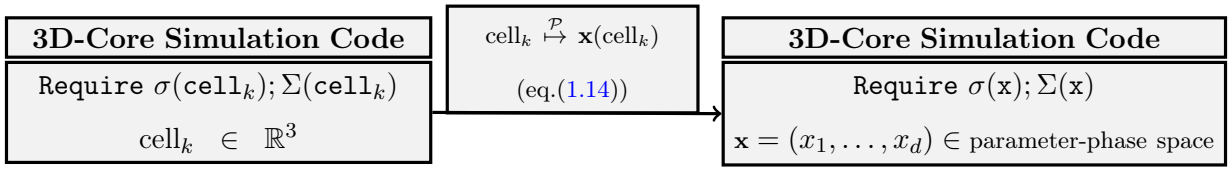


Figure 1.12 – Dependence of cross-sections on parameters in the parameter-phase space in a 3D-core simulation.

This explains why we can reconstruct cross-sections in the core from pre-calculated values in the parameter-phase space of the assemblies.

1.5 Core simulation at EDF and cross-section reconstruction problem

1.5.1 Calculation scheme with APOLLO2 (lattice code) and COCAGNE (core code)

EDF [6] owns and manages 58 PWRs in France. In order to control safely the operation of PWRs, a new core code named COCAGNE [Plagne and Ponçot, 2005] is being developed for a future core calculation chain of EDF. In this chain, APOLLO2 code [Sanchez et al., 2010] developed at CEA [7] is integrated and used as a lattice code via a package, called *GAB* (Library Automatic Generator).

The GAB package developed by EDF is for the objective of generating automatically neutron libraries from a given dataset. GAB uses the APOLLO2 code as a lattice solver. Each APOLLO2 calculation is employed for a point in the parameter-phase space. However, any calculation requires an input deck describing the geometry, material, solver options, the scheme for calculation... GAB generates automatically this input deck for each point (which is represented under feedback form, e.g: (bu, t_f, ρ_m, b_c, xe)), then distributes jobs on a cluster that requires APOLLO2 calculations and

6. Electricité de France
7. Commissariat à l´Énergie Atomique et aux Énergies Alternatives

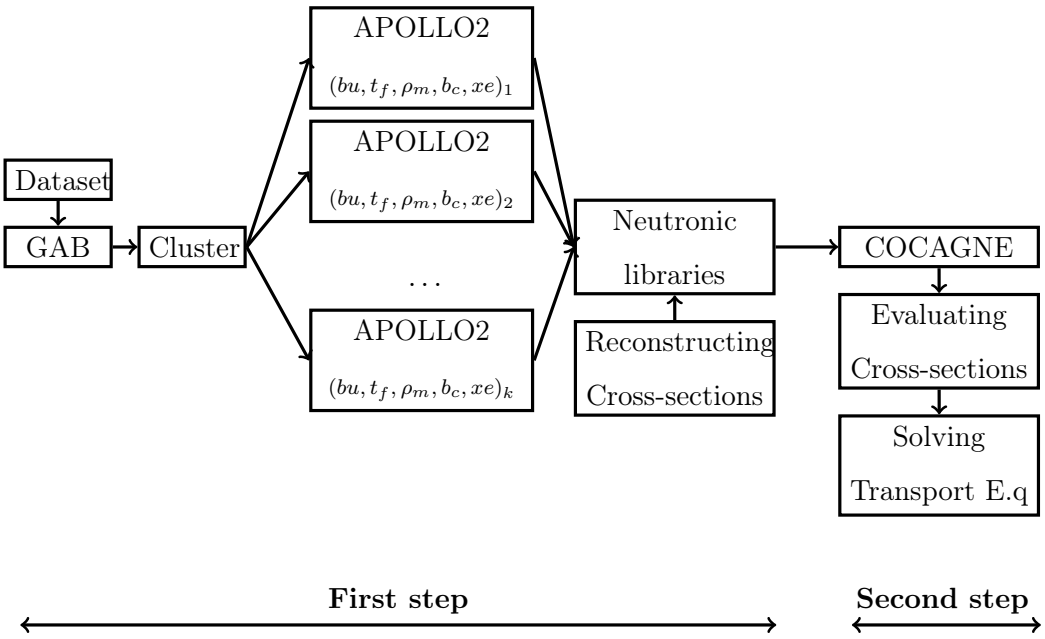


Figure 1.13 – Calculation scheme in core simulation at EDF

finally, gathers all results inside neutron libraries. One important thing is that the dataset generated by GAB is presented under a tensorized form (grid).

The calculation scheme described in section 1.4.2 and 1.4.3 is now applied to the core calculation chain of EDF as shown in the diagram of figure 1.13.

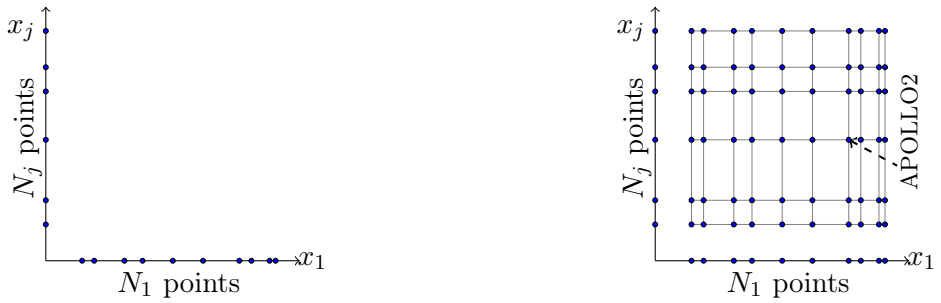
1.5.2 Current model for the reconstruction of cross-sections: multilinear interpolation

The multilinear interpolation model is currently implemented in the core code COCAGNE in order to reconstruct cross-sections. We describe with more details here the two main steps employed in this reconstruction:

(i) In the first step (pre-computing cross-sections on assemblies), each parameter $x_1, \ldots, x_d$ in the parameter-phase space is represented by a corresponding axis on which it is discretized into 1D-points (figure 1.14a). A tensorized grid is created (via GAB), containing the points $\mathbf{x} = (x_1, \ldots, x_d)$ which are constituted as a tensor product of all 1D-axial disretizations (figure 1.14). This grid is referred to as *multilinear grid*.

APOLLO2 calculations are performed at every node of this grid in order to provide all cross-section values of all cross-section kinds for the multilinear grid. The values obtained are then stored directly into neutron libraries.

(ii) In the second step (reconstructing cross-sections in core), the cross-sections are reconstructed



(a) Discretization on each parameter axis (feedback parameters)

(b) Tensorized grid - multilinear grid

Figure 1.14 – Multilinear grid used for the multilinear interpolation model.

by a multilinear interpolation which requires 2^d values of the surrounding nodes for a reconstructed value.

With the multilinear interpolation model, the reconstruction information stored in neutron libraries is exactly pre-computed cross-section values. There is no reconstruction process performed over these values before the storage of useful information into the neutron libraries (this can be different with other models, for example, in our proposed model (Tucker decomposition), we store basis functions and coefficients of our model instead of pre-computed cross-section values). The evaluating process in the core code is performed by the multilinear interpolation but it could have been performed by an other polynomial interpolation (e.g. Lagrange interpolation).

1.5.3 Requirement of a new model

In the multilinear interpolation model, if we presume that each axis has $N_j = N$ discretized points, we obtain:

— The number of nodes in the multilinear grid is equal to $\prod_{j=1}^{d} N_j = N^d$ (exponential function of dimension). Moreover, if we want to extend or refine some axes, we have to add a lot of nodes in the multilinear grid. For example, adding k points on an axis leads to the number of nodes is $k * N^{d-1} + N^d$, compared to N^d.

— The number of APOLLO2 calculations is always equal to the number of nodes in the multilinear grid, which becomes huge when adding new parameters or extending/refining the calculation domain.

— The storage size does not depend on the cross-section kind, it is the same for all cross-sections and equal to the number of nodes in the multilinear grid, always of the order of $\mathcal{O}(N^d)$.

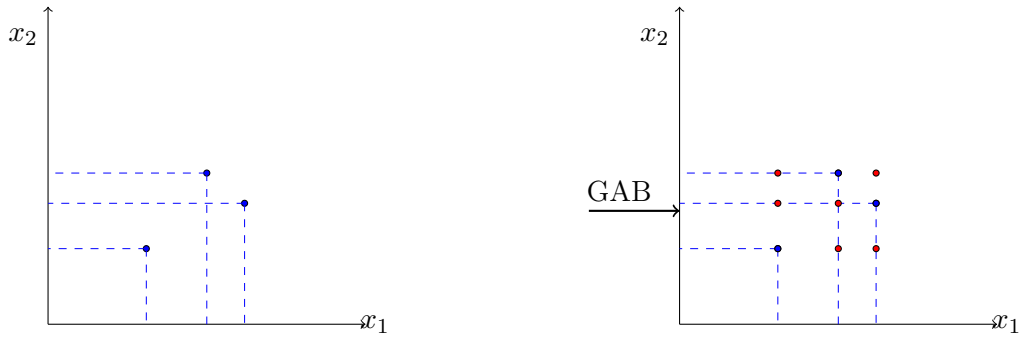
Because of the huge number of APOLLO2 calculations and the storage size in the neutron libraries, the multilinear interpolation model lacks of performance in some complex situations, e.g.:

incidental situations where new parameters need to be taken into account, and/or the calculation domain could be larger than the standard one.

Therefore, with industrial constraints, another model which allows us to overcome the limitations of the multilinear model (on the number of APOLLO2 calculations as well as the storage size) is required. The new model must also assure the accuracy of reconstructed cross-sections where for some cross-sections, this accuracy should be of the order of pcm with 1 pcm $= 10^{-5}$.

1.5.4 Computational constraints for the new reconstruction model

In our work, the GAB/APOLLO2 package is currently used to calculate cross-section values. With this package, APOLLO2 calculations are performed on a tensorized grid automatically generated from discretized values of each parameter. In order to avoid using a huge number of APOLLO2 calculations (that is very time-consuming), our model needs to be restricted in some situations. For an illustration in 2D, if we want to compute cross-sections on three points $\mathbf{x} = (x_1, x_2)$ where their three coordinates (per axis) are totally different, we have to use $3 \times 3 = 9$ APOLLO2 calculations via GAB, instead of only 3 which we need (see illustration in figure 1.15).



(a) 3 initial points requires 3 APOLLO2 calculations.

(b) Using GAB package, 9 APOLLO2 calculations (instead of 3) are used.

Figure 1.15 – GAB (always performed on a tensorized grid) is not flexible and waste of expensive APOLLO2 calculations for a reconstruction model which does not require input as a tensorized grid.

Therefore, in our current work, we have not yet realized some propositions, such as: performing our method on randomized points, constructing some set of points without tensorized structure.

Another point needs to be noted is that an APOLLO2 calculation provides at the same time values of all cross-section kinds at a given point in the parameter-phase space. The reason is that these cross-sections depend on the neutron flux calculated by APOLLO2 code and this flux is very expensive in calculation time. Therefore, each cross-section reconstruction must be "optimized" with the others in order to limit the number of APOLLO2 calculations.

Chapter 2

Mathematical background

This chapter aims at providing an overview of low-rank tensor approximations, classified as reduced order models. These models are well-suited to reduce the complexity of the parameter-dependent problems and can be applied to the approximation of multivariate functions (neutron cross-sections in our context). We recommend the reader to the references: [Hackbusch, 2012], [Nouy, 2016] and [Oseledets, 2011] for more details about different low-rank tensor approximations.

2.1 Problem statement

As presented in previous chapter (chapter 1), we need to reconstruct all cross-sections $(\sigma^g_{r,i}, \Sigma^g_{r,i})$ for each isotope i, energy g and cross-section kind r (here, $i \sim 50$ and $g \sim 2$, $r \sim 10$, leading about 1000 different cross-sections). Each cross sections is a multi-variate function depending on many physical parameters. Therefore, this reconstruction stands as an approximation of a set of multi-variate functions $\{f_k(\mathbf{x})\}_{k=1}^{k=K}$ defined over a domain $\mathbf{\Omega}$. Here, $\mathbf{\Omega} = \Omega_1 \times \ldots \times \Omega_d$ with $\Omega_{i,1\leq i\leq d} \subset \mathbb{R}$, d is the number of parameters and $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbf{\Omega}$. In our applications, the number of functions need to be reconstructed is $K \sim 10$ (if $f_k = \Sigma^g_r$) or $K \sim 1000$ (if $f_k = (\sigma^g_{r,i}, \Sigma^g_{r,i})_{r,i,g}$) and $d = 5$.

This reconstruction is based on 3 steps:

- Step 1 (offline): we need to evaluate, as precisely as possible, each function f_k at some points:

$$\{\mathbf{x}_{j(k)}\}_{j(k)=1}^{J(k)}$$

  For practical reason, the $\{\mathbf{x}_{j(k)}\}_{j(k)=1}^{J(k)}$ does not depend on k, i.e. $\{\mathbf{x}_{j(k)}\}_{j(k)=1}^{J(k)} \equiv \{\mathbf{x}_j\}_{j=1}^{J}$, $\forall k$. Each of these J computations is done with a software (in our case APOLLO2) that is quite expensive, hence J should be small.
- Step 2 (offline): From this acquisition of $\{f_k(\mathbf{x}_j)\}_{j=1}^{J}$, we must extract information that would allow to reconstruct approximatively each f_k at any point in $\mathbf{\Omega}$ - this information needs to

be stored and this storage should be as small as possible since each f_k is independent and requires a separate storage of data.

In addition, the information stored should be easily deduced from $\{f_k(\mathbf{x}_j)\}_{j=1}^{J}$.

- Step 3 (online): From each stored information, we evaluate f_k at any point $\mathbf{x}$ in $\mathbf{\Omega}$ that we want this evaluation should be as cheap as possible together and as accurate as possible, meaning that we want our approach to have a large [high accuracy]/[complexity] ratio.

To deal with this problem, we aim at proposing a model that allows us to:

(i) Offline: Acquire information efficiently, meaning that the cardinal J of the common set $\{\mathbf{x}_j\}_{j=1}^{J}$ used for $\{f_k\}_{k=1}^{k=K}$ should be as small as possible but information obtained is still at high quality.

(ii) Offline: Store data with less storage size possible since K, the total number of functions f_k, is big.

(iii) Online: Reconstruct each f_k with a large [high accuracy]/[complexity] ratio because the expected accuracy is of the order of 10^{-5} (or pcm).

In addition, we want to generalize our approach to $d > 5$ and to larger domains $\mathbf{\Omega}$.

2.2 Overview of different methods for the reconstruction of neutron cross-sections

The reconstruction of neutron cross-sections is a well-known problem in neutron simulation because these cross-sections belong to a high dimensional space (requiring high computational cost, large storage size). With the development of nuclear power in the world, many calculation chains for nuclear reactor core simulation have been developed, for instance: CASMO-SIMULATE [Edenius et al., 1986], [Rhodes et al., 2006], ARCADIA(HERMES)-ARTEMIS [Hobson et al., 2008], DRAGON-DONJON [Hébert, 2006], NEXUS-ANC [Müller et al., 2007], [Mayhue et al., 2006], etc. Inside each calculation chain, a reconstruction model for neutron cross-sections is employed. We can classify these reconstruction models into two main categories:

— The reconstruction of cross-sections is based on some perturbation or correction techniques applied to cross-section values calculated at or around the nominal point (a special point in the parameter-phase space on which the reactor operates normally). These techniques often rely on physical knowledges or some expansion techniques, such as the Taylor expansion. Some research works related to this reconstruction method can be found in: [Fujita et al., 2014], [Turski et al., 1997], [Müller et al., 2007], [Stålek and Demazière, 2008].

— The reconstruction of cross-sections is based on interpolation techniques applied to cross-section values calculated on a grid, e.g., tensorized grid [Watson and Ivanov, 2002], sparse grid [Danniëll and Pavel, 2014], quasi-random grid [Dufek, 2011], etc. These models are also different by the choice of basis functions: piece-wise linear functions, B-spline, Lagrange polynomials, etc.

The first category's methods have been proven suitable for the simulation on the standard domain where parameter-values are near the nominal condition. But far away from this condition, their accuracy is unknown since the heuristics used around nominal values may not be valid anymore. The second ones are more general but requires a lot of pre-calculated data to achieve high accuracy. Therefore, either we have highly efficient reconstruction method (meaning high accuracy with few point) but only applicable on a specified domain, or we have "expensive" reconstruction method (meaning high accuracy at the expense of a lot of pre-calculated points) but suitable for any domain.

From the lastest methods, we see that the multilinear interpolation can be considered as a good reference to compare with other reconstruction methods, in the following criteria:

— Offline: the number of pre-calculations used to calculate cross-section values in order to acquire useful information.

— Offline: the stored data size after the acquisition step.

— Online: the accuracy of cross-section evaluation.

To deal with this problem, we propose a new model, non-linear and ad'hoc, based on the Tucker format which belongs to the family of *low-rank tensor approximations*. The application is restricted to real multi-variate functions (neutron cross-sections in our context).

In next sections, we present these low-rank tensor methods and explain why we choose the Tucker format for the cross-section reconstruction problem.

2.3 Preview of low-rank tensors

Low-rank tensor methods are widely used to deal with high dimensional problems. Depending on the application, the two following frames should be distinguished:

— *Low-rank tensor representation (in linear algebra)*: tensors are *exactly* represented under various *formats*. These formats based on some notions of tensor ranks, such as: r-rank (a direct extension of matrix rank), α-rank (a rank obtained by a matricization process with respect to α, here α denotes a subset of the index set $D = \{1, \ldots, d\}$ with a given $d \in \mathbb{N}$). The term *representation* can be replaced by the term *format*.

— *Low-rank tensor approximation (in functional analysis)*: tensor space (or multivariate function space in our case) is equipped with a norm. Since the exact representation is often far

away to achieve in reality (due to its computational complexity), approximations with respect to a given norm is used to approach the exact representation. Low-rank tensor formats are again employed with some supplementary techniques, e.g. truncation.

2.4 Low-rank tensor representation

2.4.1 Singular value decomposition (SVD) for matrix

As we see later, matrix is a particular case of tensors in two-dimensional spaces. Some properties of matrices can not be directly extended to tensors but they are useful for many definitions related to tensors, particularly for the notion of rank and low-rank decomposition. We therefore recall here a well-known matrix factorization technique, called *singular value decomposition* (abbreviation: SVD), which allows us to:

— Represent a matrix M as a product of smaller size matrices but having the same rank of M (see later in (2.1)).

— Determine the rank of matrix M by the number of terms presented in its SVD decomposition.

The SVD is an extension of the eigenvalue decomposition for the case of non-symmetric matrices and non-square matrices, attributed by Erhard Schmidt [Schmidt, 1989] and Eckart and Young [Eckart and Young, 1936]. We consider here only the case in $\mathbb{R}$.

For a given rectangular matrix $M := M_{mn} = (m_{ij})_{\substack{1\leq i\leq m\\ 1\leq j\leq n}} \in \mathbb{R}^{m\times n}$, the SVD leads to an expression:

$$M_{mn} = U_{mm} S_{mn} V_{nn}^T$$

Where:

(i) $U := U_{mm} \in \mathbb{R}^{m\times m}$ and $V := V_{nn} \in \mathbb{R}^{n\times n}$

(ii) U, V are orthogonal matrices, i.e. $UU^T = I_m$ and $VV^T = I_n$.

(iii) Each column u_i of U corresponds to an orthonormal eigenvector of MM^T, i.e.: $U = [u_i]_{i=1}^{i=m}$ and $MM^T u_i = \lambda_i u_i$.
Each column v_j of V corresponds to an orthonormal eigenvector of $M^T M$, i.e.: $V = [v_j]_{j=1}^{j=n}$ and $M^T M v_j = \lambda_j v_j$.

(iv) The matrix $S := S_{mn} \in \mathbb{R}^{m\times n}$ is a diagonal matrix containing the square roots of eigenvalues of MM^T (they are also eigenvalues of $M^T M$):

$$(S)_{ij} = \begin{cases} \sqrt{\lambda_i} & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

The columns u_i of U are called *left singular vectors* of M and the columns v_j of V are called *right singular vectors* of M. The eigenvalues λ_k are called the *singular values* of M, they are sorted in the decreasing order: $\lambda_1 \geq \lambda_2 \ldots \geq \lambda_k \geq \ldots \geq 0$.

If $rank(M) = r$ $(r \leq \min\{m, n\})$ then $rank(S) = r$, i.e. $\lambda_k = 0$ with $k > r$, M can be represented as follows:

$$M_{mn} = U_{mr} S_{rr} V^T_{nr} = \sum_{i=1}^{r} \sqrt{\lambda_i} \underbrace{u_i}_{\in \mathbb{R}^m} \underbrace{v_i^T}_{\in \mathbb{R}^n} \tag{2.1}$$

where:

— U_{mr} is the matrix containing the first r columns of U: $u_1, u_2, \ldots, u_r$.

— S_{rr} is the matrix containing all $(= r)$ singular values different to 0 and taken from the matrix S.

— V_{nr} is the matrix containing the first r columns of V: $v_1, v_2, \ldots, v_r$.

— $u_i v_i^T$ is the product of the vector $u_i \in \mathbb{R}^m$ with the transpose v_i^T of the vector $v_i \in \mathbb{R}^n$.

Assuming that $m = n$, the initial storage cost of M is reduced from n^2 to $2nr$ by means of the representation (2.1).

For a given matrix M, (2.1) is a minimal representation of this matrix in the following sense:

$$r = \min\{s \in \mathbb{N}_0 |\ \exists \{\tilde{u}_i\}_{i=1}^s \subset \mathbb{R}^m, \exists \{\tilde{v}_i\}_{i=1}^s \subset \mathbb{R}^n : M = \sum_{i=1}^{s} \tilde{u}_i \tilde{v}_i^T\} \tag{2.2}$$

Now, if we want to "approximate" the matrix M with respect to some given norms $||.||$ in $\mathbb{R}^{m \times n}$, the singular value decomposition is again useful. Indeed, if we set $M_{\tilde{r}} = U_{m\tilde{r}} S_{\tilde{r}\tilde{r}} V^T_{\tilde{r}r}$ from (2.1) with $\tilde{r} \leq r$, we have $||M - M_{\tilde{r}}||_2 = \sqrt{\lambda_{\tilde{r}+1}}$ and $||M - M_{\tilde{r}}||_F = \sqrt{\sum_{i=\tilde{r}+1}^{min\{m,n\}} \lambda_i}$, where: $||.||_2$ is the spectral norm and $||.||_F$ is the Frobenius norm with the following definitions:

$$||M||_2 = \sqrt{\lambda_{max}(M^T M)} \text{ and } ||M||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} m_{ij}^2} = \sqrt{trace(M^T M)} \tag{2.3}$$

It thus appears that $M_{\tilde{r}}$ minimized $||M - \tilde{M}_{\tilde{r}}||$ for $||.||_2$ and $||.||_F$ over all $\tilde{r}$-rank matrices in $\mathbb{R}^{m \times n}$. Because $M \in \mathbb{R}^{m \times n}$ and the vector space $\mathbb{R}^{m \times n}$ has the finite dimension, all matrix norms are therefore equivalent. This approximation leads to the storage size of the matrix M is scaled from $\mathcal{O}(n^2)$ to $\mathcal{O}(n\tilde{r})$.

These properties of SVD will be useful for the construction of minimal subspaces in tensor approximation, when we extend matrix notion to tensor notion.

2.4.2 Tensor and tensor space

In this section, we restrict our presentation in finite dimension spaces over the field $\mathbb{R}$.

Tensors in finite-dimensional spaces can be considered as a generalization of array notion in two dimension entities (e.g: matrix) to multi-dimension entities. A tensor in a d-dimensional space is also called d-way array where d designates the order of the tensor.

Let V_j, $1 \leq j \leq d$, be finite-dimensional vector spaces over $\mathbb{R}$. An *elementary tensor* $\mathbf{v}$, defined by the so-called *tensor product* of d vectors v_j ($v_j \in V_j$), is written as follows:

$$\mathbf{v} = v_1 \otimes \ldots \otimes v_d = \bigotimes_{j=1}^{d} v_j,\ v_j \in V_j \tag{2.4}$$

Assuming that $dim(V_j) = n_j$ and $B_j = \{b_1^{(j)}, \ldots, b_{i_j}^{(j)}, \ldots, b_{n_j}^{(j)}\}$ is a basis of V_j, any $v_j \in V_j$ has thus the following representation:

$$v_j = \sum_{i_j=1}^{n_j} v_{j_{i_j}} b_{i_j}^{(j)}, v_{j_{i_j}} \in \mathbb{R}$$

Using this representation, the tensor $\mathbf{v}$ can be interpreted as a multidimensional array where each its entry is determined by a tuple index $i = [i_1, \ldots, i_d] \in \mathbf{N}^{n_1 \times \ldots \times n_d}$ ($1 \leq i_j \leq n_j$ and $1 \leq j \leq d$), as follows:

$$\mathbf{v} = v_1 \otimes \ldots \otimes v_d = \Big(\sum_{\substack{1 \leq i_1 \leq n_1 \\ 1 \leq j \leq d}} v_{1_{i_1}} b_{i_1}^{(j)}\Big) \otimes \ldots \otimes \Big(\sum_{\substack{1 \leq i_d \leq n_d \\ 1 \leq j \leq d}} v_{d_{i_d}} b_{i_d}^{(j)}\Big) = \sum_{\substack{1 \leq i_1 \leq n_1 \\ 1 \leq j \leq d}} \ldots \sum_{\substack{1 \leq i_d \leq n_d \\ 1 \leq j \leq d}} \prod_{j=1}^{d} v_{j_{i_j}} \bigotimes_{j=1}^{d} b_{i_j}^{(j)} \tag{2.5}$$

We now consider the space spanned by the elementary tensors. This space is called *tensor space* and its elements are called *tensors*. The tensor space is denoted as the tensor product of all vector spaces V_j:

$$\mathbf{V} = V_1 \otimes \ldots \otimes V_d = \bigotimes_{j=1}^{d} V_j = span\{\bigotimes_{j=1}^{d} v_j,\ v_j \in V_j\} \tag{2.6}$$

It is apparent that $\{\bigotimes_{j=1}^{d} b_{i_j}^{(j)},\ b_{i_j}^{(j)} \in B_j\}$ is a basis of the tensor space $\mathbf{V}$ and we have:

$$\mathbf{V} = span\{\bigotimes_{j=1}^{d} b_{i_j}^{(j)},\ b_{i_j}^{(j)} \in B_j\} \tag{2.7}$$

This leads to any tensor $\mathbf{v} \in \mathbf{V}$ possesses a representation called the *full representation*:

$$\mathbf{v} = \sum_{i_1=1}^{n_1} \ldots \sum_{i_d=1}^{n_d} \mathbf{a}_{i_1 \ldots i_d} \bigotimes_{j=1}^{d} b_{i_j}^{(j)},\ \text{with } \mathbf{a}_{i_1 \ldots i_d} \in \mathbb{R}, b_{i_j}^{(j)} \in B_j \tag{2.8}$$

Here, the array of coefficients $\mathbf{a} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ is called the *core tensor*.

If any vector space V_j, $j \in \{1, \ldots, d\}$, has the same dimension: $dim(V_j) = n_j = n$, the storage size (denoted by N_{mem}^{full}) of the full representation (2.8) will be:

$$N_{mem}^{\text{full}}(\mathbf{v}) = \underbrace{\sum_{j=1}^{d} n_j * size(b_{i_j}^{(j)})}_{\text{for the bases}} + \underbrace{\prod_{j=1}^{d} n_j}_{\text{for the core tensor}} = d * n^2 + n^d \tag{2.9}$$

We see that this representation depends exponentially on d (by the term n^d), due to the core tensor size. Such a complexity is often described by the term *curse of dimensionality*, introduced by Bellman in [Bellman, 1957], [Bellman, 1961]. In order to represent or approximate the tensor $\mathbf{v}$ with reduced storage size (the *curse of dimensionality* problem is scaled), low-rank tensor formats based on *tensor rank* notions are introduced.

2.4.3 Tensor rank

2.4.3.1 Tensor of order two

If we take $d = 2$ in the tensor space definition (2.6), the tensor space $\mathbf{V}$ becomes $\mathbf{V} = V_1 \otimes V_2$. A tensor $\mathbf{v} \in \mathbf{V}$ is now a *tensor of order two*. This particular case is considered here because any elementary tensor $\mathbf{v} = v_1 \otimes v_2$ in a finite-dimensional tensor space can be written as a matrix, denoted by $M(\mathbf{v})$, where $M(\mathbf{v}) = v_1 v_2^T \in \mathbb{R}^{n_1 \times n_2}$ (the tensor product $\otimes$ is replaced by the vector product). In the general case, a tensor $\mathbf{v}$ with the full representation $\sum_{i_1=1}^{n_1} \sum_{i_1=1}^{n_2} \mathbf{a}_{i_1 i_2} v_{i_1} \otimes v_{i_2}$, can be always represented under a matrix form as follows:

$$\underbrace{\mathbf{v}}_{\text{tensor}} = \sum_{i_1=1}^{n_1} \sum_{i_1=1}^{n_2} \mathbf{a}_{i_1 i_2} \underbrace{v_{i_1} \otimes v_{i_2}}_{\text{tensor product}} \longmapsto \sum_{i_1=1}^{n_1} \sum_{i_1=1}^{n_2} a_{i_1 i_2} \underbrace{v_{i_1} v_{i_2}^T}_{\text{dot product}} := \underbrace{M(\mathbf{v})}_{\text{matrix}} \in \mathbb{K}^{n_1 \times n_2}$$

In the inverse way, any matrix can be written as a tensor by its representation found in (2.1) issued from the SVD process. Thus, the notion of *rank* of a tensor of order two $\mathbf{v} \in V_1 \otimes V_2$ is introduced by the rank notion of the corresponding matrix $M(\mathbf{v})$, as follows:

$$rank(\mathbf{v}) = rank(M(\mathbf{v})) = \min\{r \in \mathbb{N}_0 | \, \exists v_i \in V_1, \exists w_i \in V_2 : \mathbf{v} = \sum_{i=1}^{r} v_i \otimes w_i\} \tag{2.10}$$

2.4.3.2 Tensor rank

The notion of tensor rank for tensors of order d is a natural extension of the rank notion introduced for tensors of order two. In order to get this notion, we define a set $\mathcal{R}_r$ ($r \in \mathbb{N}_0$) in the tensor space $\mathbf{V} = \bigotimes_{j=1}^{d} V_j$ containing linear combinations of r elementary tensors:

$$\mathcal{R}_r = \{\sum_{i=1}^{r} \bigotimes_{j=1}^{d} v_i^{(j)}, \, v_i^{(j)} \in V_j\} \tag{2.11}$$

The tensor rank of a tensor $\mathbf{v} \in \mathbf{V}$ is defined by:

$$rank(\mathbf{v}) = \min\{r \in \mathbb{N}_0 : \, \mathbf{v} \in \mathcal{R}_r\} \tag{2.12}$$

For a given r, we can characterize the set $\mathcal{R}_r$ by:

$$\mathcal{R}_r = \{\mathbf{v} \in \mathbf{V} : rank(\mathbf{v}) \leq r\} \tag{2.13}$$

2.4.3.3 Matricization and α-rank

We consider now another notion of rank for the tensor space $\mathbf{V} = \bigotimes_{j=1}^{d} V_j$ with $d \geq 3$. It is related to the subspace notion (tensor subspace or vector subspace) of $\mathbf{V}$ and V_j.

Let α be a subset of the set $D = \{1, \ldots, d\}$ such that: $\emptyset \neq \alpha \subset D$. We denote the complementary of α in the set D by α^c, i.e. $\alpha^c = D \setminus \alpha$. The tensor space $\mathbf{V} = \bigotimes_{j=1}^{d} V_j$ is then represented as $\mathbf{V} = \mathbf{V}_\alpha \bigotimes \mathbf{V}_{\alpha^c}$ with $\mathbf{V}_\alpha = \bigotimes_{j\in\alpha} V_j$ and $\mathbf{V}_{\alpha^c} = \bigotimes_{j\in\alpha^c} V_j$. Using this way, any tensor $\mathbf{v}$ is expressed as $\mathbf{v} \in V_\alpha \bigotimes V_{\alpha^c}$, which can be interpreted as a tensor of order two (see section 2.4.3.1). This process is called the *matricization* of $\mathbf{V}$ with respect to α, denoted by $\mathcal{M}_\alpha$. Such a matricization transforms a tensor $\mathbf{v}$ to a matrix, denoted by $\mathcal{M}_\alpha(\mathbf{v})$, where $\mathcal{M}_\alpha(\mathbf{v}) \in \mathbb{K}^{p\times q}$ with $p = \prod_{j\in\alpha} dim(V_j)$ and $q = \prod_{j\in\alpha^c} dim(V_j)$. The $rank_\alpha$ notion of a tensor $\mathbf{v}$ is therefore introduced as follows:

$$rank_\alpha(\mathbf{v}) = rank\mathcal{M}_\alpha(\mathbf{v}) \tag{2.14}$$

This notion is also equivalent to the following definition:

$$rank_\alpha(\mathbf{v}) = \min\{r \in \mathbb{N}_0 |\, \exists \mathbf{v}_i \in \mathbf{V}_\alpha, \exists \mathbf{w}_i \in \mathbf{V}_{\alpha^c} : \mathbf{v} = \sum_{i=1}^{r} \mathbf{v}_i \otimes \mathbf{w}_i\} \tag{2.15}$$

By means of the matrix rank property, we obtain $rank_\alpha(\mathbf{v}) = rank_{\alpha^c}(\mathbf{v})$. If α is a singleton $\{j\}$, we have $\mathbf{V}_\alpha := V_{\{j\}} = V_j$ and $rank_\alpha(\mathbf{v}) = rank_j(\mathbf{v})$.

Applying the SVD process (described in section 2.4.1) to the matrix $\mathcal{M}_\alpha(\mathbf{v})$, we can determine *minimal subspaces* $U_\alpha^{min}(\mathbf{v}) \subset V_\alpha$ and $U_{\alpha^c}^{min}(\mathbf{v}) \subset V_{\alpha^c}$ such that:

$$\mathcal{M}_\alpha(\mathbf{v}) \in U_\alpha^{min}(\mathbf{v}) \otimes U_{\alpha^c}^{min}(\mathbf{v}) \tag{2.16}$$

$$dim(U_\alpha^{min}(\mathbf{v})) = dim(U_{\alpha^c}^{min}(\mathbf{v})) = rank_\alpha(\mathbf{v}) \tag{2.17}$$

and $\mathbf{v}$ has a representation:

$$\begin{array}{c} \mathbf{v} = \sum_{i=1}^{rank_\alpha(\mathbf{v})} v_i \otimes w_i \\ \{v_i\}_{i=1}^{rank_\alpha(\mathbf{v})} : \text{ basis of } U_\alpha^{min}(\mathbf{v}),\ \{w_i\}_{i=1}^{rank_\alpha(\mathbf{v})} : \text{ basis of } U_{\alpha^c}^{min}(\mathbf{v}) \end{array} \tag{2.18}$$

The technique based on SVD to find $U_\alpha^{min}(\mathbf{v})$ ($\alpha \in D$) can be referred to as *higher-order singular value decomposition* (abbreviation: HOSVD).

2.4.4 r-term representation

r-term representation is proposed by Carroll [Carroll and Chang, 1970], Harshman [Harshman, 1970] and Comon [Comon et al., 2009]. Depending on different research fields, this representation is known as different names, such as: CANDECOMP/PARAFAC (CP) decomposition, canonical polyadic decomposition (CPD) or parallel factors model.

For a given $r \in \mathbb{N}_0$, if a tensor $\mathbf{v} \in \mathcal{R}_r \subset \mathbf{V} = \bigotimes_{j=1}^{d} V_j$ (see the definition of $\mathcal{R}_r$ in (2.13)), $\mathbf{v}$ possesses a r-term representation expressed by r elementary tensors as follows:

$$\mathbf{v} = \sum_{i=1}^{r} \otimes_{j=1}^{d} v_i^{(j)},\ v_i^{(j)} \in V_j \tag{2.19}$$

Presuming that for all j, $1 \leq j \leq d$, $dim(V_j) = n_j = n$, i.e. $size(v_i^{(j)}) = n$, $\forall v_i^{(j)} \in V_j$, the storage size of the r-term representation (2.19) is therefore equal to:

$$N_{mem}^{\text{r-term}}(\mathbf{v}) = \sum_{j=1}^{d} r * size(v_i^{(j)}) = \sum_{j=1}^{d} r * n = drn \tag{2.20}$$

depending linearly on d. However, this representation suffers from a possible numerical instability because the set of r-term tensors is not closed (see [de Silva and Lim, 2008]).

2.4.5 Tensor Subspace Representation (TSR)

Tensor Subspace Representation (TSR), also known as *Tucker format* is described in the chapter 8 of the book [Hackbusch, 2012] (page 217-248). The original idea is proposed by Tucker in [Tucker, 1966].

For a given tensor $\mathbf{v} \in \mathbf{V} = \bigotimes_{j=1}^{d} V_j$, the main idea of TSR is to find *subspaces* $U_j \subset V_j$ such that $\mathbf{v} \in \mathbf{U} = \bigotimes_{j=1}^{d} U_j$, see the illustration in figure 2.1. Here, $\mathbf{U}$ is called a tensor subspace of $\mathbf{V}$ (meaning subspace and tensor space).

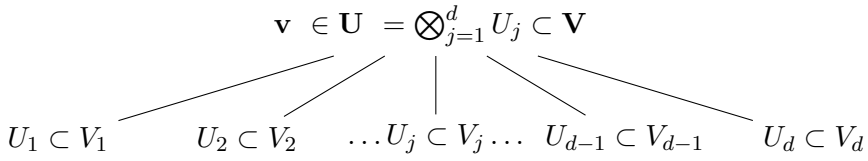


Figure 2.1 – Tensor subspace $\mathbf{U}$ used for the tensor subspace representation (also called Tucker format) of a tensor $\mathbf{v}$.

This representation is reserved for tensors $\mathbf{v}$ which belong to the following set $\mathcal{T}_{\mathbf{r}}$:

$$\mathcal{T}_{\mathbf{r}} = \mathcal{T}_{(r_1,\ldots,r_d)} = \{\mathbf{v} \in \mathbf{V} = \bigotimes_{j=1}^{d} V_j : rank_j(\mathbf{v}) \leq r_j\} \tag{2.21}$$

where $\mathbf{r} = (r_1, \ldots, r_d) \in \mathbb{N}_0^d$.

Assuming that $dim(V_j) = n_j$, $1 \leq j \leq d$, and V_j has a basis B_j. If $\mathbf{v} \in \mathcal{T}_{\mathbf{r}}$, there exist vector subspaces U_j of V_j such that $U_j = span\, B_{j,r_j}$ with $B_{j,r_j} = \{b_1^{(j)}, b_2^{(j)}, \ldots, b_{r_j}^{(j)}\} \subset B_j$, and $\mathbf{v}$ admits a $(r_1, \ldots, r_d)$-tensor subspace representation (or Tucker format) as follows:

$$\mathbf{v} = \sum_{i_1=1}^{r_1} \ldots \sum_{i_d=1}^{r_d} \mathbf{a}_{i_1,\ldots,i_d} b_{i_1}^{(1)} \otimes \ldots \otimes b_{i_d}^{(d)}, \; (\, b_{i_j}^{(j)} \in B_{j,r_j}) \tag{2.22}$$

In the case where $r_j = r$ and $dim(V_j) = n$ for all $j \in \{1, \ldots, d\}$, the storage size of the Tucker format for the tensor $\mathbf{v}$ is equal to:

$$N_{mem}^{\mathrm{TSR}}(\mathbf{v}) = \underbrace{\sum_{j=1}^{d} r_j * \underbrace{size(b_{i_j}^{(j)})}_{=n}}_{\text{for bases}} + \underbrace{\prod_{j=1}^{d} r_j}_{\text{for core tensor}} = drn + r^d$$

This storage size still depends exponentially on the dimension d but scaled from n^d to r^d, compared to the full representation (2.9).

2.4.6 Hierarchical Tensor Representation (HTR)

The Hierarchical Tensor Representation (HTR) can be found in [Hackbusch and Kühn, 2009] and the chapter 11 of Hackbusch's book [Hackbusch, 2012]. The main idea is to keep the subspace structure but avoiding the dimensionality problem. This format also appeared under the name "$\mathcal{H}$-Tucker format", in order to describe that the subspace idea of the Tucker format is recursively used [Grasedyck, 2010], [Kressner and Tobler, 2012].

Let us first introduce some notions used for the HTR. We say T_D is a *dimension partition tree* of the set $D = \{1, \ldots, d\}$ if T_D is a binary tree, i.e. each node $\alpha \in T_D$ has 0 or 2 sons α_1, α_2 (see the illustration in figure 2.2).

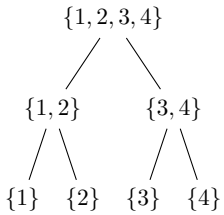


Figure 2.2 – A dimension partition tree (balanced tree) for $D = \{1, 2, 3, 4\}$.

In such a tree, we distinguish:

(i) The root α where $\alpha = D = \{1, \ldots, d\}$.

(ii) The leaves α where $\alpha = \{j\}$, $j \in D$. We denote by $\mathcal{L}(T_D)$ the set containing all the leaves:

$$\mathcal{L}(T_D) = \{\{j\},\, j \in D\}$$

(iii) The non-leaf and non-root vertices α, with $\alpha \in T_D \backslash \{\mathcal{L}(T_D) \bigcup D\}$.

For a given dimension partition tree T_D, we construct a corresponding *tensor space partition* for $\mathbf{V} = \bigotimes_{j=1}^{d} V_j$. From this partition, we determine subspaces $\mathbf{U}_{\alpha,\alpha\in T_D}$ such that:

(i) If $\alpha = D$ (root of the tree) then $\mathbf{v} \in \mathbf{U}_D \subset \mathbf{V}$, ($\mathbf{U}_D$: tensor subspace of $\mathbf{V}$).

(ii) If $\alpha \in T_D \backslash \{\mathcal{L}(T_D) \bigcup D\}$ (non-leaf and non-root of the tree) then $\mathbf{U}_\alpha \subset \mathbf{U}_{\alpha_1} \bigotimes \mathbf{U}_{\alpha_2} \subset \mathbf{V}_\alpha$ with α_1 *and* α_2 *are the two sons of* α, ($\mathbf{U}_\alpha$: tensor subspace of $\mathbf{V}_\alpha$).

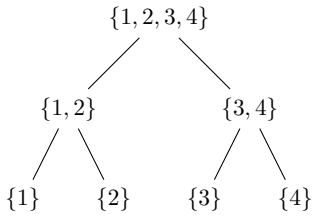
(iii) If $\alpha = \{j\} \in \mathcal{L}(T_D)$ then $U_j \subset V_j$, $1 \leq j \leq d$, (U_j: vector subspace of V_j).

We illustrate in figure 2.3 the process which provides us the path to construct the subspaces U_j and $\mathbf{U}_\alpha$ in the case $D = \{1, 2, 3, 4\}$:

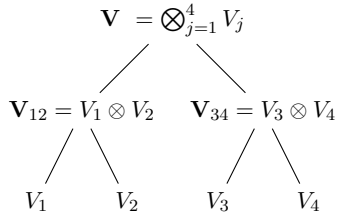
a) Given a balanced partition of D by the tree T_D (figure 2.3a).

b) We split the tensor space $\mathbf{V}$ by the same partition way in T_D (figure 2.3b).

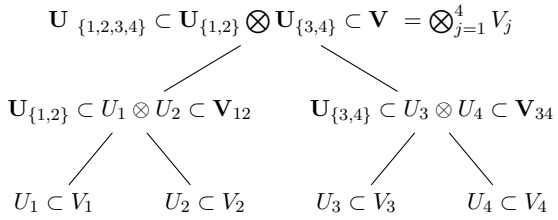
c) We find for each vertex α, a subspace $\mathbf{U}_\alpha \subset \mathbf{V}_\alpha$ as described in figure 2.3c.



(a) A dimension partition tree T_D.



(b) The corresponding tensor-space partition on $\mathbf{V}$.



(c) The corresponding vector subspaces $U_j \subset V_j$ on leaves $j \in D = \{1, \ldots, d\}$ and tensor subspaces $\mathbf{U}_\alpha \subset \mathbf{V}_\alpha$, $\alpha \in T_D \backslash \mathcal{L}(T_D)$, used for the hierarchical tensor format.

Figure 2.3 – The process to construct a Hierarchical Tensor Representation.

Therfore, for a given tree T_D, the HTR associated with this tree is used for the tensors $\mathbf{v}$ which belong to the following set:

$$\mathcal{H}_{\mathbf{r}}^{T_D} = \{\mathbf{v} \in \mathbf{V} = \bigotimes_{j=1}^{d} V_j : rank_\alpha(\mathbf{v}) \leq r_\alpha,\, \alpha \in T_D\} \tag{2.23}$$

here, $\mathbf{r} = (r_\alpha)_{\alpha \in T_D} \in \mathbb{N}_0^{\#(T_D)}$ with $\#(T_D) \leqslant 2d-1$.

The main process used in the HTR is to construct the basis for each tensor subspace $\mathbf{U}_\alpha$ (α is not a leaf) from the bases of subspaces U_j on the leaves j. This construction is recursively performed from the root level to the leaf level by applying the Tucker format to each basis element of a tensor subspace $\mathbf{U}_\alpha$. Once the basis of the tensor subspace $\mathbf{U}_D$ at the root is determined by using this way, the tensor $\mathbf{v} \in \mathbf{U}_D \subset \mathbf{V}$ will be expressed by a linear combination of the basis elements in $\mathbf{U}_D$.

Concretely, we assume that $dim(\mathbf{U}_\alpha) = r_\alpha$ here α is a non-leaf vertex and $B_\alpha = \{\mathbf{b}_{l[\alpha]}^{(\alpha)}\}_{1 \leq l[\alpha] \leq r_\alpha}$ is the basis of $\mathbf{U}_\alpha$. The Tucker format is recursively applied to $\{\mathbf{b}_{l[\alpha]}^{(\alpha)}\}_{\alpha \in T_D \setminus \mathcal{L}(T_D)}$ until the bases $B_{\{j\}}$ (bases of the leaves), as follows:

$$\mathbf{b}_{l[\alpha]}^{(\alpha)} = \sum_{l[\alpha_1]=1}^{r_{\alpha_1}} \sum_{l[\alpha_2]=1}^{r_{\alpha_2}} c_{l[\alpha_1],l[\alpha_2]}^{(\alpha,l[\alpha])} \mathbf{b}_{l[\alpha_1]}^{(\alpha_1)} \otimes \mathbf{b}_{l[\alpha_2]}^{(\alpha_2)}, \forall \alpha \in T_D \setminus \mathcal{L}(T_D), \text{ with } \alpha_1, \alpha_2 : \text{sons of } \alpha \tag{2.24}$$

where the matrix $C_{\alpha,l[\alpha]} = (c_{l[\alpha_1],l[\alpha_2]}^{(\alpha,l[\alpha])}) \in \mathbb{R}^{r_{\alpha_1} \times r_{\alpha_2}}$, with $\alpha \in T_D \setminus \mathcal{L}(T_D)$, $1 \leq l[\alpha] \leq r_\alpha$. We continue to replace $\mathbf{b}_{l[\alpha_1]}^{(\alpha_1)}$ and $\mathbf{b}_{l[\alpha_2]}^{(\alpha_2)}$ with the formula (2.24) and obtain:

$$\mathbf{b}_{l[\alpha]}^{(\alpha)} = \sum_{\substack{l[\beta]=1 \\ \beta \in T_\alpha \setminus \mathcal{L}(T_\alpha)}}^{r_\beta} \prod_{\beta \in T_\alpha \setminus \mathcal{L}(T_\alpha)} c_{l[\beta_1],l[\beta_2]}^{(\beta,l[\beta])} \bigotimes_{j \in \alpha} b_{l[\{j\}]}^{(j)}, \text{ with } \beta_1, \beta_2 : \text{sons of } \beta \tag{2.25}$$

where T_α is the sub-tree of T_D with root α and $\mathcal{L}(T_\alpha) = T_\alpha \bigcap \mathcal{L}(T_D)$.

Because $\mathbf{v} \in \mathbf{U}_D$, we therefore have:

$$\mathbf{v} = \sum_{l[D]=1}^{r_D} C_{l[D]}^{(D)} \mathbf{b}_{l[D]}^{(D)} \tag{2.26}$$

We insert (2.25) with $\alpha = D$ into (2.26), we finally have:

$$\mathbf{v} = \sum_{l[D]=1}^{r_D} C_{l[D]}^{(D)} \Big(\sum_{\substack{l[\beta]=1 \\ \beta \in T_D \setminus \mathcal{L}(T_D)}}^{r_\beta} \prod_{\beta \in T_D \setminus \mathcal{L}(T_D)} c_{l[\beta_1],l[\beta_2]}^{(\beta,l[\beta])} \bigotimes_{j=1}^{d} b_{l[\{j\}]}^{(j)} \Big), \text{ with } \beta_1, \beta_2 : \text{sons of } \beta \tag{2.27}$$

Since $\#(T_D) \leq 2d-1$ and $size(C^{(D)}) = r_D$ can be neglected, the storage size of the HTR is dominated by:

$$N_{mem}^{\text{TSR}}(\mathbf{v}) = size(C_\alpha)_{\alpha \in T_D \setminus \mathcal{L}(T_D)} + size(\{b_{l[\{j\}]}^{(j)}\}_{\substack{j \in D \\ 1 \leq l[\{j\}] \leq r_j}}) \tag{2.28}$$

$$= \sum_{\alpha \in T_D \setminus \mathcal{L}(T_D)} r_\alpha r_{\alpha_1} r_{\alpha_2} + \sum_{j=1}^{d} r_j size(b_{l[\{j\}]}^{(j)}) \tag{2.29}$$

$$= \sum_{\alpha \in T_D \setminus \mathcal{L}(T_D)} r_\alpha r_{\alpha_1} r_{\alpha_2} + \sum_{j=1}^{d} r_j n_j \tag{2.30}$$

If we assume that $r_\alpha = r$ for all $\alpha \in T_D$ and $n_j = dim(V_j) = n$ for all $j \in \{1, \dots, d\}$, we obtain:

$$N_{mem}^{\mathrm{TSR}}(\mathbf{v}) \leqslant (d-1)r^3 + dnr$$

(because the number of non-leaf vertices in T_D is less than $d-1$). We see that the storage size of HTR linearly depends on d.

2.4.7 Tensor Train Representation (TTR)

Tensor Train Representation (TTR) [Oseledets, 2011] is a special form of the hierarchical format, corresponding to an unbalanced tree of type $TT_D = D \cup \{\{j\}, \{j+1, \dots, d\}, 1 \leq j \leq d-1\}$, see figure 2.4 for example.

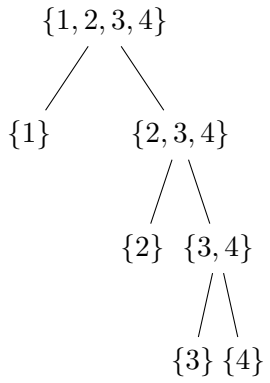


Figure 2.4 – A dimension partition tree TT_D used for Tensor Train Representation with $D = \{1, 2, 3, 4\}$.

For a given tree TT_D, the TTR is reserved for tensors $\mathbf{v}$ in the following set:

$$\mathcal{T}_{\mathbf{r}}^{TT_D} = \{\mathbf{v} \in \mathbf{V} = \bigotimes_{j=1}^{d} V_j : rank_{\underbrace{\{j+1, \dots, d\}}_{\in TT_D}}(\mathbf{v}) \leq r_j,\ 1 \leq j \leq d-1\} \tag{2.31}$$

with $\mathbf{r} = (r_j)_{1 \leq j \leq d-1} \in \mathbb{N}_0^{d-1}$.

Since $rank_\alpha(\mathbf{v}) = rank_{\alpha^c}(\mathbf{v})$, we therefore can write the set $\mathcal{T}_{\mathbf{r}}^{TT_D}$ as:

$$\mathcal{T}_{\mathbf{r}}^{TT_D} = \{\mathbf{v} : rank_{\underbrace{\{j+1, \dots, d\}}_{\in TT_D}}(\mathbf{v}) = rank_{\underbrace{\{1, \dots, j\}}_{\notin TT_D}}(\mathbf{v}) \leq r_j,\ 1 \leq j \leq d-1\} \tag{2.32}$$

In the TTR, we apply successively the representation like in (2.18) to minimal subspaces of two sons of each $\alpha = \{j+1, \dots, d\} \in TT_D \backslash \mathcal{L}(D)$. Due to the structure of the tree TT_D, for each j, we denote by j_c the set $\{j+1, \dots, d\}$. This notation is useful to describe the TTR formula.

We start with the root where $\alpha = D$. Since D has two sons: $\{1\}$, $1_c := \{2, \ldots, d\}$ and $rank_{\{2,\ldots,d\}} \leq r_1$ (definition (2.32)), we obtain:

$$\mathbf{v} = \sum_{i_1=1}^{r_1} \underbrace{v_{i_1}^{(1)}}_{\in U_1^{min}(\mathbf{v})} \otimes \underbrace{\mathbf{w}_{i_1}^{(1_c)}}_{\in \mathbf{U}_{\{2,\ldots,d\}}^{min}(\mathbf{v})} \tag{2.33}$$

Because $\{2, \ldots, d\}$ has two sons: $\{2\}$ and $2_c := \{3, \ldots, d\}$ and $rank_{\{3,\ldots,d\}} = r_2$ (definition (2.32)), so that $\mathbf{w}_{i_1}^{(1^c)}$ can be represented as:

$$\mathbf{w}_{i_1}^{(1^c)} = \sum_{i_2=1}^{r_2} \underbrace{v_{i_1,i_2}^{(2)}}_{\in V_2} \otimes \underbrace{\mathbf{w}_{i_2}^{(2_c)}}_{\in \mathbf{U}_{\{3,\ldots,d\}}^{min}(\mathbf{v})} \tag{2.34}$$

We insert $\mathbf{w}_{i_1}^{(1^c)}$ in (2.34) to (2.33), we obtain:

$$\mathbf{v} = \sum_{i_1=1}^{r_1} \sum_{i_2=1}^{r_2} \underbrace{v_{i_1}^{(1)}}_{\in U_1^{min}(\mathbf{v})} \otimes \underbrace{v_{i_1,i_2}^{(2)}}_{\in V_2} \otimes \underbrace{\mathbf{w}_{i_2}^{(2_c)}}_{\in \mathbf{U}_{\{3,\ldots,d\}}^{min}(\mathbf{v})} \tag{2.35}$$

This process continues and we have:

$$\mathbf{v} = \sum_{i_1=1}^{r_1} \sum_{i_2=1}^{r_2} \cdots \sum_{i_{d-2}=1}^{r_{d-2}} \sum_{i_{d-1}=1}^{r_{d-1}} \underbrace{v_{i_1}^{(1)}}_{\in U_1^{min}(\mathbf{v}) \subset V_1} \otimes \underbrace{v_{i_1,i_2}^{(2)}}_{\in V_2} \otimes \ldots \otimes \underbrace{v_{i_{d-2},i_{d-1}}^{(d-1)}}_{\in V_{(d-1)}} \otimes \underbrace{\mathbf{w}_{i_{d-1}}^{((d-1)_c)}}_{\in U_d^{min}(\mathbf{v}) \subset V_d} \tag{2.36}$$

The last term expressed as $\mathbf{w}_{i_{d-1}}^{((d-1)_c)} \in U_d^{min}(\mathbf{v}) \subset V_d$, so that we must have:

$$\mathbf{w}_{i_{d-1}}^{((d-1)_c)} \equiv v_{i_{d-1}}^{(d)} \in U_d^{min}(\mathbf{v}) \subset V_d$$

Therefore, $\mathbf{v}$ finally has the TTR as follows:

$$\mathbf{v} = \sum_{i_1=1}^{r_1} \sum_{i_2=1}^{r_2} \cdots \sum_{i_{d-2}=1}^{r_{d-2}} \sum_{i_{d-1}=1}^{r_{d-1}} \underbrace{v_{i_1}^{(1)}}_{\in V_1} \otimes \underbrace{v_{i_1,i_2}^{(2)}}_{\in V_2} \otimes \ldots \otimes \underbrace{v_{i_{d-2},i_{d-1}}^{(d-1)}}_{\in V_{d-1}} \otimes \underbrace{v_{i_{d-1}}^{(d)}}_{\in V_d} \tag{2.37}$$

The storage size of TTR for a tensor $\mathbf{v}$ is denoted by $N_{mem}^{\mathrm{TTR}}(\mathbf{v})$ and decomposed as follows:

$$\begin{aligned} N_{mem}^{\mathrm{TTR}}(\mathbf{v}) =& size(\{\underbrace{v_{i_1}^{(1)}}_{\in V_1}\}_{1 \leq i_1 \leq r_1}) \\ &+ \sum_{2 \leq j \leq d-1} size(\{\underbrace{v_{i_{j-1},i_j}^{(j)}}_{\in V_j}\}_{\substack{1 \leq i_j \leq r_j \\ 1 \leq i_{j-1} \leq r_{j-1}}}) \\ &+ size(\{\underbrace{v_{i_{d-1}}^{(d)}}_{\in V_d}\}_{1 \leq i_{d-1} \leq r_{d-1}}) \end{aligned}$$

Therefore, we obtain:

$$\begin{aligned} N_{mem}^{\mathrm{TTR}}(\mathbf{v}) =& r_1 size(\underbrace{v_{i_1}^{(1)}}_{\in V_1}) + \sum_{j=2}^{d-1} r_{j-1} r_j size(\underbrace{v_{i_{j-1},i_j}^{(j)}}_{\in V_j}) + r_{d-1} size(\underbrace{v_{i_{d-1}}^{(d)}}_{\in V_d}) \\ =& r_1 n_1 + \sum_{j=2}^{d-1} r_{j-1} r_j n_j + r_{d-1} n_d \\ =& 2rn + (d-2)r^2 n \quad (\text{if } n_j = n, r_j = r, \forall j \in \{1, \ldots, d\}) \end{aligned} \tag{2.38}$$

This storage size of TTR linearly depends on d.

2.4.8 Summary

In the lemma 8.6 (page 219) and the lemma 11.55 (page 353) of [Hackbusch, 2012], the author showed that the set $\mathcal{T}_r$ and $\mathcal{H}_r$ are closed (i.e., in the sense: U is closed if for any sequence $(v_n)_{n\in\mathbb{N}} \subset U \subset V$ which converges to v, then $v \in U$). Therefore, we summarize in table 2.1 some properties of the presented formats.

	r-term	Tucker format	Hierarchical format
Complexity	$\mathcal{O}(rnd)$	$\mathcal{O}(rnd + r^d)$	HTR : $\mathcal{O}(d-1)r^3 + dnr)$ TTR : $\mathcal{O}(2rn + (d-2)r^2 n)$
Closedness	No	Yes	Yes

Table 2.1 – Comparison of different low-rank tensor formats.

2.5 Low-rank tensor approximation

2.5.1 Tensor approximation in general

Now, we go from exact representations (in linear algebra) to approximate representations (in functional analysis). The classification of low-rank tensor formats allows us to exactly represent a tensor $\mathbf{v} \in \mathbf{V} = \bigotimes_{j=1}^{d} V_j$ with some specific tensor structures, as previously described. In order to reduce the storage size and computational operations for $\mathbf{v}$, this tensor can be approximated by another tensor $\mathbf{u} \in \mathbf{V}$ such that $\mathbf{v} \approx \mathbf{u}$, with respect to a norm $||.||_{\mathbf{V}}$ defined on $\mathbf{V}$. The low-rank tensor formats should be used again in order to find $\mathbf{u}$. In general, *the reduction of computational cost often leads to the increase of approximation error* $||\mathbf{v} - \mathbf{u}||_{\mathbf{V}}$.

In previous sections, the complexity of each tensor format related to storage size was presented. However, two other questions are raised in practice:

(i) How to compute components of those formats?

(ii) How to use them in a concrete situation?

Since the SVD in linear algebra can be considered as the Karhunen-Loève decomposition in functional analysis, they become useful and efficient tools to answer the two above questions.

2.5.2 Application to our problem

2.5.2.1 Low-rank tensor approximation for multivariate functions

In our problem, neutron cross-sections are considered as d-variate functions defined on the domain $\mathbf{\Omega} = \Omega_1 \times \ldots \times \Omega_d$ with $\Omega_j \subset \mathbb{R}$, $1 \leq j \leq d$. The evaluation of such a function often requires a grid constructed on $\Omega_1 \times \ldots \times \Omega_d$. The classical method is the multilinear interpolation where the grid has a tensor structure. Since this method requires the complexity of the order of $\mathcal{O}(n^d)$ (with n is the number of discretized points per axis for this grid), we hope to find another tensor structure for such evaluation with reduced cost. Therefore, low-rank tensor can be applied to the multivariate function approximations.

In such an application, the tensor product of d univariate functions $f_j(x_j), 1 \leq j \leq d$, is a d-variate function:

$$(\bigotimes_{j=1}^{d} f_j)(x_1, \ldots, x_d) = \prod_{j=1}^{d} f_j(x_j)$$

2.5.2.2 Model choice

In our problem for the reconstruction (or approximation) of neutron cross-sections, the accuracy is very important while the dimension d is not very large ($d = 5, 6$). Hence, we are more interested in the Tucker format (2.22) than others (e.g. HTR, TTR), because of its promising accuracy, even the complexity of the this format is still high due to the term r^d (see table 2.1). Furthermore, the term r^d is acceptable in our problem because $d \sim 5, 6$ and by an approximation process (e.g. truncation), we can obtain r small in general ($r \ll n = dim(V_j)$, $\forall j$). In addition, as we see later in the chapter 5, we can further reduce this complexity by sparse representation techniques. Noting that the complexity of the multilinear interpolation (currently used for the reconstruction of cross-sections) is known as in the order of $\mathcal{O}(n^d)$.

Although HTR is an interesting format, we need to condense the parameters on which cross-sections depend into certain groups. This requires a best knowledge about the correlation between

these physical parameters in neutronics, that we have not yet determined until now. We therefore did not choose the HTR format in our work.

2.5.2.3 Tucker decomposition-proposed model

In our problem, the Tucker format is used and referred to as *Tucker decomposition*, it is written as follows:

$$f(x_1,\ldots,x_d) \approx \tilde{f}(x_1,\ldots,x_d) = \sum_{i_1=1}^{r_1} \cdots \sum_{i_d=1}^{r_d} \mathbf{a}_{i_1\ldots i_d} \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_j) \tag{2.39}$$

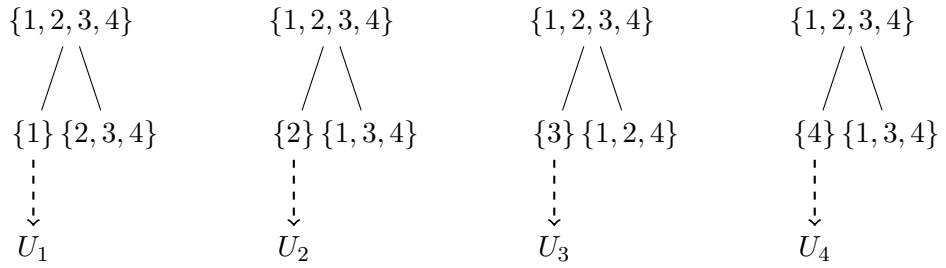
In this decomposition, we need to determine:

(i) The functions $\{\varphi_{i_j}^{(j)}\}_{i_j=1}^{r_j}$ for each direction j, referred to as *tensor directional basis function.*

(ii) The coefficients $\mathbf{a}_i := \mathbf{a}_{i_1\ldots i_d}$.

To deal with the problem (i), we relied on the HOSVD process which is described as an extension of the Karhunen-decomposition in our work. Such a process corresponds to a matricization applied each time to a couple $(\alpha = \{j\}, \alpha^c = \{1,\ldots,d\} \setminus \{j\})$ to construct tensor directional basis functions for each direction j (see the illustration of this process in figure 2.5).

$\mathbf{U} = \bigotimes_{j=1}^{4} U_j \subset \bigotimes_{j=1}^{4} V_j$ where:



(a) U_1 subspace (b) U_2 subspace (c) U_3 subspace (d) U_4 subspace

Figure 2.5 – Construction of subspaces U_j for the tensor subspace $\mathbf{U} = \bigotimes_{j=1}^{d} U_j$ used in the Tucker decomposition.

For the problem (ii) mentioned above, we determine the coefficients by solving a system of linear equations where the greedy algorithm is employed to select the points required by this system.

In order to avoid the repetition, we propose the reader to see the descriptions and the applications of the Tucker decomposition with more details in next chapters. We would like to specify that almost next chapters correspond to our journal papers written in this thesis.

Part II

Proposed model and numerical results

Chapter 3

Tucker decomposition model for the reconstruction of neutron cross-sections

This is a submitted paper with Y.Maday, M.Guillo and P.Guérin. Its reference in the manuscript is [Luu et al., 2016b].

Abstract

The full representation of a d-variate function requires exponentially storage size as a function of dimension d and high computational cost. In order to reduce these complexities, function approximation methods (called *reconstruction* in our context) are proposed, such as: interpolation, approximation, etc. The traditional interpolation model like the multilinear one, has this dimensionality problem. To deal with this problem, we propose a new model based on the Tucker format - a low-rank tensor approximation method, called here *the Tucker decomposition*. The Tucker decomposition is built as a tensor product of one-dimensional spaces where their one-variate basis functions are constructed by an extension of the Karhunen-Loève decomposition into high-dimensional space. Using this technique, we can acquire, direction by direction, the most important information of the function and convert it into a small number of basis functions. Hence, the approximation for a given function needs less data than that of the multilinear model. Results of a test case on the neutron cross-section reconstruction demonstrate that the Tucker decomposition achieves a better accuracy while using less data than the multilinear interpolation.

Keywords: function approximation (reconstruction), low-rank tensor approximation, Tucker decomposition, Karhunen-Loève decomposition, cross-sections, neutronics

3.1 Introduction

The concept of "function approximation" is widely used in many branches of applied mathematics and computer science to apprehend quantities of interest that depend on, or a function of input parameters, such as spatial position, time evolution or any other type of input quantity. The a priori knowledge of the quantity of interest may either be "explicit", as coming out from measurements, or "implicit" as solution of a modeling equation. There is a third way that is relevant to this paper. Cross-sections that feed the neutron flux solver cannot be directly measured in an environment such as a nuclear reactor core. They neither can be computed using a single equation since there is an interaction between many physics (neutronics, hydraulic, thermic, ...). The way to solve this situation is through a simplified "experimental simulation" and we will call later on this process a "calculation scheme". Since cross-sections depend continuously on many parameters, a parametrized calculation scheme is also very useful to emulate many different "experiments" with different configurations. Therefore, one calculation point corresponds to one experiment for a given configuration. That explains why, although cross-sections are computed and not measured, we will consider them as "explicit" data acquired through "experimental simulation".

Multipurpose/universal approaches such as global or piecewise polynomial approximations are general approximation methods that are valid for a large variety of functions: depending on the hypothesis/knowledge we have on the function such as regularity in terms of existence of a certain number of derivatives, we may prefer to use global polynomial versus piecewise ones. This step of the approximation requires some know-how that is now rather well understood. Once the approximation basis set is chosen, the coefficients or components of the function we are interested in and we want to approximate in this basis set are determined in order to fit with the "input" that are explicitly available and have been acquired by some series of measurements: the stability of the mapping from the input to the coefficients or components that is an important feature for the quality of the approximation also depends on the chosen basis set. Once this is done, a second step in the approximation is the reconstruction (or evaluation) of the function we are interested in, in other points than those that have been used to construct the approximation.

All this framework involves four different concepts: i) data acquisition, ii) storage of these data, iii) reconstruction of the function we are interested in, iv) further evaluation, that all have their particular complexity and cost, that, of course depend on the number of the inputs that are used to define the function of interest. These complexity and cost suffer from what is known as the curse of dimension that leads to an exponential explosion of the complexity and cost with respect to the number of inputs.

In order to face this particular problem, different ad'hoc strategies have been proposed and leave away the notion of linear approximation in multipurpose/universal representation spaces for prefer-

ring nonlinear, adapted representations. This enters in the concept of model reduction approaches.

Our interested in the research presented in this paper comes from the particular application we have in mind which is the reconstruction of cross-sections in neutronics.

In neutronics, cross-sections are used to represent the probability of interaction between an incident neutron and a given target nuclei ([Marguet, 2013]). These cross-sections are inputs for the Boltzmann equation ([Harris, 2004]) that describe the neutron population in the core. They depend on various parameters which characterize the local material behavior. Among the parameters stand for instance: *i) burnup, ii) fuel temperature, iii) moderator density, iv) boron concentration v) xenon level, ...* These are 5 parameters that we are interested in in this paper but there may be many more leading to larger values of d. They are denoted by $\mathbf{x} = (x_1, \ldots, x_d)$, where here $d = 5$ and they vary in a space called *parameter-phase space*; hence cross-sections are multivariate functions of $\mathbf{x}$, $\mathbf{x} \in$ *parameter-phase space*.

There are different cross-section kinds that represent different aspects of the physics involved (fission, absorption, scattering ...). These different kinds of *reaction* are indexed by *" r"*.

The cross-sections also depend on the energy of the incident neutron, this energy is discretized through "groups" and we designate by the exponent *"g"* the incident discretized energy group. *Microscopic cross-sections* (σ) depend also on the target nuclei (or isotope), designated by *"i"*. Therefore, $\{\sigma^g_{r,i}\}$ stands for these microscopic cross-sections.

Macroscopic cross-sections (Σ) that feed the neutron flux solver are related to above quoted microscopic ones using a formula such as:

$$\Sigma^g_r = \sum_{i=1}^{I} c_i \sigma^g_{r,i} \tag{3.1}$$

where c_i is the concentration of isotope i.

For EDF's applications, $I \sim 50$ isotopes, $g = 2$ energy groups and $r \sim 10$ reactions, we obtain already one thousand types of microscopic cross-sections, i.e one thousand multivariate functions to approximate.

In the current core simulations, the core is described as a full three-dimensional object $\subset \mathbb{R}^3$. At each different position P in the core, we have specific thermo-hydraulic-state conditions, leading to a corresponding value of $\mathbf{x} = (x_1, \ldots, x_d)$ in the parameter-phase space. It means that the cross-sections $\{\sigma^g_{r,i}\}$, $\{\Sigma^g_r\}$ that are functions of $\mathbf{x}$ thus depend (implicitely) on the position P in the core since $\mathbf{x}$ is a function of P: $P \in \mathbb{R}^3 \mapsto \mathbf{x}(P)$.

In practice, the core is discretized into cells. Therefore, $\mathbf{x}$ depends now on the position k of a cell, $\mathbf{x} = \mathbf{x}(\text{cell}_k)$. Hence, cross-sections need to be determined cell per cell (see figure 3.1).

In the core simulation, we need accurate values for the various cross-sections at all cells (about $200,000$ cells for industrial cases and 10 times more for "reference" cases). This leads to the number

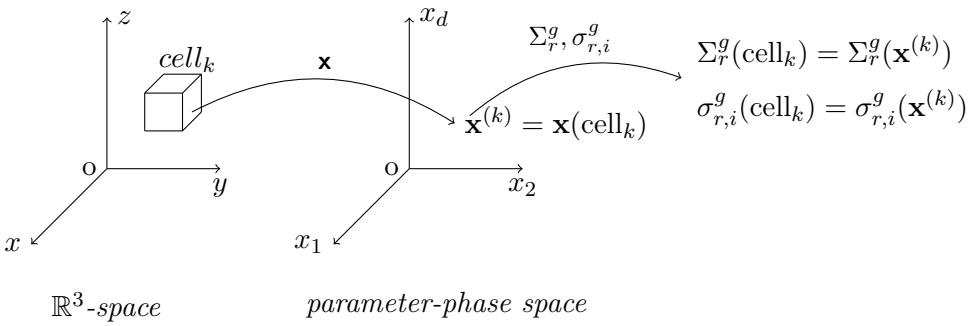


Figure 3.1 – Dependence of cross-sections on parameters.

of cross-section values needed for the simulation which scales in billions.

It may be time to indicate how, for a given value of $\mathbf{x} = (x_1, \ldots, x_d)$ each cross-section is computed. These are homogenized quantities representing average behaviors computed on a small homogenization cell, representing locally the core, with the full complexity of the physics (note that currently the full solution of such a model in a full three dimensional complexity of a core is far out of reach: this explains why a two stage approach has to be done).

In our applications at EDF-R&D, these cross-sections are extracted from the lattice code named APOLLO2 [Sanchez et al., 2010] that is developed at CEA. In this step, for a given point $\mathbf{x}$ on which cross-sections depend, *an APOLLO2 calculation is performed to provide all microscopic cross-sections in* $\{\sigma^g_{r,i}\}$ *(as the functions of* $\boldsymbol{x}$*) at this point.* The macroscopic cross-sections $\{\Sigma^g_r\}$ are then derived via the relation (3.1).

The main goal of APOLLO2 calculations is to compute the *neutron flux* ϕ. The calculation schemes used in APOLLO2 code are very complex because the resolution for many neutron equations are required. Such a calculation is expensive in time and for a given $\mathbf{x}$, it is referred to as *calculation point*. From this flux, all required cross-sections (see figure 3.2) can be computed very quickly .

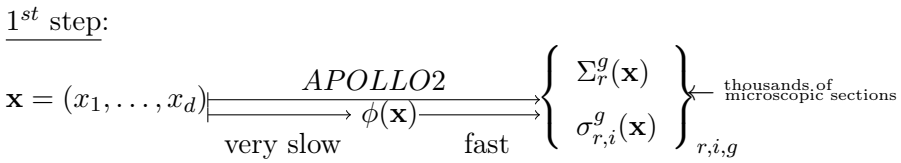


Figure 3.2 – Diagram of APOLLO2 calculations used in the first step.

The determination on the fly of all billion values of cross-sections by APOLLO2 is impossible because too complex and time consuming.

Currently, in most of the core codes, like in the COCAGNE code that is developed at EDF-R&D in our team, out of few calculations performed in a first stage (with APOLLO2) and stored in files, called *neutron libraries*, approximations of all the required cross-sections $\Sigma^g_r(\mathbf{x}(cell))$ for each cell

are obtained through multilinear interpolation. Note that a constraint in neutronics is that cross-sections need to be evaluated with a high accuracy (the absolute error is around "pcm" (per cent mille) where 1 pcm = 10^{-5}). After this is done in a first step of COCAGNE, the flux solver of the core code which computes the neutron flux $\phi(cell)$ can be performed as a second step. This whole stage is illustrated in figure 3.3.

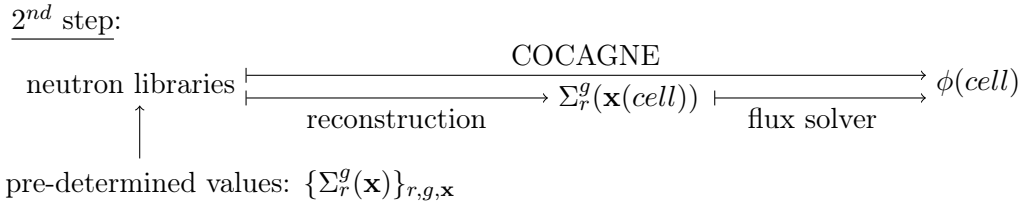


Figure 3.3 – Diagram of cross-sections reconstruction with the COCAGNE code in the second step.

Let us explain in more details the multilinear interpolation that is currently implemented in COCAGNE to reconstruct each cross-section. A high-dimensional tensorized grid for $\mathbf{x}$ is defined, where each axis (or phase direction) is a set of parameter values. Such a grid is referred to as *multilinear grid*. The values of each set $\{\Sigma_r^g(\mathbf{x})\}_{r,g}$ and $\{\sigma_{r,i}^g(\mathbf{x})\}_{r,i,g}$ are required for each node $\mathbf{x}$ of this grid. This is obtained in the first stage above mentioned out of the APOLLO2 code performed on every node. The values obtained are then stored in the neutron libraries. Then, a multilinear interpolation is used in the core code to evaluate cross-sections at any point inside this grid, that involves a simple linear combination of the cross-section values at the 2^d closest nodes.

Currently, cross-sections computed by APOLLO2 depend on $d = 5$ parameters. If the multilinear grid contains n points per axis, then we need to perform n^5 calculations with APOLLO2. With thousands of microscopic and around ten macroscopic cross-sections considered in the simulation, the pre-calculated process is heavy in computation time and in storage. This leads to difficulties for the following situations:

— 5 parameters are currently used in the regular situation but incident situations depend on more parameters.

— We would like to extend the calculation domain with parameter values $\mathbf{x}$ far away from the current ones.

The first situation increases the number of dimensions, whereas the last one increases the number of values per axis: we go from n^d to $m^{d'}$ with $m > n$ and $d' > d$. The time necessary to generate the neutron libraries as well as the size of data needs to be stored increase exponentially.

To deal with this problem, there exists already some works which try to improve the cross-section reconstruction model. For instance, global polynomial interpolation on sparse grids [Botes and Bokov, 2011], [Danniëll and Pavel, 2014] or spline [Hobson et al., 2013]. These have some limits due to lack of

regularity of the cross-section in some directions.

In this paper, we introduce a new, non-linear and ad'hoc model based on the Tucker format [Luu et al., 2015], [Hackbusch, 2012], viewed as a low-rank tensor approximation technique. Using this model, each cross-section is approached by a limited linear combination of tensor products of one-dimensional functions, referred to as *tensor directional basis functions*. The tensor directional basis functions are constructed by an extension of the Karhunen - Loève decomposition to high-dimensional spaces.

The tensor directional basis functions are not *a priori* chosen but tuned to each cross-section we consider. In particular, the number of the retained tensor directional basis functions (for a given accuracy) depend on the cross-section and also on the phase direction. Thanks to the Karhunen - Loève decomposition technique, the most important information of each cross-section is extracted, axis by axis, and represented into a few tensor directional basis functions.

The coefficients in the combination of the tensor products are determined by a system of linear equations traducing interpolation equalities at some points. Note that the points are the same, whatever the cross-section, this allows us to limit the number of APOLLO2 calculations performed on these points at the stage of determining the coefficients. In order to determine the points on which this system depends, we rely on the idea presented in the empirical interpolation [Maday et al., 2013], based on a *greedy algorithm*.

With these techniques, we can reconstruct the cross-sections with a high accuracy while reducing significantly the calculation points and the storage in the neutron library.

In the light to what was presented at the beginning of the introduction about the four concepts for approximation, our approach allows us to minimize the number of data acquisition that each involves a use of the code APOLLO2 hence also the storage of these data since this is an important part of the storage, the other part being the definition of the tensor directional basis functions. The interpolation process through the tensor shape of the approximation of each cross-section allows us to minimize the complexity of the reconstruction of the function we are interested in together with further evaluation in various different points.

The paper is organized in the following manner:

Section 2 describes the theoretical background of the Karhunen - Loève decomposition on which our approach based in order to construct the tensor directional basis functions.

In section 3, we present our proposed methodology for the Tucker decomposition in a general case.

Section 4 is reserved for practical applications to our problem: the reconstruction of cross-sections in neutronics. The implementation procedure as well as the cost of Tucker model will be detailed.

In section 5, we show the numerical results of a test case in order to compare the Tucker model

with the multilinear model using the following criteria: the number of calculation points, the storage in neutron libraries and the accuracy.

Section 6 is reserved for conclusion and discussion.

3.2 Theoretical background

3.2.1 Problem statement

From a mathematical point of view, our problem: the reconstruction of cross-sections, stands as the approximation of about one thousand multivariate functions $\{f_k(\mathbf{x})\}_k$, defined on the domain $\mathbf{\Omega}$. Here, $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbf{\Omega} \subset \mathbb{R}^d$, d is the number of parameters and $\mathbf{\Omega} = \Omega_1 \times \ldots \times \Omega_d$ $(\Omega_{i,1\leq i\leq d} \subset \mathbb{R})$.

The objective is to acquire information, store the data, and propose a reconstruction of each f_k with a high accuracy/complexity ratio. Remember that we expect an absolute accuracy to be of the order of 10^{-5} (or *pcm*).

Our approach is based on a low-rank tensor technique in order to represent each function f_k. Low-rank tensor representations (or formats) are widely used to treat the large-scale problems and there are many ways to express them, depending on the specific domain of application. We refer to [Hackbusch, 2012], [Nouy, 2016] and [Oseledets, 2011] for a general description of different formats. The Karhunen-Loève decomposition will be at the basis of our approach so we recall in the following subsection some elements of context.

3.2.1.1 Karhunen-Loève decomposition

The Karhunen - Loève decomposition that was introduced in statistics for continuous random processes [Karhunen, 1946], [Loève, 1955] is also known in various communities, as e.g. Proper Orthogonal Decomposition (POD) [Berkooz et al., 1993], Principal Component Analysis (PCA) [Pearson, 1901], [Hotelling, 1933], Empirical Orthogonal Functions (EOFs) [Lyons, 1982]. This decomposition is available for two-dimensional function spaces but has no direct extension to high-dimensional ones. The interest of this decomposition is that it provides an optimal rank-r approximation for two-variate functions, with respect to the L^2-norm.

Let $f = f(x, y)$ be a two-variate function in $L^2(\Omega_x \times \Omega_y)$. Through the Karhunen - Loève decomposition, f can be expressed as follows:

$$f(x, y) = \sum_{n=1}^{+\infty} \sqrt{\lambda_n}\varphi_n(x)\psi_n(y), \forall (x, y) \in \Omega_x \times \Omega_y \tag{3.2}$$

where $\{\varphi_n(x)\}_{n=1}^{+\infty}$ (resp. $\{\psi_n(y)\}_{n=1}^{+\infty}$) is a L^2-orthonormal basis of $L^2(\Omega_x)$ (resp. L^2-orthonormal

basis of $L^2(\Omega_y)$). Furthermore, each $(\lambda_n, \varphi_n(x))$ (resp. $(\lambda_n, \psi_n(y))$) is a couple of eigenvalue-eigenfunction of a Hilbert-Schmidt operator $K_f^{(x)}$ (resp. Hilbert-Schmidt operator $K_f^{(y)}$) with the following definitions:

$$\begin{array}{rcccl} K_f^{(x)} & : & L^2(\Omega_x) & \longrightarrow & L^2(\Omega_x) \\ & & u & \longmapsto & K_f^{(x)}u(x) = \int_{\Omega_x}\int_{\Omega_y} f(x,y)f(x',y)dyu(x')dx' \\ (resp.\ K_f^{(y)} & : & L^2(\Omega_y) & \longrightarrow & L^2(\Omega_y) \\ & & u & \longmapsto & K_f^{(y)}u(y) = \int_{\Omega_y}\int_{\Omega_x} f(x,y)f(x,y')dxu(y')dy' \quad) \end{array} \tag{3.3}$$

The $\{\lambda_n\}_{n=1}^{+\infty}$ are all positive and sorted in decreasing order: $\lambda_1 \geq \lambda_2 \geq \ldots > 0$.

For a given number $r \in \mathbb{N}^*$, if we search an approximation of f by r-rank tensor product :

$$f(x,y) \approx f_r(x,y) = \sum_{n=1}^{r} a_n u_n(x) v_n(y),\ \forall (x,y) \in \Omega_x \times \Omega_y \tag{3.4}$$

then the Karhunen-Loève decomposition provides the best approximation (see [Šimša, 1992]) in the sense of minimizing the root mean squared error (RMSE):

$$e^{RMSE} = \sqrt{||f(x,y) - f_r(x,y)||^2_{L^2(\Omega)}} = \sqrt{\int_{\Omega_x}\int_{\Omega_y} (f(x,y) - f_r(x,y))^2 dxdy} \tag{3.5}$$

Here, the best approximation f_r^{best} is the truncation by the first r terms of the Karhunen-Loève decomposition (3.2):

$$f_r^{best} = \sum_{n=1}^{r} \sqrt{\lambda_n}\varphi_n(x)\psi_n(y) \tag{3.6}$$

In this case, we have:

$$e^{RMSE} = \sqrt{||f(x,y) - f_r^{best}(x,y)||^2_{L^2(\Omega)}} = \sqrt{\sum_{n=r+1}^{\infty} \lambda_n} \tag{3.7}$$

The Karhunen-Loève decomposition for two-dimensional function spaces shows that the most important information to represent a two-variate function can be found in the first eigenfunctions (associated with the first largest eigenvalues). This property is exploited by the Higher-Order Singular Value Decomposition (HOSVD) technique [De Lathauwer et al., 2000], [Hackbusch, 2012], which is for objectives:

(i) Exploiting the best r-rank approximation for truncation.

(ii) Using $\{\varphi_n\}_n$ (or $\{\psi_n\}_n$) as tensor directional basis functions.

3.2.1.2 Implementation of the Karhunen-Loève decomposition

The tensor directional basis functions $\{\varphi_i(x)\}_i$ (resp. $\{\psi_i(y)\}_i$) of $L^2(\Omega_x)$ (resp. $L^2(\Omega_y)$) are thus eigenfunctions of the Hilbert-Schmidt operator $K_f^{(x)}$ (resp. $K_f^{(y)}$). One of the corresponding eigenproblem becomes an integral problem as given as follows:

$$\begin{aligned} &\text{Finding } (\lambda, \varphi) \in \mathbb{R} \times L^2(\Omega_x) \text{ such that:} \\ &K_f^{(x)}\varphi = \lambda\varphi \\ \Leftrightarrow \quad &\boxed{\int_{\Omega_x}\int_{\Omega_y} f(x,y)f(x',y)\varphi(x')dx'dy = \lambda\varphi(x),\ \forall x \in \Omega_x} \end{aligned} \tag{3.8}$$

The problem (3.8) is known as a Fredholm equation of the second type ([Atkinson, 1997]) which does not in general have an analytical solution. Hence, this problem is numerically solved by discretizing the domain $\Omega_x \times \Omega_y$ of f with a tensorized grid. The values of f on the discretized points (x_p, y_q) ($\{x_p\}_p$ being a suitable grid on Ω_x and $\{y_q\}_q$ a suitable grid on Ω_y), are denoted by $f_{pq} = f(x_p, y_q)$ and we set $\Omega_{N_x} = \{x_p | x_p \in \Omega_x,\ p = 1, 2, ..., N_x\}$, and $\Omega_{N_y} = \{y_q | y_q \in \Omega_y,\ q = 1, 2, ..., N_y\}$. The points x_p, y_q and (x_p, y_q) are *quadrature points*.

The discretization leads the discrete integral equation (3.8) to:

$$\sum_{p=0}^{N_x}\sum_{q=0}^{N_y} f(x_k, y_q)f(x_p, y_q)\Delta_{pq}\varphi(x_p) = \lambda\varphi(x_k),\ \forall x_k \in \Omega_{N_x} \tag{3.9}$$

where Δ_{pq} are *quadrature weights* at (x_p, y_q), depending on selected numerical integration method. In our work, $\Delta_{pq} = \delta_p\delta_q$ with δ_p (resp.δ_q) are one dimensional *quadrature weights* at x_p (resp. at y_q).

With the previous notions, the integral eigenproblem becomes a discrete matricial problem that reads

$$\textstyle\sum_{p=0}^{N_x}\sum_{q=0}^{N_y} f_{kq}f_{pq}\Delta_{pq}(\overrightarrow{\varphi})_p = \lambda(\overrightarrow{\varphi})_k,\ \ \overrightarrow{\varphi} = (\varphi(x_p))_{x_p \in \Omega_{N_x}} \in \mathbb{R}^{N_x} \tag{3.10}$$

Eigenfunctions $\varphi \in L^2(\Omega_x)$ in (3.8) are discretely represented by eigenvectors $\overrightarrow{\varphi} \in \mathbb{R}^{N_x}$ in (3.10).

If we define a matrix $A = (f_{pq}) \in M_{\mathbb{R}}(N_x, N_y)$ and $B = (\Delta_{pq}) \in M_{\mathbb{R}}(N_x, N_y)$, then the eigenvalue problem (3.10) will be written in a matrix formula as follows:

$$AB^TA^T\overrightarrow{\varphi} = \lambda\overrightarrow{\varphi} \tag{3.11}$$

Since the matrix $AB^TA^T \in M_{\mathbb{R}}(N_x, N_x)$, we obtain in general N_x eigenvalues for the problem (3.11) and N_x corresponding eigenvectors: $\overrightarrow{\varphi}_1, \ldots, \overrightarrow{\varphi}_i, \ldots, \overrightarrow{\varphi}_{N_x}$ where $size(\overrightarrow{\varphi}_i) = N_x$.

3.3 Methodology for the Tucker decomposition based on an extension of the Karhunen-Loève decomposition

3.3.1 Extension of the Karhunen-Loève decomposition into high-dimensional space for the construction of one-dimensional tensor directional basis functions

For a two-variate function, Karhunen-Loève decomposition leads, after truncation, to approximations of the behavior of the function in each variable with few ad'hoc one dimensional basis function. As we mentioned earlier, there is no direct extension of the Karhunen-Loève decomposition for high-dimensional spaces.

Following the formalism of the Tucker decomposition and of α-rank decompositions, we propose to consider a multidimensional function as a two-variate function of x and $\mathbf{y}$ and apply the Karhunen-Loève decomposition on this expression of the function: the first direction being one of the original variable present in the multidimensional setting, i.e. $x = x_j$ for $j = 1, \ldots, d$, the other being the remaining variables all condensed into one $\mathbf{y} := (x_1, \ldots, x_{j-1}, x_{j+1}, x_d)$. This approach, known as matricization, performed independently in each direction, allows us to propose appropriate tensor directional basis functions.

The Karhunen-Loève decomposition is iteratively used for each j and leads to d integral problems, each one is written as follows:

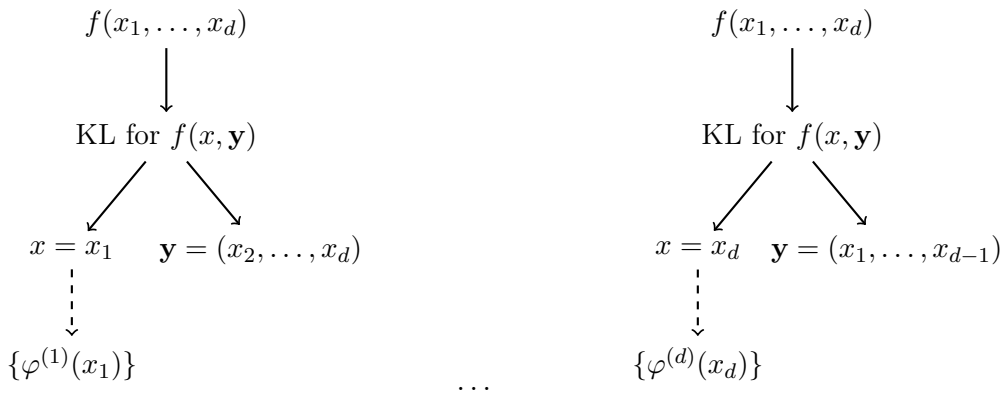
$$\int_{\Omega_x} \int_{\mathbf{\Omega_y}} f(x, \mathbf{y}) f(x', \mathbf{y}) \varphi(x') dx' d\mathbf{y} = \lambda \varphi(x), \; \forall x \in \Omega_x \tag{3.12}$$

Solving these integral problems allows us to determine tensor directional basis functions $\{\varphi^{(j)}(x_j)\}$ for each direction j $(1 \leq j \leq d)$. We only keep from this decomposition, the family of eigenvectors in the x_j variable. The extension process is illustrated in figure (3.4).

3.3.2 Numerical integration method used for integral equation

In order to solve numerically the integral equations like (3.8) (for two-dimensional spaces) or (3.12) (for high dimensional spaces), as explained in subsection 3.2.1.2 we need a method to approximate the integrals. Even if the approach is based on a two-variate presentation of the multidimensional function, the function in the variable $\mathbf{y}$ is in dimension $d-1$. In this context we want to propose a quadrature rule adapted to our quest, which is to capture the behavior in each variable while get some understanding of the global function.

In practice, for a d-variate function $f = f(x_1, \ldots, x_d)$, we shall need d different quadrature rules based on d grids, each one is used to construct tensor directional basis functions for only one



(a) Construction of tensor directional basis functions for the direction $x = x_1$.

(b) Construction of tensor directional basis functions for the direction $x = x_d$.

Figure 3.4 – Construction of tensor directional basis functions for all directions using an extension of the Karhunen-Loève decomposition (KL).

direction j. Therefore, we propose adaptive discretization in order to capture well information of the concerned direction but using as less discretized points as possible. Concretely, the concerned direction j is finely discretized while the others is coarsely (say by 2^{d-1} points). Such a discretization is referred to as a *Tucker grid*. Because we have a total of d grids in a d-dimensional space, so on each axis (or direction), we realize d different discretizations but only one is fine. These discretizations are shown in figure 3.5.

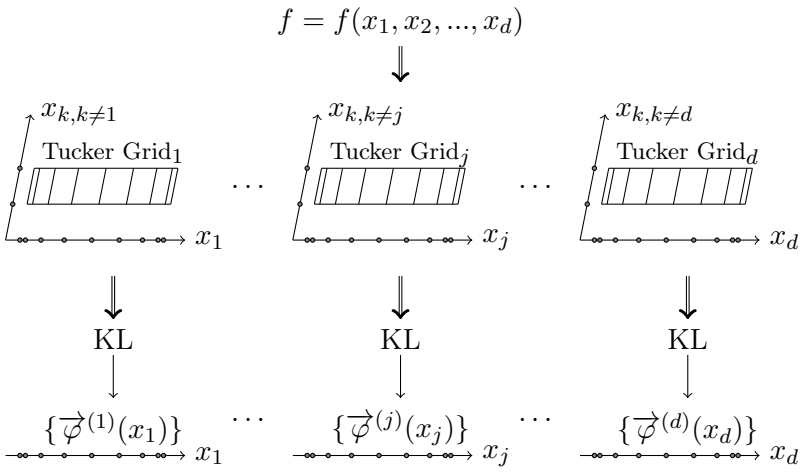


Figure 3.5 – Adaptive discretizations (*Tucker grids*) used for an extension of the Karhunen-Loève decomposition (*KL*) to construct tensor directional basis functions, direction per direction.

In our case, we chose the Clenshaw-Curtis quadrature associated with *Clenshaw-Curtis points* [Clenshaw and Curtis, 1960], [Boyd, 2001], [Trefethen, 2008] because the Clenshaw-Curtis points

are nested from N points to $2N$ points. If we discretize the interval $[-1, 1]$ into $N + 1$ Clenshaw-Curtis points then each one is defined by the following formula:

$$x_p = cos(\frac{p\pi}{N}),\ 0 \leq p \leq N \tag{3.13}$$

3.3.3 Determination of tensor directional basis functions in the Tucker decomposition

After solving numerically the integral equations like (3.12), we obtain eigenvectors $\{\overrightarrow{\varphi}^{(j)}(x_j)\}$ but not yet eigenfunctions $\{\varphi^{(j)}(x_j)\}$ for each direction j, $1, \leq j \leq d$. However, our goal is to propose an approximation of our multivariate function written as a low rank tensor, more precisely a tensor written as a Tucker decomposition:

$$f(x_1, \ldots, x_d) \approx \tilde{f}(x_1, \ldots, x_d) = \sum_{i_1=1}^{r_1} \ldots \sum_{i_d=1}^{r_d} \mathbf{a}_{i_1 \ldots i_d} \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_j) \tag{3.14}$$

that demands eigenfunctions on its right hand side. Then we need a method to reconstruct eigenfunctions from eigenvectors, direction by direction.

Such a reconstruction method applied to a direction j, should be coherent with the integration method used for this direction. In our current work, the integration method is based on a polynomial approach (with the quadrature rules). Hence, we choose the Lagrange interpolation for the reconstruction of eigenfunctions. It means that, for a direction j, the eigenfunctions $\varphi^{(j)}$ are represented as Lagrange polynomials via the quadrature points of this direction and via the eigenvector values at these points. The quadrature points mentioned here are the points in the fine discretization of the concerned direction, i.e., in our test, the Clenshaw-Curtis points (see figure 3.6).

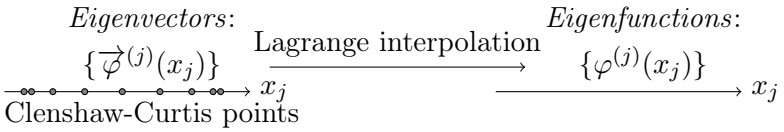


Figure 3.6 – Current method for the reconstruction of eigenfunctions from eigenvectors.

3.3.4 Criterion for the selection of tensor directional basis functions in the Tucker decomposition

We need to select from integral problems like (3.12) the most important eigenfunctions. This selection takes only eigenfunctions associated with the eigenvalues λ_i, where λ_i satisfies $\frac{\lambda_i}{\lambda_1} > \epsilon$, with $\lambda_1 \geq \lambda_2 \geq \ldots > 0$. Here, ϵ is a free parameter that can be chosen by the user. In our test, we take $\epsilon = 10^{-10}$.

3.3.5 Determination of coefficients in the Tucker decomposition

In the application of the Tucker decomposition (3.14) for $f = f(x_1, \ldots, x_d)$, we have to determine $R = \prod_{j=1}^{d} r_j$ coefficients $\mathbf{a}_i$, where r_j is the number of tensor directional basis functions in the direction j: $r_j = \#\{\varphi_{i_j}^{(j)}\}$. We can find the best $\mathbf{a}_i$ through projection, i.e. $\mathbf{a}_i = a_{i_1 \ldots i_d} =< f, \prod_{j=1}^{d} \varphi_{i_j}^{(j)} >$. But this method requires a high computational cost. A simpler approach that uses the explicit knowledge we have is based on interpolation at R well selected points $\{\mathbf{x}_t\}_{t=1}^{R}$. If these points are given with their coordinates $\mathbf{x}_t = (x_{t_1}, \ldots, x_{t_j}, \ldots, x_{t_d})$, then R coefficients $\{\mathbf{a}_1, \ldots, \mathbf{a}_i, \ldots, \mathbf{a}_R\}$ are the solution of the following system:

$$\begin{cases} f(\mathbf{x}_1) = \sum_{i=1}^{R} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{1_j}) \\ \ldots \\ f(\mathbf{x}_t) = \sum_{i=1}^{R} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{t_j}) \\ \ldots \\ f(\mathbf{x}_R) = \sum_{i=1}^{R} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{R_j}) \end{cases} \tag{3.15}$$

In our context, $\{\mathbf{x}_t\}_{t=1}^{R}$ will be referred to as *evaluated points of* f. The coordinates $\{x_{t_j}\}_{t=1}^{R}$ for a given direction j will be referred to as *evaluated points of tensor directional basis functions* $\{\varphi_{i_j}^{(j)}\}_{i_j=1}^{r_j}$ *in the direction* j.

3.4 Particular application for the reconstruction of cross-sections

3.4.1 Subdivision in the discretization of integral equations

In order to get an expertise on the functions we want to approximate, we have studied the variation of the cross-sections per parameter by varying the values of one parameter while the others are fixed at *nominal values* (the values on which the reactor operates normally). We found that the curve as a function of burnup varies a lot while the others are more close to linear. We show here an example in the case of the cross-section $\nu\Sigma_f^2$ (see figure (3.7)).

We thus have used far more points on the burnup axis than on any other axes (25 points for burnup while we have 5 for others). In order to capture more information of the burnup axis and avoid a high degree of polynomial approximations (used in the integration method), we subdivided this axis into three sub-intervals: $[0, 150]$, $[150, 10000]$ and $[10000, 80000]$ (see figure 3.8). Each subinterval has 9 Clenshaw-Curtis points (there are two common points: 150 and 10000 for 3 intervals).

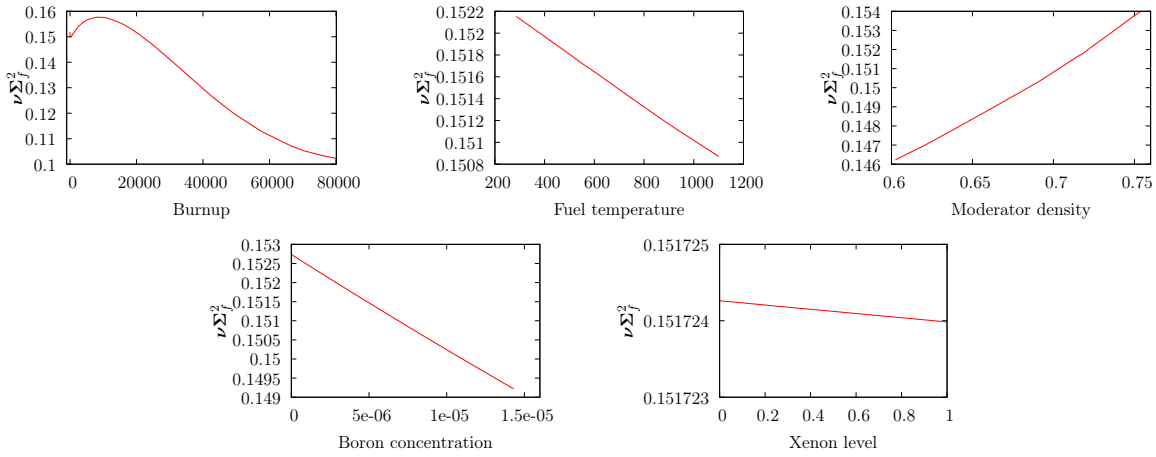


Figure 3.7 – Cross-section $\nu\Sigma_f^2$ as function of each parameter.

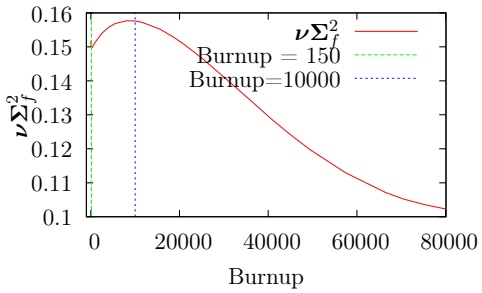


Figure 3.8 – Subdivision of the burnup axis.

3.4.2 Selection of the evaluated points by using recursively the greedy algorithm

3.4.2.1 Evaluated points on the left hand side of the system (3.15)

On the left hand side of (3.15), we need the values of each cross-section f at R evaluated points of $\{\mathbf{x}_t\}_{t=1}^{R}$. Already, the APOLLO2 calculations (performed in order to determine the cross-section values on Tucker grids) can be re-used for interpolating f at those points. As we know in section 5.1, APOLLO2 calculations are expensive in computation time so we want to reduce as much as possible these calculations at this stage.

In our case, the number of retained tensor directional basis functions in the Tucker decomposition (3.15) depends on the cross-section Σ_k with $1 \leq k \leq K$. Thus the selected points for interpolation depend on the cross-section.

Since an APOLLO2 calculation provides values of all cross-sections at a given point, we can determine a set of points, denoted by $\{\mathbf{x}_t(\bigcup_k \Sigma_k)\}_{t\geq 1}$, such that this set is usable for every cross-section: $\{\mathbf{x}_t(\Sigma_k)\}_{t=1}^{R(\Sigma_k)}$ can be extracted from $\{\mathbf{x}_t(\bigcup_k \Sigma_k)\}_{t\geq 1}$. Such a set $\{\mathbf{x}_t(\bigcup_k \Sigma_k)\}_{t\geq 1}$ is optimal for the number of APOLLO2 calculations if the cross-sections can use a maximal number of common

evaluated points, i.e. $\#(\cap_{\Sigma_k}\{\mathbf{x}_t(\Sigma_k)\}_{t=1}^{R(\Sigma_k)})$ should be maximal.

3.4.2.2 Proposed constitution of evaluated points

For each cross-section, we need a total of $R=\prod_{j=1}^{d} r_j$ evaluated points $\{\mathbf{x}_t\}_{t=1}^{R}$ on the left hand side of (3.15), where r_j is the number of tensor directional basis functions in the direction j. It is natural to use a set of tensorized points. Therefore, we can choose from each direction j $(1 \leq j \leq d)$ a set of only r_j evaluated points $\{x_{t_j}^{(j)}\}_{t_j=1}^{r_j}$ $(1 \leq j \leq d)$ used on the right hand side of (3.15) to constitute $\prod_{j=1}^{d} r_j$ points of $\{\mathbf{x}_t\}_{t=1}^{R}$:

$$\{\mathbf{x}_t\}_{t=1}^{R} = \times_{j=1}^{d}\{x_{t_j}^{(j)}\}_{t_j=1}^{r_j} \tag{3.16}$$

Using this construction, the set $\{x_{t_j}\}_{t=1}^{R}$ becomes $\{x_{t_j}^{(j)}\}_{t_j=1}^{r_j}$ with $r_j < R$.

3.4.2.3 Problem of the evaluated points choice

For sake of conveniency, because this allows us to use on the right hand side of the system (3.15) the values that were used to build the tensor directional basis functions $\{\varphi_{i_j}^{(j)}\}_{i_j=1}^{r_j}$, we propose to choose the evaluated points $\{x_{t_j}^{(j)}\}_{t_j=1}^{r_j}$ among the fine discretization $\{x_1^{(j)},\ldots,x_{N_j}^{(j)}\}$ of the direction j. This choice is nontrivial because $r_j < N_j$, leads to a total of $C_{N_j}^{r_j}$ choices for $\{x_{t_j}^{(j)}\}_{t_j=1}^{r_j}$ and $\prod_{j=1}^{d} C_{N_j}^{r_j}$ choices to constitute $\{\mathbf{x}_t\}_{t=1}^{R}$ by (3.16). To illustrate, our cross-sections depend on 5 parameters and $r_j = 4$, $N_j = 10$ $(1 \leq j \leq 5)$, we already have $C_{10}^{4} = 210$ choices for evaluated points of one direction j and 210^5 choices for evaluated points $\mathbf{x} = (x_1,\ldots,x_d)$.

When many cross-sections are studied at the same time, we have to determine a choice such that:

— The cross-sections use as many common evaluated points $\mathbf{x} = (x_1,\ldots,x_d)$ as possible to reduce the number of APOLLO2 calculations. This is equal to find as many as possible the common evaluated points $\{x_j\}$ for each direction j if we use the constitution proposed by (3.16).

— The accuracy of the Tucker decomposition for every cross-section must be ensured. (Because each choice leads to a change of coefficient values $\mathbf{a}$ on which the accuracy of the Tucker decomposition depends)

Remark: The choice of evaluated points $\{\mathbf{x}_t\}_{t=1}^{R}$ to solve the system (3.15) does not change the storage cost of the Tucker decomposition. This changes only the values of the coefficients $\mathbf{a}$ determined by (3.15) which are used in the Tucker decomposition (3.14). Since the accuracy of this decomposition is very sensitive to these coefficients, we must determine accurate coefficients by choosing the pertinent evaluated points $\{\mathbf{x}_t\}_{t=1}^{R}$ and $\{x_{t_j}^{(j)}\}_{t_j=1}^{r_j}$.

3.4.2.4 Proposed choice based recursively on the greedy method

We propose to use the empirical interpolation method, described in [Maday et al., 2007], to deal with our problem of constructing evaluated points $\{x_{t_j}^{(j)}\}_{t_j=1}^{r_j}$, $1 \leq j \leq d$, and $\{\mathbf{x}_t\}_{t=1}^{R}$. This method is based on a greedy algorithm to construct interpolation points, so we refer to it as *greedy algorithm*. Let us present it in a general framework: Let ω be a domain of $\mathbb{R}^n$: $\omega \subset \mathbb{R}^n$ ($1 \leq n \in \mathbb{N}$ in our applications, n will be 1 or d). $\mathcal{G}$ is a group of K functions defined on ω: $\mathcal{G} = \{\varphi_j : \omega \to \mathbb{R} | \, \varphi_j \in L^\infty(\omega), 1 \leq j \leq K\}$. The main idea of the greedy algorithm is to determine simultaneously P interpolation points $\mathcal{X} = \{x_1, \ldots, x_P\} \subset \omega$ and the associated interpolation operators $\mathcal{I}_P$, such that for any function $\varphi \in \mathcal{G}$, $\mathcal{I}_P(\varphi)$ is an interpolation of φ with the interpolation error satisfies: $||\varphi - \mathcal{I}_P(\varphi)||_{L^\infty(\omega)} \to 0$ rapidly as $P \to \infty$ ([Maday et al., 2007]).

The greedy algorithm can be described as follows:

— For $p = 1$

$$u_1 = argmax_{\varphi^{(j)} \in \mathcal{G}} ||\varphi^{(j)}||_{L^\infty(\omega)}$$

$$x_1 = argmax_{x \in \omega} |u_1(x)|$$

(i.e. $|u_1(x_1)| = \max_{x \in \omega, \varphi^{(j)} \in \mathcal{G}} |\varphi^{(j)}(x)|$)

— Set $\mathcal{X}_1 = \{x_1\}$

— Establish the interpolation operator $\mathcal{I}_1$:

$$\forall \varphi^{(j)} \in \mathcal{G} : \mathcal{I}_1[\varphi^{(j)}] = \frac{\varphi^{(j)}(x_1)}{u_1(x_1)} u_1(.)$$

— For $p > 1$

$$u_{p+1} = argmax_{\varphi^{(j)} \in \mathcal{G}} ||\varphi^{(j)} - \mathcal{I}_p[\varphi^{(j)}]||_{L^\infty(\omega)}$$

$$x_{p+1} = argmax_{x \in \omega} |u_{p+1}(x) - \mathcal{I}_p[u_{p+1}](x)|$$

(i.e. $|u_{p+1}(x_{p+1})| = \max_{x \in \omega, \varphi^{(j)} \in \mathcal{G}} |\varphi^{(j)}(x) - \mathcal{I}_p[\varphi^{(j)}(x)]|$)

— Set $\mathcal{X}_{p+1} = \mathcal{X}_p \bigcup \{x_{p+1}\}$

— Build the interpolation operator $\mathcal{I}_{p+1}$:

$$\forall \varphi \in \mathcal{G} : \mathcal{I}_{p+1}[\varphi] = \text{Lagrange interpolation via the } p+1 \text{ points of } \mathcal{X}_{p+1}$$

— Continue until the required value, i.e. $p = P$.

The greedy algorithm is well suited to the problem of finding the evaluated points $\{x_{t_j}^{(j)}\}_{t_j=1}^{r_j}$ in each direction j. Indeed, this algorithm allows us to determine a set of points for a group of functions, where the cardinal of these points is a given number and on which, each function in the group is well represented, well evaluated.

In our case, assume that all studied cross-sections are $\{\Sigma_k\}_{1\leq k\leq K}$ (Σ_k depends on d parameters). For a cross-section Σ_k and for a concerned direction j, we need to determine $r_j(\Sigma_k)$ evaluated points for the tensor directional basis functions $\{\varphi_{i_j}^{(j)}(\Sigma_k)\}_{i_j=1}^{r_j(\Sigma_k)}$ of this cross-section. Therefore, the *greedy algorithm applied to the cross-section Σ_k and the direction j for finding evaluated points* $\mathcal{X}(\Sigma_k) = \{x_{i_j}^{(j)}(\Sigma_k)\}_{i_j=1}^{r_j(\Sigma_k)}$ is performed with:

— $\omega = \{\text{all evaluated points } x_j \text{ of the direction } j\} = \{x_1^{(j)}, \ldots, x_{N_j}^{(j)}\} \subset \mathbb{R}$

— $P = r_j(\Sigma_k) < N_j^{(j)}$

— $\mathcal{G} = \{\varphi_{i_j}^{(j)}(\Sigma_k)\}_{i_j=1}^{r_j(\Sigma_k)}$

The evaluated points $\{\mathbf{x}_t(\Sigma_k)\}_{t=1}^{R(\Sigma_k)}$ are then constituted as a tensor product of evaluated points $\{x_{i_j}^{(j)}(\Sigma_k)\}_{i_j=1}^{r_j(\Sigma_k)}$ using (3.16).

Such a determination is well-suited for each cross-section but is not optimal for the number of APOLLO2 calculations. In fact, the APOLLO2 calculations are performed on every evaluated point $\mathbf{x}$ (used on the left hand side of (3.15)). Since we have a total of $\{\Sigma_k\}_{1\leq k\leq K}$ cross-sections, so the total evaluated points used by all cross-sections is $\bigcup_{\Sigma_k,1\leq k\leq K}\{\mathbf{x}_t(\Sigma_k)\}_{t=1}^{R(\Sigma_k)}$, with $\{\mathbf{x}_t(\Sigma_k)\}_{t=1}^{R(\Sigma_k)} \neq \{\mathbf{x}_t(\Sigma_l)\}_{t=1}^{R(\Sigma_l)}$ if $k \neq l$, in general. This leads to a significant number of APOLLO2 calculations for $\bigcup_{\Sigma_k,1\leq k\leq K}\{\mathbf{x}_t(\Sigma_k)\}_{t=1}^{R(\Sigma_k)}$. As we have mentioned in section 3.4.2.3, the solution is finding as many as possible the common evaluated points between the sets $\{x_{i_j}^{(j)}(\Sigma_k)\}_{i_j=1}^{r_j(\Sigma_k)}$ for each direction j.

We denote $\{x_j(\bigcup_k \Sigma_k)\}$ by the set of common evaluated points for the direction j. The cardinal of this set must be as small as possible to have the maximal common evaluated points between the cross-sections. Since $\{x_j(\bigcup_k \Sigma_k)\}$ are used for tensor directional basis functions in $\bigcup_{\Sigma_k,1\leq k\leq K}\{\varphi_{i_j}^{(j)}(\Sigma_k)\}_{i_j=1}^{r_j(\Sigma_k)}$, we propose to take a set $\{x_j(\bigcup_k \Sigma_k)\}$ such that $\#(\{x_j(\bigcup_k \Sigma_k)\}) = P$ where P is equal to the highest number $r_j(\Sigma_k)$, i.e. :

$$P := \#\{x_j(\bigcup_k \Sigma_k)\} = r_j^{max} = \max_{\Sigma_k,1\leq k\leq K} r_j(\Sigma_k)$$

This means that all points of $\{x_j(\bigcup_k \Sigma_k)\}$ are also the evaluated points of any cross-section. If we denote this cross-section by Σ_k^{max}, then we have $r_j^{max} = r_j(\Sigma_k^{max})$. Therefore, we have $\{x_j(\bigcup_k \Sigma_k)\} \equiv \{x_{i_j}^{(j)}(\Sigma_k^{max})\}_{i_j=1}^{r_j(\Sigma_k^{max})}$. After determining $\{x_j(\bigcup_k \Sigma_k)\}$, all cross-sections Σ_k can select their evaluated points $\{x_{i_j}^{(j)}(\Sigma_k)\}_{i_j=1}^{r_j(\Sigma_k)}$ with $\#\{x_{i_j}^{(j)}(\Sigma_k)\}_{i_j=1}^{r_j(\Sigma_k)} = r_j(\Sigma_k)$ from r_j^{max} points of $\{x_j(\bigcup_k \Sigma_k)\}$. *This selection can use again the greedy algorithm.*

In our case, we propose to use a discrete version of the greedy algorithm where $\{\varphi_{i_j}^{(j)}(\Sigma_k)\}_{i_j=1}^{r_j(\Sigma_k)}$ are replaced by $\{\overrightarrow{\varphi}_{i_j}^{(j)}(\Sigma_k)\}_{i_j=1}^{r_j(\Sigma_k)}$.

Assuming that all tensor directional basis functions $\{\varphi_{i_j}^{(j)}(\Sigma_k)\}_{i_j=1}^{r_j(\Sigma_k)}$ are determined in the direction j, *our greedy version applied to a direction j for finding* $\mathcal{X}(\bigcup_k \Sigma_k) = \{x_j(\bigcup_k \Sigma_k)\}$ is described as follows:

— Collect all tensor directional basis functions of the direction j into a group $\mathcal{G}_j$:

$$\mathcal{G}_j = \bigcup_{\Sigma_k,\, 1\leq k\leq K} \{\varphi_{i_j}^{(j)}(\Sigma_k)\}_{i_j=1}^{r_j(\Sigma_k)}$$

— Find the max number of evaluated points $\{x_{i_j}^{(j)}(\Sigma_k)\}_{i_j=1}^{r_j(\Sigma_k)}$ over all cross-sections:

$$r_j^{max} = \max_{\Sigma_k} r_j(\Sigma_k) \tag{3.17}$$

— Apply the greedy algorithm (see above) to:

$\mathcal{G} \equiv \mathcal{G}_j$, $P \equiv r_j^{max}$ and $\Omega \equiv \{N_j$ points in the fine discretization for $j\}$

With this strategy, the number of APOLLO2 calculations used for evaluated points $\mathbf{x}$ (on the right hand side of (3.15)) of all studied cross-sections ($\{\Sigma_k\}_{1\leq k\leq K}$) are determined by:

$$\prod_{j=1}^{d} r_j^{max}, \text{ where } r_j^{max} \text{ is defined in (3.17)} \tag{3.18}$$

After this stage, performed offline, for each direction j, we have all evaluated points $\mathcal{X}_{r_j^{max}} = \{x_j(\bigcup_k \Sigma_k)\}$ on which each cross-section $\Sigma_{k,k\geq 1}$ can select its proper evaluated points for finding $\mathcal{X}(\Sigma_k) = \{x_{i_j}^{(j)}(\Sigma_k)\}_{i_j=1}^{r_j(\Sigma_k)}$ again from the greedy algorithm:

— $\mathcal{G} \equiv \{\varphi_1^{(j)}(\Sigma_k), \ldots, \varphi_{r_j}^{(j)}(\Sigma_k)\}$

— $P \equiv r_j(\Sigma_k)$

— $\Omega \equiv \mathcal{X}_{r_j^{max}} = \{x_j(\bigcup_k \Sigma_k)\}$

3.4.3 Summary of the implementation procedure

We have introduced in the previous sections our proper Tucker decomposition and the procedures to determine the components of this decomposition. Let us now summarize the principal steps when applied to a multivariate function $f = f(x_1, \ldots, x_d)$, where $x_j \in \Omega_j \subset \mathbb{R}$, $j = 1, \ldots, d$:

(i) *Find the one-dimensional tensor directional basis functions for all directions j, $1 \leq j \leq d$:*

— Domain discretization leads to d Tucker grids:

— Fine discretization of Ω_j into N_j points

— Coarse discretization for the other directions with $N_{k(j)}$ points, $k \neq j$

— Using the extension of the Karhunen-Loève decomposition to construct tensor directional basis functions, direction per direction:

— Establish the integral problems in high dimensional space like equation (3.12)

— Solve numerically the integral problems by solving the eigenvalue problems similar to the equation (3.10)

— Use the criterion described in section 3.3.4 to select the dominant eigenvectors $\varphi_{i_j}^{(j)}$, leads to $i_j = 1, \ldots, r_j$ in the concerned direction j

— Construct the tensor directional basis functions for the direction j by interpolating the selected eigenvectors in a polynomial basis

(ii) *Determination of the coefficients* $\boldsymbol{a}$ *in the Tucker decomposition*:

$$f(x_1,\ldots,x_d) \approx \tilde{f} = \sum_{i_1=1}^{r_1} \ldots \sum_{i_d=1}^{r_d} \mathbf{a}_{i_1\ldots i_d} \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_j) \tag{3.19}$$

— For a direction j: choose r_j points among r_j^{max} evaluated points of $\mathcal{X}_{r_j^{max}}$ with the greedy algorithm (section 3.4.2.4)

— Constitute $r_1 \ldots r_d = \prod_{j=1}^{d} r_j$ evaluated points $\mathbf{x} = (x_1,\ldots,x_d)$ by a tensor product of $\{x_{i_j}^{(j)}\}_{i_j=1}^{r_j}$ (by (3.16))

— Find the coefficients $\mathbf{a}$ by solving the linear system (3.15)

(iii) *Evaluation of* f *at any point* $\boldsymbol{x} = (x_1,\ldots,x_d)$*:*

— Evaluate all $\varphi_{i_j}^{(j)}$ at x_j with $i = 1,\ldots,r_j$, $j = 1,\ldots,d$

— Evaluate $f(\mathbf{x})$ using Tucker decomposition (3.19)

3.4.4 Cost of the multilinear interpolation and the Tucker decomposition

3.4.4.1 Multilinear interpolation process and the number of APOLLO2 calculations, the storage size of cross-sections

The multilinear interpolation process can be illustrated by figure 3.9.

$$\{\text{node}_{\mathbf{x}}\} \in \begin{matrix}\text{multilinear}\\ \text{grid}\end{matrix} \xrightarrow[\text{APOLLO2}]{} \{\Sigma_r^g(node_{\mathbf{x}})\} \xrightarrow[\text{Store}]{} \text{Neutron library}$$

Figure 3.9 – Multilinear interpolation process.

We see that the APOLLO2 calculations are performed on all nodes $\mathbf{x}$ of the multilinear grid and then the cross-section values on these nodes are directly stored in the neutron library.

If the multilinear grid has N_j points on the axis j, $(1 \leq j \leq d)$, then the number of APOLLO2 calculations is equal to:

$$\prod_{j=1}^{d} N_j \tag{3.20}$$

and the *storage is the same* for all cross-sections:

$$\sum_{j=1}^{d} N_j + \prod_{j=1}^{d} N_j \tag{3.21}$$

3.4.4.2 Tucker decomposition process and the number of APOLLO2 calculations, the storage of cross-sections

Tucker decomposition process

The Tucker decomposition process can be illustrated by figure 3.10.

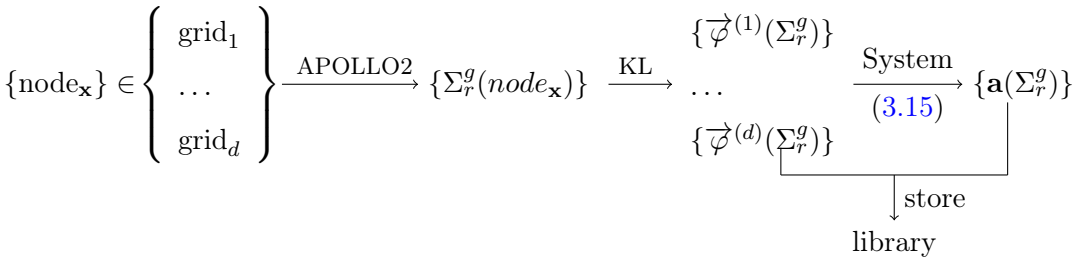


Figure 3.10 – Tucker decomposition process (here KL: Karhunen-Loève decomposition).

We see that the APOLLO2 calculations are performed on the nodes $\mathbf{x}$ of the Tucker grids, but we do not store the cross-section values on these nodes like in the case of the multilinear model. Instead of that, we store the values of the eigenvectors $\{\overrightarrow{\varphi}^{(j)}(\Sigma_r^g)\}_{j=1}^d$ and the coefficients $\mathbf{a}$.

Number of APOLLO2 calculations in the Tucker decomposition

In the Tucker model, the total number of APOLLO2 calculations used *for all cross-sections* $\{\Sigma_r^g\}_{r,g}$ is calculated as follows:

$$\sum_{j=1}^{d}(N_j \prod_{k(j)=1,k(j)\neq j}^{d} N_{k(j)}) + \prod_{j=1}^{d} r_j^{max} \tag{3.22}$$

Where:

- N_j: the number of points in the fine discretization of the direction j.
- $N_j \prod_{k(j)=1,k(j)\neq j}^{d} N_{k(j)}$: number of nodes in the j^{th} Tucker grid on which APOLLO2 calculations are performed (ref. figure 3.5).
- r_j^{max}: defined by (3.17)
- $\prod_{j=1}^{d} r_j^{max}$: the number of evaluated points $\{\mathbf{x}_t\}_{t=1}^R$ used by all cross-sections. Here, the APOLLO2 calculations are used on these points to compute cross-section values on the left hand side of each system like (3.15), to get the coefficients $\mathbf{a}$.

In our test, we chose $N_{k(j)} = 2$ so the first part of (3.22) equals to $\sum_{j=1}^{d} N_j 2^{d-1}$.

Storage in the Tucker decomposition

The storage size *for one cross-section* $\Sigma_{k,k\geq 1}$ is the number of the floating points which needs to

be stored. It is *varied by cross-section* and described as follows:

$$\sum_{j=1}^{d} N_j + \sum_{j=1}^{d} r_j(\Sigma_r^g) * N_j + \prod_{j=1}^{d} r_j(\Sigma_r^g) \tag{3.23}$$

Where:

— N_j: the number of points in the fine discretization of the direction j.

— $r_j(\Sigma_r^g) * N_j$: the values of $r_j(\Sigma_r^g)$ eigenvectors $\overrightarrow{\varphi}^{(j)}$ for the direction j. Here, N_j is also the size of $\overrightarrow{\varphi}^{(j)}$.

— $\prod_{j=1}^{d} r_j(\Sigma_r^g)$: the number of coefficients **a** for the cross-section Σ_r^g.

3.5 Numerical results

3.5.1 Description of the test case

We present here a numerical test case using the Tucker decomposition and the multilinear model to reconstruct the cross-sections. Some cross-sections are considered in our work like: the `macro totale` - Σ_t^g, the `macro absorption` - Σ_a^g, the `macro fission` - Σ_f^g and the `macro nu*fission` - $\nu\Sigma_f^g$. A particular case, the `macro scattering` depends on 3 indexes g, g' and o, denoted by $\Sigma_{so}^{g \to g'}$. In our test case, $o \in \{0, 1\}$, $g \in \{1, 2\}$, $g' \in \{1, 2\}$ and the cross-sections depend on 5 parameters: *burnup, fuel temperature, moderator density, boron concentration* and *xenon level*. These parameters vary in the intervals given in table 3.1, around their nominal values.

Parameter name	min value	max value
Burnup, MWd/t	0.0	80000.0
Fuel temperature,$^{\circ}C$	286.0	1100.0
Moderator density, g/cm^3	0.602	0.753
Boron concentration, ppm	0.0	1800.0
Xenon level, %	0.0	1.0

Table 3.1 – Parameters and their intervals.

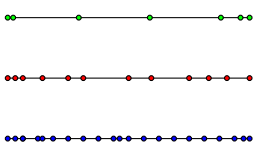
3.5.2 Discretization of the parameters space

In order to compare the Tucker decomposition with the multilinear interpolation method already implemented in the COCAGNE code, we need a reference sample of points included in a grid named *reference grid*. In this grid, we tried to select the points such that they are different from the multilinear grid and from the Tucker grids. The reason is that if a point belongs to a multilinear

grid then cross-section values are exact at this point (no interpolation) with the multilinear model, whereas for the Tucker model, it is not.

In our test case, the *multilinear grid* has a large number of points in the burnup direction (33 points) while the others have 2 or 3. The *Tucker grids* contain 5 grids corresponding to 5 directions. Each one has 5 Clenshaw-Curtis points in the studied direction and 2 points in all other directions except for the burnup-grid with 25 points in the burnup direction (see the description in section 3.4.1).

These discretizations are illustrated in a one-dimensional space in figure 3.11 and are described in table 3.2.



Tucker grids $\longrightarrow$ Tucker decomposition

multilinear grid $\longrightarrow$ multilinear interpolation

reference grid $\longrightarrow$ to calculate evaluated error of each method

Figure 3.11 – Illustration of the different grids used in the test case.

Direction	*reference grid*	*multilinear grid*	*Tucker grids*
burnup	36	33	25 ($25 * 2^4 = 400$ points)
Fuel temperature	4	3	5 ($5 * 2^4 = 80$ points)
Moderator density	4	3	5 ($5 * 2^4 = 80$ points)
Boron concentration	4	3	5 ($5 * 2^4 = 80$ points)
Xenon level	3	2	5 ($5 * 2^4 = 80$ points)
Total	6912 points $= 36 * 4^3 * 3$	1782 points $= 33 * 3^3 * 2$	720 points $= 25 * 2^4 + 4 * (5 * 2^4)$

Table 3.2 – The discretization of the *reference grid*, of the *multilinear grid* and of the *Tucker grids*.

3.5.3 Approximation errors for cross-sections

We compare the accuracy of the multilinear interpolation and the Tucker decomposition on each node $\mathbf{x}_i$ of the *reference grid*. The approximation error on the *reference grid* is defined either by the infinity norm (inf) of relative errors or by the root mean square of absolute errors ($RMSE$) :

$$e^{(inf)} = \max_{\mathbf{x}_i \in reference\ grid} | \underbrace{\frac{\tilde{f}(\mathbf{x}_i) - f(\mathbf{x}_i)}{\max|\{f(\mathbf{x}_i)\}|} * 10^5}_{\text{relative error}} | \tag{3.24}$$

or

$$e^{(RMSE)} = \sqrt{\sum_{i=1}^{N} \frac{((f(\mathbf{x}_i) - \tilde{f}(\mathbf{x}_i)) * 10^5)^2}{N}}, \text{ with } N = \#(reference\ grid) \tag{3.25}$$

where $f(\mathbf{x}_i)$ and $\tilde{f}(\mathbf{x}_i)$ are respectively exact values (calculated by APOLLO2) and approximation values (evaluated by either method) of a cross-section, performed at $\mathbf{x}_i \in$ *reference grid*. The 10^5 factor is here to have units in *pcm*.

We denote e_{Tucker} and $e_{multilinear}$ respectively the error by the Tucker approximation and by the multilinear interpolation.

3.5.4 Approximation errors for reactivity

Cross-sections are merely inputs for the flux solver. We are more interested in k_{eff} or reactivity for instance, which are outputs of the flux solver. In some simplified cases, we do not need to go through solving a neutron equation, the following analytic formula is applied (see page 1172, 1173, 1221 of the book [Marguet, 2013]):

$$\text{reactivity} = 1 - \frac{1}{k_{eff}}, \text{with } k_{eff} = k_\infty = \frac{\nu\Sigma_f^1 * (\Sigma_t^2 - \Sigma_{s0}^{2\to2}) + \nu\Sigma_f^2 * \Sigma_{s0}^{1\to2}}{(\Sigma_t^1 - \Sigma_{s0}^{1\to1}) * (\Sigma_t^2 - \Sigma_{s0}^{2\to2}) - \Sigma_{s0}^{1\to2} * \Sigma_{s0}^{2\to1}}$$

In order to measure the reactivity error, we use $\mathbf{e}^{(inf)}_{\text{reactivity}}$ with the following definition:

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

3.5.5 Results

In this section, results will be shown for the two cases: the Tucker decomposition using and not using the greedy algorithm. We will compare these results with the multilinear model using the following criteria: the number of calculation points, the storage and the accuracy.

We present in table 3.3, the results issued from the Karhunen-Loève decomposition for some macroscopic cross-sections. This allows us to compute the number of APOLLO2 calculations and the storage in the Tucker decomposition (see section 3.4.4.2).

3.5.5.1 Comparison of the number of calculation points

The number of calculation points for the Tucker model (see (3.22)) includes the calculations on all nodes of Tucker grids (in this case, we have 5 grids for 5 parameters with 720 nodes in total (table 3.2)) and the calculations performed on the evaluated points $\mathbf{x}$ on the left hand side of (3.15) (in this case, we have 378 evaluated points (see table 3.3)). The number of calculation points for the multilinear model (see (3.20)) is equal to the number of nodes in the multilinear grid. Hence, we obtain the results presented in table 3.4. These results show that, by using the Tucker decomposition, the number of calculation points has been reduced by 38% compared with multilinear model.

Cross section	Number of tensor directional basis functions r_j for direction j					Number of coefficients a ($\prod_{j=1}^{d} r_j$)
	r_{burnup}	$r_{Fuel\,temperature}$	$r_{Moderator\,density}$	r_{Boron}	r_{Xenon}	
Σ_t^1	4	2	2	2	2	64
Σ_t^2	3	2	3	2	2	72
Σ_a^1	5	3	2	3	2	180
Σ_a^2	5	2	3	3	2	180
$\nu\Sigma_f^1$	7	2	2	3	2	168
$\nu\Sigma_f^2$	5	2	3	2	2	120
$\Sigma_{s0}^{1\to1}$	4	2	2	2	2	64
$\Sigma_{s0}^{1\to2}$	4	3	3	3	2	216
$\Sigma_{s0}^{2\to1}$	6	2	3	3	2	216
$\Sigma_{s0}^{2\to2}$	3	2	3	2	2	72
Σ_f^1	7	2	2	2	2	112
Σ_f^2	5	2	3	2	2	120
$\mathbf{r_j^{max}}$	**7**	***3***	***3***	***3***	***2***	
$\Longrightarrow$ **Number of evaluated points** $\{\mathbf{x}_t\}_{t=1}^{R}$ (used on the left hand side of the systems (3.15), determined by (3.19)): 7*3*3*3*2 = **378**						

Table 3.3 – Number of: tensor directional basis functions, coefficients **a** and evaluated points **x** in the Tucker decomposition for some cross-sections.

Number of calculation points	
Tucker decomposition	Multilinear Interpolation
$\underbrace{720}_{\textit{for 5 Tucker grids}} + \underbrace{378}_{\text{for evaluated points } \mathbf{x}} = \mathbf{1098}$	**1782**

Table 3.4 – Comparison of the number of calculation points between the Tucker decomposition and the multilinear interpolation.

3.5.5.2 Comparison of the storage

The storage of the Tucker decomposition for one cross-section includes the axial discretization values, the values of selected eigenvectors and the coefficients **a** (see (3.23)). *The storage of the multilinear model* includes the axial discretization values, the cross-section values on the nodes of the multilinear grid (3.21). The number of axial discretization values (25+4*5 = 45 for the Tucker model and 33+3+3+3+2 = 44 for the multilinear model) is the same for all cross-sections and becomes small when all cross-sections are considered. Therefore, these numbers can be neglected in the comparison of the storage. We obtained the results presented in table 3.5 for some cross-sections. These results show that, by using the Tucker decomposition, the storage size has been reduced by a factor from 4 to 9 (depending on the cross-section) compared to the multilinear model.

Cross section	**Storage (number of floats)**	
	Tucker	Multilinear
Σ_t^1	244	1782
Σ_t^2	192	1782
Σ_a^1	355	1782
Σ_a^2	355	1782
$\nu\Sigma_f^1$	388	1782
$\nu\Sigma_f^2$	290	1782
$\Sigma_{s0}^{1\to1}$	244	1782
$\Sigma_{s0}^{1\to2}$	371	1782
$\Sigma_{s0}^{2\to1}$	416	1782
$\Sigma_{s0}^{2\to2}$	192	1782
Σ_f^1	327	1782
Σ_f^2	290	1782

Table 3.5 – Comparison of the storages between the Tucker decomposition and the multilinear interpolation.

3.5.5.3 Comparison of accuracy

When the tensor directional basis functions are determined, the accuracy of the Tucker decomposition depends only on values of the coefficients $\mathbf{a}$. We recall that the storage size (e.g. in table 3.5) does not depend on the coefficient values. We present here the results of the Tucker decomposition with and without using the greedy algorithm. We recall that without the greedy algorithm, we select randomly the evaluated points. The accuracies of the Tucker decomposition are compared to the multilinear interpolation over 6912 points of the reference grid.

The comparisons are presented in table 3.6. In this table, the Tucker decomposition without using the greedy algorithm has already a better accuracy for the e^{RMSE} errors for all reconstructed cross-sections (see in column 6), compared to the multilinear interpolation (see in column 5). But this is worse than the multilinear one for some cross-sections ($\nu\Sigma_f^1$, Σ_f^1) with e^{inf} errors (presented by framed errors) and for the reactivity with e^{RMSE}. With the greedy algorithm, all errors are now better than those of the multilinear interpolation.

Figure 3.12 presents the relative errors for the cross-section $\nu\Sigma_f^1$. In this case, the accuracy of the Tucker decomposition using the greedy algorithm is the best, compared to the multilinear model and the Tucker decomposition without using the greedy algorithm.

Figure 3.13 presents the relative errors for the cross-section $\Sigma_{s0}^{2\to1}$. In this case, the accuracy of the Tucker decomposition is also better than that of the multilinear interpolation. The distribution of errors in the case using the greedy algorithm seems to be the same as the case without using the

greedy algorithm.

Figure 3.14 presents the approximation errors of the reactivity. The accuracy of the Tucker decomposition with the greedy algorithm is the same order of multilinear interpolation accuracy.

Cross Section	$\mathbf{e}^{(\mathbf{inf})}_{\mathbf{multilinear}}$	$\mathbf{e}^{(\mathbf{inf})}_{\mathbf{Tucker}}$ (without greedy algorithm)	$\mathbf{e}^{(\mathbf{inf})}_{\mathbf{Tucker}}$ (with greedy algorithm)	$\mathbf{e}^{(\mathbf{RMSE})}_{\mathbf{multilinear}}$	$\mathbf{e}^{(\mathbf{RMSE})}_{\mathbf{Tucker}}$ (without greedy algorithm)	$\mathbf{e}^{(\mathbf{RMSE})}_{\mathbf{Tucker}}$ (with greedy algorithm)
Σ_t^1	47.24	28.15	26.58	6.449	3.182	3.126
Σ_t^2	118.11	40.63	34.67	77.191	14.50	10.341
Σ_a^1	195.76	25.86	28.10	0.805	0.089	0.101
Σ_a^2	417.01	49.64	49.52	8.972	1.999	2.097
$\nu\Sigma_f^1$	116.65	130.66	42.58	0.277	0.132	0.080
$\nu\Sigma_f^2$	289.75	72.80	70.72	11.601	3.949	4.146
$\Sigma_{s0}^{1\to1}$	40.86	25.42	23.88	5.322	2.795	2.724
$\Sigma_{s0}^{1\to2}$	194.94	24.07	19.28	0.663	0.663	0.096
$\Sigma_{s0}^{2\to1}$	794.39	97.17	96.73	0.642	0.124	0.119
$\Sigma_{s0}^{2\to2}$	126.08	10.61	18.92	74.947	3.824	5.023
Σ_f^1	122.62	137.54	58.79	0.108	0.052	0.033
Σ_f^2	297.06	68.16	65.81	4.560	1.454	1.525
Reactivity	**467.40**	**342.82**	**334.04**	**95.17**	**97.68**	**80.04**

Table 3.6 – Comparison of the accuracy on 6912 points of the reference grid for the Tucker decomposition and for the multilinear interpolation.

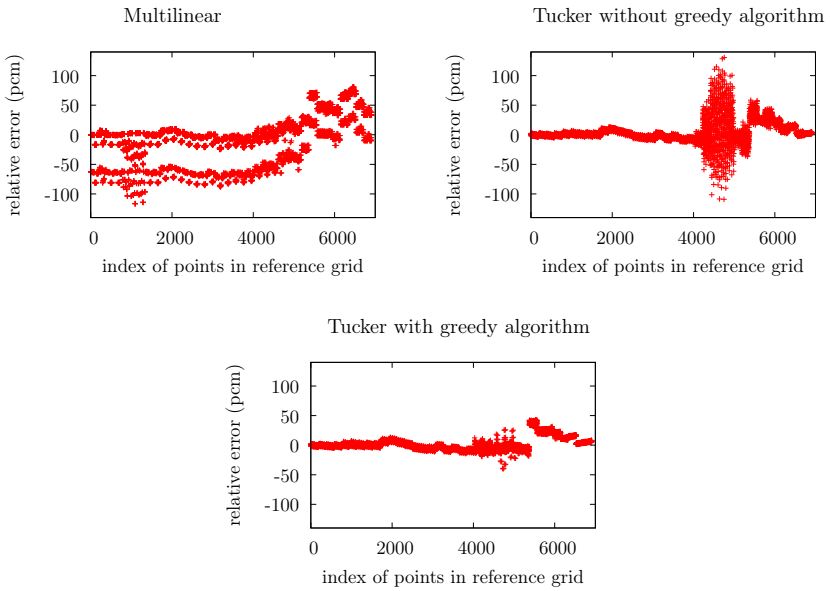


Figure 3.12 – Comparison of relative errors (pcm) for the cross-section $\nu\Sigma_f^1$.

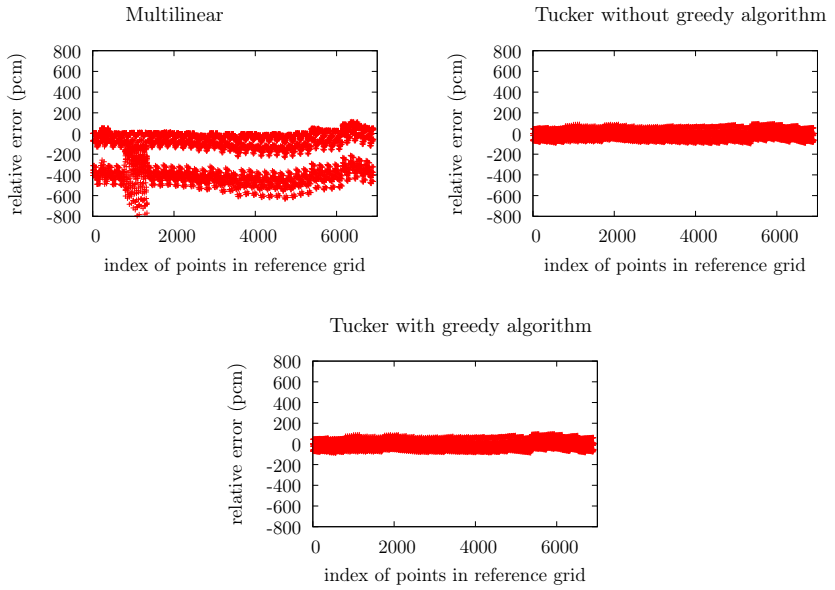


Figure 3.13 – Comparison of relative errors (pcm) for the cross-section $\Sigma_{s0}^{2\to1}$.

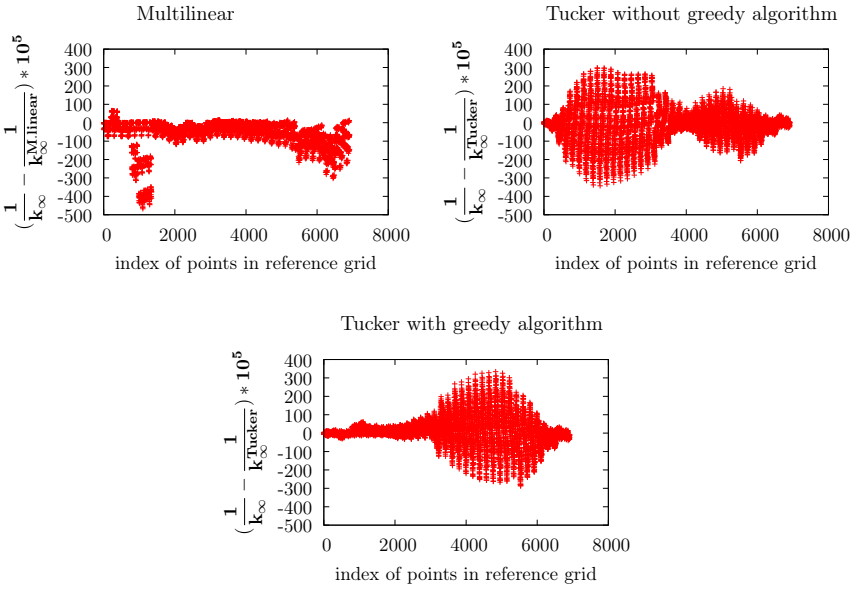


Figure 3.14 – Comparison of approximation errors (in pcm) for the reactivity.

These results show that the Tucker model is more accurate than the multilinear model in general while reducing significantly the computational cost (number of APOLLO2 calculations and storage in neutron libraries).

3.6 Conclusion and discussion

We have presented the Tucker decomposition applied to the reconstruction of neutron cross-sections. We showed that this method has better accuracy for cross-sections while using a smaller number of calculation points and a smaller storage size than those of the multilinear interpolation. Since cross-sections are merely inputs for the flux solver, the results of the reactivity are especially interesting. In the simplified hypothesis, we showed that the reactivity with the Tucker model is the same order of multilinear interpolation accuracy.

However, the Tucker decomposition has some limitations compared to the multilinear model. It may be slower than the multilinear interpolation in the reconstruction step. At this stage, the Tucker model uses the Lagrange interpolation to evaluate the tensor directional basis function (expensive in time) while the multilinear model uses the linear interpolation. Moreover, the multilinear interpolation ensures the positivity of cross-sections contrary to the Tucker model. The multilinear interpolation also keeps the linearity relation, e.g. $\Sigma_t^1 = \Sigma_a^1 + \Sigma_{s0}^{1\to 2} + \Sigma_{s0}^{1\to 1}$ is correct at every point, but this is not true for the Tucker decomposition.

Our future work will focus on the resolution of the Tucker limitations. Furthermore, we plan to study a criteria which allows us to eliminate the less important coefficients $\mathbf{a}$ in the representation of the Tucker decomposition. This should lead to a similar accuracy while reducing storage in the neutron libraries. We will also apply the Tucker model to more complex cases, e.g. extension of the calculation domain. The accuracy will be verified for real values of k_{eff} which are performed by the flux solver instead of analytic formula which is only valid for simplified cases.

Chapter 4

Benchmarking of Tucker decomposition method for reconstruction of neutron macroscopic cross-sections

This is a submitted paper with Y.Maday, M.Guillo and P.Guérin. Its reference in the manuscript is [Luu et al., 2016a].

Abstract

The neutron cross-sections are inputs for nuclear reactor core simulations, they depend on various physical parameters. Because of industrial constraint (e.g. calculation time), the cross-sections can not be calculated on the fly due to the huge number of them. Hence, approximate reconstruction (or interpolation) of cross-sections for a given set of parameters is used to evaluate on the fly the cross-sections at every point it is required, from (as few as possible) pre-calculated points. With most classical methods (for example: multilinear interpolation which is used in the core code COCAGNE of EDF (Électricité De France)), high accuracy for the reconstruction often requires a lot of pre-calculated points. We propose to use the Tucker decomposition, a low-rank tensor approximation method, to deal with this problem. The Tucker decomposition allows us to capture the most important information (one parameter at a time) to reconstruct the cross-sections. This information is stored as basis functions (called *tensor directional basis functions*) and the *coefficients* of the decomposition instead of pre-calculated points. Full reconstruction is done at the core code level using these decompositions. In this paper, a simplified multivariate analysis technique (based on statistical analysis) is also proposed in order to demonstrate that we can improve the quality of

the acquired information as well as the accuracy of our approach. Using the Tucker decomposition, we will show in proposed benchmarks that we can reduce significantly the number of pre-calculated points and the storage size (compared to the multilinear interpolations) while achieving high accuracy for the reconstruction, even on a larger domain of parameters.

Keywords: cross-sections reconstruction, Tucker decomposition, interpolation, low-rank tensor approximation, statistical analysis

4.1 Introduction

Like most companies working in nuclear electricity production, Électricité De France in its research and development (EDF R&D) departments develops highly accurate nuclear reactor core simulator system. Two main classes of approach are employed for simulations : deterministic and probabilistic. Our work relates to the deterministic one.

The purpose of a core simulator system is to be able to simulate any kind of physical quantity for the proper operation and the safety of the power plant. In order to do that, one has to solve Boltzmann's equations (or an approximation of these) for neutrons. These equations need, at every (physical-)cell of the 3D space, some physical inputs, the cross-sections, denoted by the letter Σ, that model the interactions between fission-induced neutrons and nuclei from either the fuel or the moderator (water in the case of PWR).

These cross-sections vary from one (physical-)point of the core to another - hence $\Sigma = \Sigma(\overrightarrow{P})$ with $\overrightarrow{P} = (x, y, z)$ - and depend on d local parameters (so called feedback parameters), such as: *burnup (bu), fuel temperature (t_f), moderator density (ρ_m), boron concentration (b_c), xenon level (xe), etc.* Each feedback parameter can be seen as one axis of a d-dimensional space called *parameter-phase space* and a set of parameter values as a point in this parameter-phase space. Therefore, for each cell in a 3D space, there is a d-tuple parameter's value in the d-dimensional parameter-phase space. The actual value of d is model dependent.

In order to do that, core simulator systems classically involve two solvers used in chain : a lattice code and a core code. At the EDF in the department SINETICS (SImulation NEutronique, Technologie de l'Information, Calcul Scientifique), we use, in a first step, the lattice code APOLLO2 [Sanchez et al., 1988], [Sanchez et al., 2010] developed at CEA, that generates cross-sections in the parameter-phase space and store them inside a file. This file acts like a database; we call it a " *nuclear library*". Then, in a second step, the core code COCAGNE [Plagne and Ponçot, 2005] reads this library and, for every cell in the 3D space, computes the values of the d parameters (using for instance thermal-hydraulic code, depletion loop and so on) and evaluates the values of the cross-sections using the database and a reconstruction model. The reconstruction allows from values in

the nuclear library (the smaller in size the better) to get an approximate value at any point in the parameter-phase space (the more accurate the better).

For the simulation of nuclear reactor core, one generally defines a particular point in the parameter-phase space at which the reactor core operates normally. This point is called *nominal point/condition*. Actually, the *burnup* direction is not involved in the definition of the nominal point since it is related to time. The domain near the nominal point (which thus lives in $\mathbb{R}^{d-1}$) is referred to as the *standard domain*. It is composed of values of the parameters in the parameter-phase space that are encountered under standard working conditions of the power plant. On the contrary, it is called the *extended domain* when the parameters get out of these running situations and are encountered in some special operations for the reactor or incidental situations. There is no much precise definition of these domains that may represent different objects through various publications.

In general, high accuracy for the reconstruction requires a lot of pre-calculated points, i.e., a lot of lattice calculations. The total number of pre-calculations performed by the lattice code is usually so large ($\sim$ thousands) that it takes a lot of calculation time (while a calculation takes only few seconds). This becomes more cumbersome for an extended domain with a model such as multilinear interpolation. The reduction of the number of these calculations as much as possible has been the motivation for introducing in [Luu et al., 2016b] a new reconstruction based on a low-rank tensor approximation method, that is referred to as *the Tucker decomposition*. The purpose of the current paper is to present and test on various assemblies (UOX, UOX-Gd, MOX, on standard and extended domains) this new reconstruction and compare it to the currently used multi-linear interpolator in COCAGNE.

The paper is organized in the following manner:

Section 2 gives an overview of different methods for the reconstruction of cross-sections used by different utility companies. In section 3, we present the Tucker decomposition applied to a set of multivariate functions. Section 4 illustrates how we can efficiently use the Tucker decomposition for the cross-section reconstruction problem and how we can deduce cross-section properties in high dimension. In this section, we will show the advantages of our model, which allow us to pre-analyze and post-analyze our approach. Section 5 is reserved to our proposed benchmarks applied to different fuel assemblies (UOX, UOX-Gd, MOX). The results obtained demonstrate that we can reduce the pre-calculated data while achieving high accuracy in both cases: the standard domain and the extended domain. We also present some problems that we encountered in this section. Section 6 is dedicated to conclusion and perspective.

4.2 Overview of different methods for the reconstruction of neutron cross-sections

There are many core simulator systems for nuclear reactor simulations, for example: the pair of softwares ARCADIA(HERMES)-ARTEMIS [Hobson et al., 2013] used by AREVA, DRAGON-DONJON [Polytechnique Montréal, 2016] used by Canadian Nuclear Society, or NEXUS-ANC [Müller et al., 2007] used by Westinghouse, etc. Each such system employs a model in order to reconstruct the cross-sections. These models can be classified in two main categories:

— The cross-section values are approximated by adding some perturbation or correction terms to values calculated at or around the nominal point, see e.g. [Fujita et al., 2014], [Turski et al., 1997], [Müller et al., 2007], [Stålek and Demazière, 2008]. In general, the correction terms are based on physical knowledge or some expansion techniques, such as the Taylor expansion.

— The cross-section values are interpolated from the pre-calculated values on a grid (for instance, Cartesian grid constructed with a tensor product of the discretized points on each axis [Watson and Ivanov, 2002] , sparse grid [Danniëll and Pavel, 2014], quasi-random grid [Dufek, 2011], etc.). The interpolation techniques in these models are different by the choice of basis functions: piece-wise linear functions, B-spline, Lagrange polynomials, etc. COCAGNE belongs to this category where the Cartesian grid and the piece-wise linear functions are used in the multilinear interpolation model.

The first category's methods have been proven suitable for the simulation on a "standard" domain where the parameter-values are rather close to the nominal condition. Of course, when they are far away from this condition, their accuracy is lost since the heuristics used around nominal values may not be valid anymore. The second ones are more general but high accuracy requires a lot of pre-calculated data. For instance, the multilinear approach that is used at EDF, relies on the acquisition of the values of the various cross sections on a Cartesian grid of the (standard or extended) domain in the parameter-phase space, i.e. in $\mathbb{R}^5$ from the lattice code. This nuclear library has a cardinal equal to N^d, where N stands for the number of discretized points in each phase direction. Then the reconstruction allows us to build an approximation of each cross section at any point by: i) locating this point to one of the cells of the Cartesian grid (to which this point belongs), here each cell is seen as a d dimensional object and its vertices are the points on which the value of each cross section is available, ii) averaging the previous values in a convex way to provide a linear approximation in each dimension. This is a very simple methods, and its accuracy (second oder in L^2 or L^∞ norms scaling like $\mathcal{O}(N^{-2})$) in terms of size of the cell that implies to have a quite large library on standard domains and very - even too much - large nuclear library on extended domains.

Therefore, either we have a high efficiency reconstruction method (meaning high accuracy with

few points) but on a specific domain only, or we have an expensive reconstruction method (meaning high accuracy at a lot of pre-calculated points) but suitable for any domain.

We present in the next section a new approach in this field [Luu et al., 2016b], based on the Tucker decomposition [Tucker, 1966], [Hackbusch, 2012], [Luu et al., 2015].

4.3 Reconstruction model based on the Tucker decomposition

In this work, the term "parameter" can be replaced by another ones, such as: *variable*, *axis* or *direction*.

Let $f(\mathbf{x}) = f(x_1, \ldots, x_d)$ be a multivariate function defined on $\mathbf{\Omega} = \Omega_1 \times \ldots \times \Omega_d \subset \mathbb{R}^d$. The Tucker decomposition relies on the tensor product of one-variate functions, that takes the following form for the function f:

$$f(\mathbf{x}) \approx \tilde{f}(x_1, \ldots, x_d) = \sum_{i_1=1}^{r_1} \ldots \sum_{i_d=1}^{r_d} \underbrace{\mathbf{a}_{i_1 \ldots i_d}}_{\text{coefficient}} \prod_{j=1}^{d} \underbrace{\varphi_{i_j}^{(j)}(x_j)}_{\text{tensor directional basis function}} \tag{4.1}$$

In this decomposition, we need to determine:

(i) The r_j tensor directional basis functions $\{\varphi_{i_j}^{(j)}\}_{i_j=1}^{r_j}$ for each direction j, $1 \leq j \leq d$.

(ii) The total of $R = \prod_{j=1}^{d} r_j$ coefficients $\mathbf{a}_i = \mathbf{a}_{i_1 \ldots i_d}$

First, the tensor directional basis functions are constructed by an extension of the Karhunen-Loève decomposition, known as Higher-Order Singular Value Decomposition (HOSVD) technique (see [De Lathauwer et al., 2000] and section 8.3, page 230 of the book [Hackbusch, 2012]). In general, the number r_j of tensor directional basis functions used in each direction is small.

Next, the coefficients are obtained by solving a system of linear equations. This system requires inputs as the values of the function f on a set of points selected by using a greedy algorithm (see [Maday et al., 2007], [Maday et al., 2013]). We refer to our previous article [Luu et al., 2016b] for more details on these techniques. We present briefly the methodology in next section.

4.3.1 Determination of tensor directional basis functions

In our work, we do not *a priori* choose tensor directional basis functions. The tensor directional basis functions are constructed, direction by direction, by extracting important information (principal component) from a sample of points. This sample is selected per direction such that it contains rich information for the studied direction. The technique used to extract information is based on the Karhunen-Loève decomposition and the HOSVD technique.

Concretely, the tensor directional basis functions for each direction j $(1 \leq j \leq d)$ are chosen among the eigenfunctions $\{\varphi^{(j)}\}$ of a corresponding Hilbert-Schmidt operator $K_f^{(j)}$, where:

$$\begin{aligned} & K_f^{(j)}\varphi^{(j)}(x_j) = \int_{\Omega_j}\int_{\Omega_{\mathbf{y}}} f(x_j,\mathbf{y})f(x'_j,\mathbf{y})\varphi^{(j)}(x'_j)dx'_j d\mathbf{y},\ \forall x_j \in \Omega_j \\ \text{with} \quad & \mathbf{y} \in \Omega_{\mathbf{y}} := \Omega_1 \times \ldots \times \Omega_{j-1} \times \Omega_{j+1} \times \ldots \times \Omega_d \end{aligned} \tag{4.2}$$

Therefore, $\{\varphi^{(j)}\}$ are solutions of the following eigenproblem:

$$\boxed{\int_{\Omega_j}\int_{\Omega_{\mathbf{y}}} f(x_j,\mathbf{y})f(x'_j,\mathbf{y})\varphi^{(j)}(x'_j)dx'_j d\mathbf{y} = \lambda\varphi^{(j)}(x_j),\ \forall x_j \in \Omega_j} \tag{4.3}$$

The problem (4.3) is known as a Fredholm equation of the second type [Atkinson, 1997] which does not in general have an analytical solution. Hence, this problem is numerically solved by discretizing the domain $\Omega_j \times \Omega_{\mathbf{y}}$ of f into $\Omega_{N_j} \times \Omega_{N_{\mathbf{y}}}$, where $\#\Omega_{N_j} = N_j$, $\#\Omega_{N_{\mathbf{y}}} = N_{\mathbf{y}}$. The values of f on the discretized points $(x_p, \mathbf{y}_q)$ are denoted by:

$$f_{pq} = f(x_p, \mathbf{y}_q),\ \text{with}\ (x_p, \mathbf{y}_q) \in \Omega_{N_j} \times \Omega_{N_{\mathbf{y}}} \tag{4.4}$$

If the integral weight at $(x_p, \mathbf{y}_q)$ is denoted by Δ_{pq} and the discrete values of x_j (in (4.3)) are chosen in Ω_{N_j}, the problem (4.3) can be written as the following discrete equations:

$$\sum_{p=0}^{N_j}\sum_{q=0}^{N_y} f_{kq}f_{pq}\Delta_{pq}(\overrightarrow{\varphi}^{(j)})_p = \lambda(\overrightarrow{\varphi}^{(j)})_k,\ \overrightarrow{\varphi}^{(j)} = (\varphi^{(j)}(x_p))_{x_p \in \Omega_{N_j}} \in \mathbb{R}^{N_j} \tag{4.5}$$

On the left hand side of the equations (4.5), values f_{pq} of f at discretized points (cross-section values performed by lattice code in our application) are required. In order to reduce the number of required values f_{pq} while capturing information efficiently for each direction, we propose to realize the following discretization:

— Fine discretization for the direction j with N_j points.

— Coarse discretization for the other directions k $(k \neq j)$ with 2 points.

Such a discretization corresponds to a grid, referred to as *Tucker grid* or *parameter-name grid* if we want to mention which parameter is concerned in this grid.

In order to get high accuracy approximation with the quadrature rules for the integral equations (4.3), we chose the Clenshaw-Curtis points [Clenshaw and Curtis, 1960], [Boyd, 2001] for the fine discretizations. If we discretize the interval $[-1, 1]$ into $(N + 1)$ Clenshaw-Curtis points then each one is defined by the following formula:

$$x_p = cos(\frac{p\pi}{N}),\ 0 \leq p \leq N \tag{4.6}$$

We see that the Clenshaw-Curtis points are the projection of $(N + 1)$ equidistant points of the semi circle $C(0, 1)$ on $[-1, 1]$. Therefore, these points are closer to the endpoints than the middle of the interval.

By the theory of Hilbert-Schmidt operators, the eigenvalues of the equation (4.3) are all positive. Moreover, they decrease quickly in general. Another property of this operator is that the eigenfunctions corresponding to the largest eigenvalues can be used to represent the principal information of f. Hence, the tensor directional basis functions for the direction j of the Tucker decomposition are selected from the first dominant eigenfunctions of the equation (4.3). The discretized values of these eigenfunctions are obtained by solving the linear system (4.5).

In practice, we keep only the eigenfunctions associated with the eigenvalues λ_i satisfying: $\dfrac{\lambda_i}{\lambda_1} > \epsilon$, with $\lambda_1 \geq \lambda_2 \geq \ldots > 0$. Here, ϵ is a parameter that can be chosen by the user.

The procedure to construct tensor directional basis functions per direction results in having as many Tucker grids as the number of parameters. This procedure is illustrated by figure 4.1.

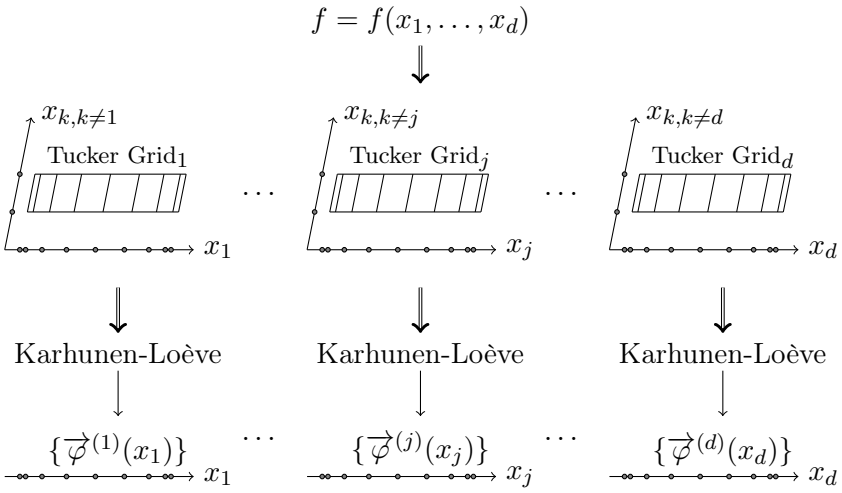


Figure 4.1 – Construction of the tensor directional basis functions using the Tucker grids and the Karhunen-Loève decomposition, direction by direction.

As previously mentioned, the eigenvectors are discretized on Clenshaw-Curtis points. In order to reconstruct tensor directional basis function from these values, we choose the Lagrange interpolation (see illustration in figure 4.2).

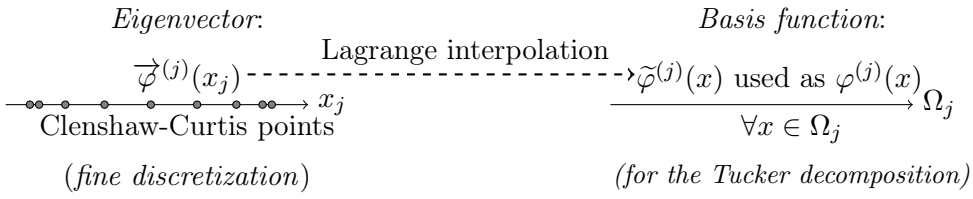


Figure 4.2 – Lagrange interpolation in order to reconstruct the tensor directional basis functions from the eigenvectors for a direction j.

4.3.2 Determination of coefficients

In the Tucker decomposition (4.1), we need to determine $R = \prod_{j=1}^{d} r_j$ coefficients $\mathbf{a}_i$, where r_j is the number of the tensor directional basis functions in the direction j. We propose to choose these coefficients as the solution of the following system:

$$\begin{cases} f(\mathbf{x}_1) = \sum_{i=1}^{R} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{1_j}) \\ \dots \\ f(\mathbf{x}_t) = \sum_{i=1}^{R} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{t_j}) \\ \dots \\ f(\mathbf{x}_R) = \sum_{i=1}^{R} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{R_j}) \end{cases} \tag{4.7}$$

where $\mathbf{x}_t = (x_{t_1}, \dots, x_{t_j}, \dots, x_{t_d}) \in \Omega \subset \mathbb{R}^d$.

It means that the approximation by the Tucker decomposition is exact at R points of the set $\{\mathbf{x}_t\}_{t=1}^{R}$ (the Tucker decomposition becomes an interpolation of f on these R points).

At this stage, new values of function f (values of cross-sections performed by the lattice code APOLLO2 in our application) are again required on the left hand side of the system (4.7). Indeed, the points $\mathbf{x}_1, \dots, \mathbf{x}_R$ do not belong to any of the previous Tucker grids because they are constructed as a tensor product of points extracted from the fine discretizations of the Tucker grids (see later).

The choice of the set $\{\mathbf{x}_t\}_{t=1}^{R}$ is not trivial because we can take any point $\mathbf{x} \in \Omega$ to constitute such a set. But the accuracy of the Tucker decomposition depends on the coefficients, i.e., also depends on the choice of the set $\{\mathbf{x}_t\}_{t=1}^{R}$. To deal with this problem, we propose to constitute the set $\{\mathbf{x}_t\}_{t=1}^{R}$ as a tensor product of 1D-sets $\{x_{t_j}^{(j)}\}_{t_j=1}^{r_j}$:

$$\{\mathbf{x}_t\}_{t=1}^{R} = \times_{j=1}^{d} \{x_{t_j}^{(j)}\}_{t_j=1}^{r_j} \tag{4.8}$$

where $\{x_{t_j}^{(j)}\}_{t_j=1}^{r_j}$ are selected among the N_j ($N_j > r_j$) points of the fine Clenshaw-Curtis discretization of the direction j. For each axis, a greedy algorithm is applied to the set of tensor directional basis functions $\{\varphi_{i_j}^{(j)}\}_{i_j=1}^{r_j}$ and the corresponding fine discretization in the direction j in order to find the r_j points of the set $\{x_{t_j}^{(j)}\}_{t_j=1}^{r_j}$. We let the reader refer to [Maday et al., 2007] for detailed explanation of the greedy algorithm.

If we have to reconstruct not only one function f but a set of functions $\{f_k\}_k$ (as in our application to the reconstruction of cross-sections), the set of points $\{\mathbf{x}_t\}_{t=1}^{R}$ need to be constituted such that it is suitable for the use of any function f_k. In this case, the greedy algorithm is recursively employed for the determination of the set $\{x_{t_j}^{(j)}\}_{t_j=1}^{r_j}$ on each axis (see [Luu et al., 2016b] for more details).

4.4 Application of the Tucker decomposition for the reconstruction of cross-sections

4.4.1 Cross-section notions

In neutron physics, the cross-sections are denoted by Σ_r^g where r designates the reaction kind and g designates the energy group. For our applications to the COCAGNE software, we refer to some cross-sections like the `macro totale` - Σ_t^g, the `macro absorption` - Σ_a^g, `macro fission` - Σ_f^g and the `macro nu*fission` - $\nu\Sigma_f^g$. A particular case, the `macro scattering` also depends on anisotropy order o, departure energy group g and arrival energy group $g^{'}$, denoted by $\Sigma_{so}^{g\rightarrow g^{'}}$.

In our work, there are two energy groups considered: fast group ($g = 1$) and thermal group ($g = 2$). The cross-sections depend on 5 parameters: *burnup, boron concentration, moderator density, fuel temperature* and *xenon level*. Therefore, we have 5 Tucker grids corresponding to 5 parameters.

4.4.2 Reference grid

In order to measure the accuracy of our reconstruction method, we use a Cartesian grid, referred to as *reference grid*. This grid is very fine (with around 10,000 points for a standard domain and with around 14,000 points for an extended domain). The discretized values on each axis of the reference grid are chosen such as they are different from the fine discretization of the Tucker grids and relatively randomized. The cross-section values performed by the APOLLO2 code [Sanchez et al., 1988], [Sanchez et al., 2010] on the reference grid are *exact values* (Σ_r^g).

The values of the cross-sections evaluated by the Tucker decomposition on the reference grid are *approximated values* ($\widetilde{\Sigma}_r^g$).

From the exact and approximated values on the reference grid, we can evaluate the error of our approach and test the quality of the reconstruction. We illustrate the use of the reference grid in the diagram presented in figure 4.3.

$$\mathbf{x} = (x_1, \ldots, x_d) \in \text{Reference Grid} \xrightarrow{\text{APOLLO2}} \textit{exact value: } \Sigma_r^g(\boldsymbol{x}) \Longrightarrow \text{evaluated error}$$
$$\mathbf{x} = (x_1, \ldots, x_d) \in \text{Reference Grid} \xrightarrow{\text{Tucker}} \textit{approximated value: } \widetilde{\Sigma}_r^g(\boldsymbol{x}) \Longrightarrow \text{evaluated error}$$

Figure 4.3 – Reference grid in order to measure the accuracy of our approach.

As it is often the case when dealing with cross-sections, we will use the *pcm* unit ($1\ pcm = 10^{-5}$) to evaluate errors. As we saw in figure 4.3, each point $\mathbf{x}$ of the reference grid can be associated with

an evaluated error. In our work and for cross-sections, this evaluated error is a relative error which is defined by:

$$e_{\Sigma_r^g,\,\text{Relative}}(\mathbf{x}) = \underbrace{\frac{\Sigma_r^g(\mathbf{x}) - \tilde{\Sigma}_r^g(\mathbf{x})}{\max_{\mathbf{x}} |\Sigma_r^g(\mathbf{x})|} * 10^5}_{\text{Relative Error (pcm)}}, \quad \mathbf{x} \in \textit{reference grid}$$

The "max" is employed in the denominator of the relative error in order to correctly deal with small cross-sections.

In order to have an estimate for the accuracy of our approach, a final error is defined as follows:

$$e_{\Sigma_r^g}(pcm) = \max_{\mathbf{x} \in \textit{reference grid}} |e_{\Sigma_r^g,\,\text{Relative}}(\mathbf{x})| \tag{4.9}$$

4.4.3 Analyses used in order to achieve high accuracy for the reconstruction problem

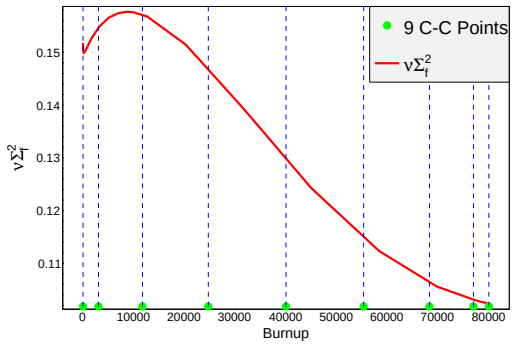
As we saw, the Tucker decomposition depends on the chosen tensor directional basis functions and these bases are constructed axis by axis from the eigenvectors of the Karhunen-Loève decomposition. Therefore, information represented in the tensor directional basis functions as well as the number of eigenvectors taken for each axis impact directly the accuracy of the Tucker method. These two characters are exploited in our analyses (pre-analysis and post-analysis) in order to propose adaptive solutions to achieve the [desired accuracy]/[storage] ratio for the reconstruction problem.

4.4.3.1 Pre-analysis of cross-sections as functions of each parameter

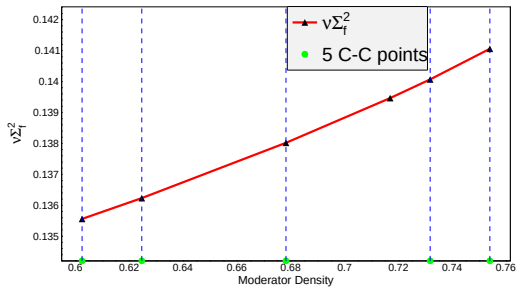
In our work, as we explained in the previous section, the approximation of the functions is done by polynomials, this approximation is global and of high order, this is true at least if the regularity is coherent over the whole domain Ω_j. Therefore, if the cross-sections vary much in some zones, the approximation must be adjusted in order to capture the variation, either by increasing the degree of polynomials or by splitting up the domain Ω_j in sub-domains (like in the spectral element method). At some points however, increasing the degree may lead to instabilities, hence we limit the order of the Lagrange polynomials in our approach to 8, thus using a maximum of 9 Clenshaw-Curtis points.

In order to analyze cross-section variation and adjust our approach, the idea is thus to analyze each cross-section as a function of one parameter at a time, which allows us to see *a priori* the variation of cross-sections.

In practice, we study the variation of cross-sections for each parameter by varying only this parameter while the others are fixed at nominal values. We show in figure 4.4 an example with the cross-section $\nu\Sigma_f^2$, where the UOX assembly is studied on the standard domain.



(a) $\nu\Sigma_f^2$ as a function of burnup. The discretization with 9 Clenshaw-Curtis (C-C) points (represented by small circles) is not enough at low *burnup* values.



(b) $\nu\Sigma_f^2$ as a function of moderator density. Clenshaw-Curtis (C-C) points (represented by small circles) are suitable in this case. (*The triangle symbols represent the values of* $\nu\Sigma_f^2$ *at the Clenshaw-Curtis points, in which, the nominal point (*~ 0.72*) is automatically added by the simulator system*).

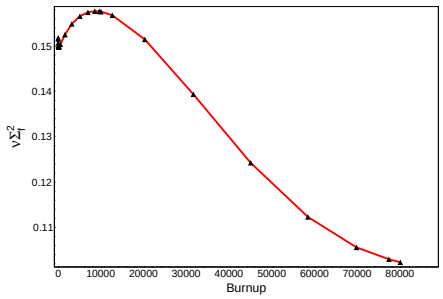
Figure 4.4 – Variation of the cross-section $\nu\Sigma_f^2$ as a function of one parameter (case of the UOX assembly on the standard domain).

In the figure 4.4, we see that the curve of $\nu\Sigma_f^2$ as a function of *burnup* varies a lot (see figure 4.4a) while for the other parameters, it is quite linear, e.g., figure 4.4b for the parameter *moderator density*. Hence, we need more discretized points on the *burnup* axis than on any other axes.

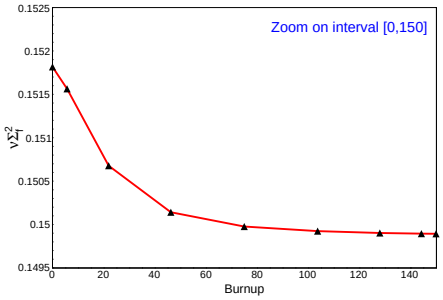
If we use a polynomial approximation of degree 8, hence using 9 Clenshaw-Curtis points on the *burnup* axis (these points are represented by small circles on the *burnup* axis in figure 4.4a), the first two discretized values obtained are: $0.0\,(MWd/t)$ and $3044.81\,(MWd/t)$. We can see that this discretization can not capture the information about cross-section variations due to strong variations at the low *burnup* values, that is caused by the *xenon* effect before its concentration reaches equilibrium. Since the low *burnup* is around the interval $[0, 150]$ while $3044.81 \gg 150$, we

therefore propose to sub-divide the whole interval $[0, 80000]$ into 3 sub-intervals: $[0, 150]$, $[150, 10000]$ and $[10000, 80000]$. For the case studied here (UOX assembly on the standard domain), we choose 9 Clenshaw-Curtis points for each sub-interval (there are thus two common points: 150 and 10000). The new discretization with 25 points in total is now well suited to the variation of the cross-section $\nu\Sigma_f^2$, as presented in figure 4.5.

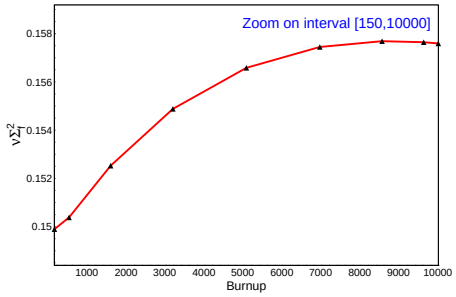
This technique, interval subdivision, can be used each time when the fine discretization can not capture cross-section variations.



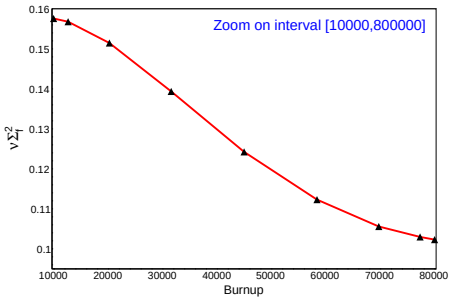
(a) 25 Clenshaw-Curtis points on [0,80000].



(b) 9 Clenshaw-Curtis points on [0,150].



(c) 9 Clenshaw-Curtis points on [150,10000].



(d) 9 Clenshaw-Curtis points on [10000,80000].

Figure 4.5 – Subdivision of the *burnup* axis (with 9 Clenshaw-Curtis points on each sub-interval) in order to capture the cross-section variation (case of the UOX assembly on the standard domain).

4.4.3.2 Post-analysis of evaluated errors

4.4.3.2.1 Histogram of evaluated errors In our work, each point $\mathbf{x}$ of the reference grid (in the phase space) has 5 coordinates: $x_1, \ldots, x_5$. In practice, we store successively these coordinates and the evaluated errors associated with the point $\mathbf{x}$ into files. This allows us to present and analyze the evaluated errors as a function of each parameter.

From such stored files, we present the evaluated errors as *histograms*. A histogram contains some bins used to divide the entire evaluated errors into adjacent intervals. The height of a bin represents the number of values (evaluated errors in our case) that fall into this bin. In our test, we want to

present percentages in order to maintain the scale (in %) when we change the number of entries; we thus normalize the bins by dividing the height of each bin by the total number of entries (the number of points in the reference grid). The representation of evaluated errors by histogram allows us to see how the evaluated errors distribute and which the error zones need to be analyzed.

In general, the centered histogram with small standard deviations is an ideal result, which means that most of the evaluated errors are around 0. Note however that an analysis of the deviated distributions can be exploited in order to determine where they come from and which parameters are responsible for this behavior. These analyses allow us to improve the accuracy and error distributions as we will show in the following section.

4.4.3.2.2 Post-analysis via histogram of evaluated errors The accuracy of the Tucker decomposition depends on the number of eigenvectors (or tensor directional basis functions) taken for each direction. It also depends on the fine discretization realized on each axis because tensor directional basis functions are constructed from these discretizations. Improving one of these two factors may improve the accuracy of our approach. Therefore, they are the solutions to the problems revealed by our analyses, as we will show here.

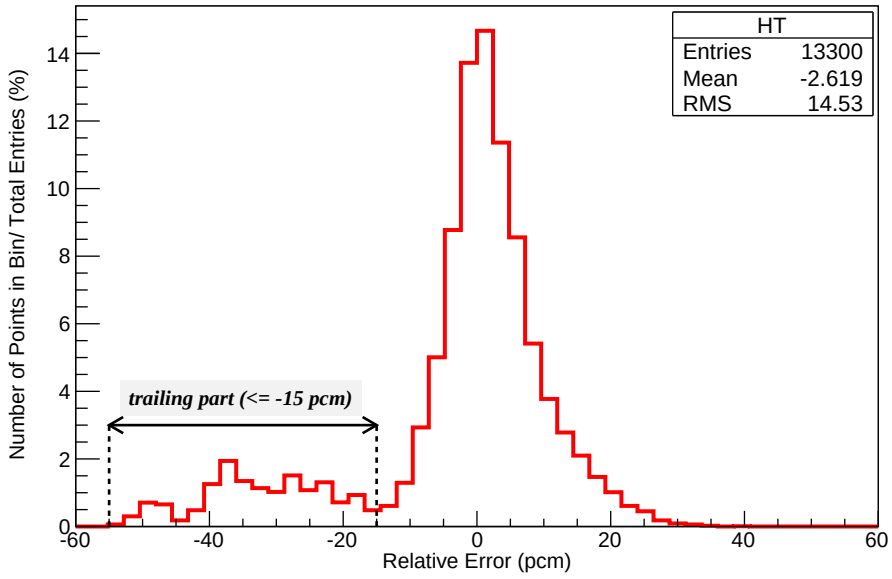
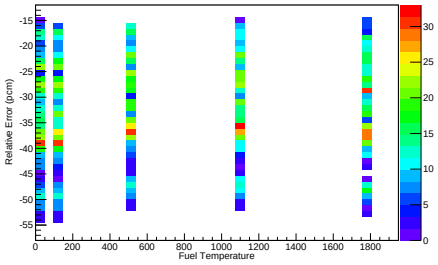


Figure 4.6 – Relative errors of Σ_t^1 (deviation distribution) before being improved.

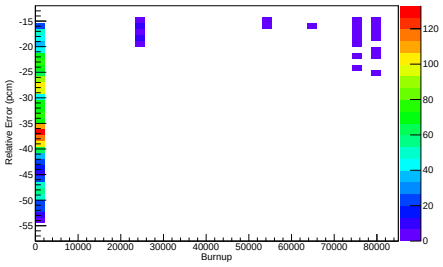
In order to illustrate our analysis techniques, we present an example in the case of the UOX-Gd assembly on the extended domain with the cross-section Σ_t^1 (fast group). The distribution of relative errors is shown in figure 4.6. This distribution varies in the interval [-55 pcm, 35 pcm] and looks like a Gaussian distribution with a trailing part ($\sim [-55\,pcm, -15\,pcm]$).

In order to improve the accuracy as well as the distribution of relative errors of the cross-section Σ_t^1, we will analyze the trailing part ($-55\,pcm \leq$ error $\leq -15\,pcm$) to see if its behavior depends on a particular parameter. Analysis per parameter is one of the method which helps us to see how the errors depend on each parameter and which specific values are involved.

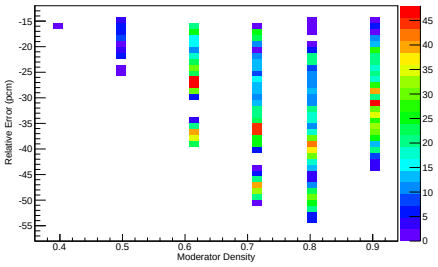
Applying this idea, we obtained the distribution of the trailing part as a function of respectively each parameter: *burnup, boron concentration, moderator density, fuel temperature* and *xenon level*.



(a) Trailing part ($\leq -15\,pcm$) as a function of *fuel temperature*.



(b) Trailing part ($\leq -15\,pcm$) as a function of *burnup*.



(c) Trailing part ($\leq -15\,pcm$) as a function of *moderator density*.

Figure 4.7 – Trailing part ($\leq -15\,pcm$) of Σ_t^1 (fast group) as a function of one parameter.

In figure 4.7, these error distributions are presented as the 2D-histograms, in the following manner:

— The error distributions are represented by the variations of colors in each *column inside* a figure. These columns are placed above each discretized value (used for the fine discretization of the analyzed parameter) to describe how an error value depends on this fixed discretized value while varying the values of other parameters.

— Each color corresponds to a number of points occurred (or density of points) as described in the column on the right of each figure.

From the error distributions shown in figure 4.7, we can see that:

— For each of the following parameters: *fuel temperature, boron concentration* and *xenon level*,

its discretized values are all presented, see figure 4.7a for an example about the parameter *fuel temperature*. In this figure, the error distributions are presented by 5 columns corresponding to all 5 discretized values of the parameter *fuel temperature*. We can see that these 5 columns are similar to each other in length and density distribution of occurred points (i.e. color pattern). Therefore, we can say that the error distributions are quite homogeneous and do not depend on a particular discretized value of the *fuel temperature* parameter. (The same conclusion for *boron concentration* and *xenon level* parameter from their corresponding distributions not shown here).

— For each of the following parameters: *burnup* and *moderator density*, the error distributions are quite different and vary as a function of only some particular values, for example, around 0 for the case of *burnup* (figure 4.7b), and around 0.8 for the case of *moderator density* (figure 4.7c).

Using this analysis, we can conclude that *burnup* and *moderator density* are major factors which are the cause of the trailing part. Hence, we propose to add more tensor directional basis functions on these two directions. Concretely, we can improve the accuracy with 7 tensor directional basis functions for burnup (instead of 4) and 4 tensor directional basis functions for moderator density (instead of 3). The new result is shown in figure 4.8.

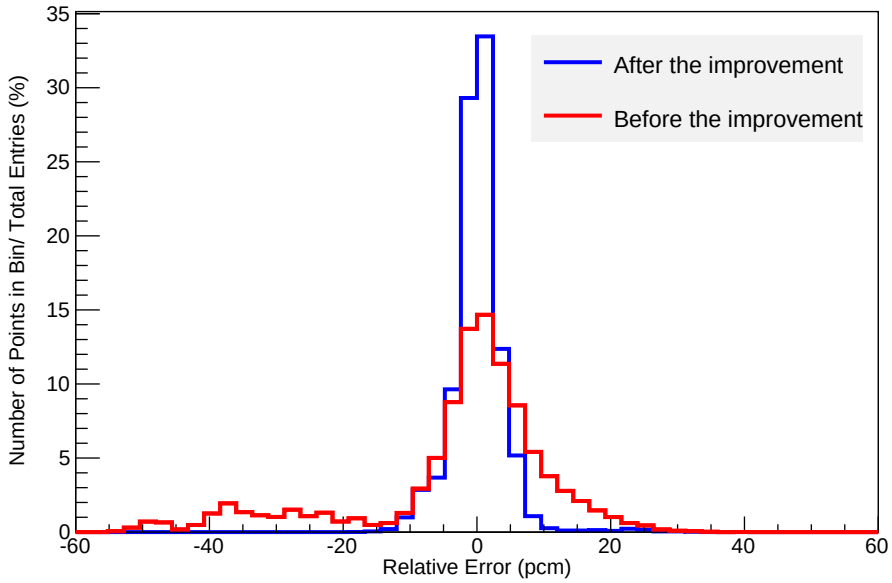


Figure 4.8 – Relative errors of Σ_t^1 before (in red) and after (in blue) being improved (number of tensor directional basis functions for *burnup* axis and *moderator density* axis are increased).

We see that the relative errors are improved, they vary in the interval [-15 pcm, 30 pcm] and

become more centered. For this improvement, we did not increase the number of discretized points on the Tucker grid in the direction of *burnup* nor *moderator density* axis, but we added more tensor directional basis functions for these axes. The reason we did not try to increase discretized points on these axes is that this technique requires more APOLLO2 calculations. If we want to improve the accuracy by changing the fine discretization, then the zones of the *burnup* around 0 and the *moderator density* around 0.8 need to be better discretized.

We summarize here the main steps in the post-analysis of evaluated errors:

- Finding the major parameters which are the cause of analyzed error zones:
  - Analyzing the distribution of evaluated errors as a function of each parameter.
  - Determining the parameters and the particular discretized values on which the accuracy does not satisfy our requirement.
- Once the major parameters are identified, accuracy can be improved by either of the following methods:
  - Adding more tensor directional basis functions in these directions. This can require new APOLLO2 calculations if the points used in the system for coefficients (4.7) are changed due to the new tensor directional basis functions.
  - Changing the axial discretization in order to take into account the parameter values that are responsible for large error. The new APOLLO2 calculations will be required on the new Tucker grid which corresponds to the new discretization and on the new points used in the system (4.7) for the coefficients of the Tucker decomposition .

In general, these improvements need more APOLLO2 calculations but the first solution (adding more tensor directional basis functions) often requires less calculations than the second one.

4.4.4 Cost of the Tucker decomposition compared to the multilinear interpolation

4.4.4.1 Number of APOLLO2 calculations

In the Tucker decomposition, the APOLLO2 calculations are performed in two stages:

- First, the APOLLO2 calculations are used for d Tucker grids in order to solve numerically the integral equation (4.3) which provides the tensor directional basis functions.
- Then, the APOLLO2 calculations are used for the cross-section values on the left hand side of the system (4.7) in order to determine the coefficients of the Tucker decomposition.

Therefore, the number of APOLLO2 calculations is:

$$\underbrace{\sum_{j=1}^{d} N_j * 2^{d-1}}_{\text{for d grids}} + \underbrace{\prod_{j=1}^{d} r_j^{max}}_{\text{for points on the left hand side of (4.7)}} \tag{4.10}$$

where N_j is the number of points of the fine discretization in direction j and r_j^{max} is the maximal number of tensor directional basis functions in direction j, over all cross-sections.

In the multilinear interpolation, we compute cross-section values on a Cartesian grid. Therefore, if each axis has N_j points, the number of APOLLO2 calculations for the multilinear interpolation is:

$$\prod_{j=1}^{d} N_j \tag{4.11}$$

4.4.4.2 Storage size in the neutron libraries

In the Tucker decomposition, the stored data for one cross-section includes eigenvector values for all directions (r_j eigenvectors for a direction j) and $R = \prod_{j=1}^{d} r_j$ coefficients. Therefore, the storage size is cross-section dependent. The number of stored floating points for each cross-section is:

$$\underbrace{\sum_{j=1}^{d} r_j * N_j}_{\text{for tensor directional basis functions}} + \underbrace{\prod_{j=1}^{d} r_j}_{\text{for coefficients}} \tag{4.12}$$

On the contrary, the storage sizes for the multilinear interpolation are the same for all cross-sections because we store all cross-section values performed on the Cartesian grid. Therefore, the number of stored floating points for the multilinear interpolation is:

$$\prod_{j=1}^{d} N_j \tag{4.13}$$

4.5 Proposed benchmarks

4.5.1 Reactivity

About the reconstruction of cross-sections, the following points must be noted:

— Cross-sections are not the ones used in the core code because they need to be corrected in order to take into account leakage, equivalence, historical correction, ...before entering the flux solver.

— Cross-sections are merely inputs for the flux solver.

We are therefore interested in outputs of the flux solver, such as: k_{eff} or reactivity. For a two-group diffusion theory and infinite medium, the neutron flux and the cross-sections are constant on the parameter-phase space. Therefore, with this simplified hypothesis, we do not need a flux solver, k_{eff} becomes now k_∞ and the following analytic formula are applied (see page 1172, 1173, 1221 of the book [Marguet, 2013]):

$$\text{reactivity} = 1 - \frac{1}{k_{eff}}, \text{with } k_{eff} = k_\infty = \frac{\nu\Sigma_f^1 * (\Sigma_t^2 - \Sigma_{s0}^{2\to 2}) + \nu\Sigma_f^2 * \Sigma_{s0}^{1\to 2}}{(\Sigma_t^1 - \Sigma_{s0}^{1\to 1}) * (\Sigma_t^2 - \Sigma_{s0}^{2\to 2}) - \Sigma_{s0}^{1\to 2} * \Sigma_{s0}^{2\to 1}} \tag{4.14}$$

A final indicator for the accuracy of the reactivity is expressed as follows:

$$\mathbf{e}_{\text{Reactivity}}(pcm) = \max_{\mathbf{x}\in \textit{reference grid}} |e_{\text{Reactivity, Absolute}}(\mathbf{x})| \tag{4.15}$$

where:

$$e_{\text{Reactivity, Absolute}}(\mathbf{x}) = \underbrace{(\frac{1}{k_\infty(\mathbf{x})} - \frac{1}{\tilde{k}_\infty(\mathbf{x})}) * 10^5}_{\text{Absolute Error (pcm)}} \tag{4.16}$$

4.5.2 Grids used in benchmarks

In the following benchmarks, the cross-sections depend on 5 parameters: *burnup, boron concentration, moderator density, fuel temperature* and *xenon level*. Therefore, we have 5 Tucker grids, each one corresponds to one direction in the Tucker decomposition.

For each benchmark, we use a reference grid (see section 4.4.2) in order to measure the accuracy of the Tucker decomposition.

4.5.3 Benchmark for the standard domain with UOX assembly

In this case, we consider an UOX assembly, 3.7% enrichment for a 900MWe-PWR (Pressurized Water Reactor). In the Tucker decomposition, we use 5 Tucker grids with a total of 720 points. Only the grid for the *burnup* axis is sub-divided into 3 intervals: [0,150], [150,10000] and [10000, 80000]. Each sub-interval has 9 Clenshaw-Curtis points. The other grids have 5 Clenshaw-Curtis points for the fine discretization without using subdivision. The reference grid described in section 4.4.2 contains 9300 points.

The accuracy (in pcm) of the Tucker decomposition is shown in table 4.1. We see that the relative errors of cross-sections are smaller than 100 pcm in general. These accuracies are better than ones of the multilinear interpolation employed in COCAGNE with the currently used number of points (shown in table 4.2).

We observed that the accuracy of the cross-section $\Sigma_{s0}^{2\to 1}$ is the worst result obtained by the Tucker decomposition. It is also the worst case for the multilinear interpolation. As we will see, these observations are valid for all benchmarks presented in this paper.

Cross Section	$\mathbf{e_{Tucker}}$ (pcm) formula (4.9) for cross-sections formula (4.15) for reactivity
Σ_t^1	28
Σ_t^2	40
Σ_a^1	38
Σ_a^2	49
$\nu\Sigma_f^1$	61
$\nu\Sigma_f^2$	79
$\Sigma_{s0}^{1\to1}$	27
$\Sigma_{s0}^{1\to2}$	19
$\Sigma_{s0}^{2\to1}$	100
$\Sigma_{s0}^{2\to2}$	15
Σ_f^1	66
Σ_f^2	74
Reactivity	387

Table 4.1 – Accuracy of the Tucker decomposition over 9300 points of the reference grid for the UOX assembly on the standard domain.

In order to achieve the accuracy in the table 4.1, we employed the number of APOLLO2 calculations and the storage size respectively presented in table 4.2 and table 4.3. They are compared to those of the multilinear interpolation.

Number of APOLLO2 calculations	
Tucker decomposition	Multilinear Interpolation
$\underbrace{720}_{\textit{for 5 Tucker grids}} + \underbrace{378}_{\text{for the system (4.7)}} = \mathbf{1098}$	**1782**

Table 4.2 – Comparison of the number of APOLLO2 calculations between the Tucker decomposition and the multilinear interpolation for the UOX assembly on the standard domain.

Cross section	**Storage (number of floats)**	
	Tucker	Multilinear
Σ_t^1	204	1782
Σ_t^2	192	idem
Σ_a^1	416	idem
Σ_a^2	355	idem
$\nu\Sigma_f^1$	388	idem
$\nu\Sigma_f^2$	290	idem
$\Sigma_{s0}^{1\to1}$	204	idem
$\Sigma_{s0}^{1\to2}$	529	idem
$\Sigma_{s0}^{2\to1}$	371	idem
$\Sigma_{s0}^{2\to2}$	192	idem
Σ_f^1	388	idem
Σ_f^2	290	idem

Table 4.3 – Comparison of the storage for each cross-section, between the Tucker decomposition and the multilinear interpolation for the UOX assembly on the standard domain.

We note that the number of APOLLO2 calculations (1782) in the table 4.2 for the multilinear interpolation comes from an optimized industrial process in COCAGNE whereas with the Tucker decomposition, this is not so much the case since this is a preliminary work. Even though, these results show that the Tucker decomposition reduced by 38% of the number of APOLLO2 calculations and by a factor of 3.4 to 8.7 of the storage size (depending on the studied cross-section).

4.5.4 Benchmarks for the extended domain

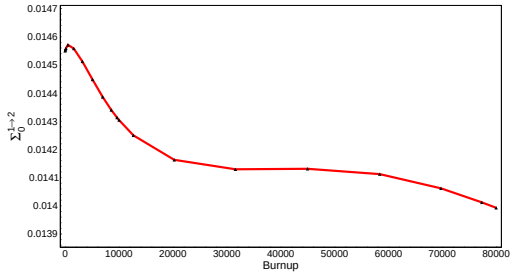
The extended domain is tricky for many reconstruction models, especially those involved with corrections from values computed around the nominal values (Taylor expansions, heuristics, etc.). The multilinear interpolation does not suffer from this problem. However, to have high quality cross-section, it requires a lot of points.

These following benchmarks are realized on an extended domain where the values of four parameters: *boron concentration* ($\sim [0\,ppm, 2200\,ppm]$), *moderator density* ($\sim [0.3\,g/cm^3, 1\,g/cm^3]$), *fuel temperature* ($\sim [10℃, 2000℃]$) and *xenon level* ($\sim [0, 3]$) are larger than those of the standard domain (the interval of *burnup* does not change). Again, precise values issued from the industrial scheme can not be shown here.

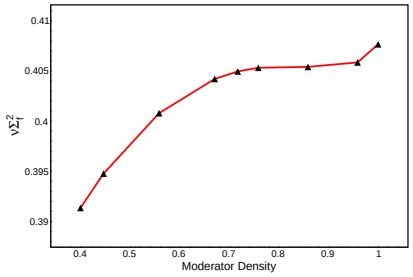
For all the following test cases on the extended domain, the Tucker decomposition employees the same 5 grids (each one corresponds to one direction). By chance, the number of total points on these 5 grids is the same in the standard domain case (720 points) but the discretizations are completely different. We summarize here *the fine discretization* (using the Clenshaw-Curtis points) for each parameter-grid:

- *Burnup* grid: sub-divided into 3 intervals: [0,150] with 3 points, [150,10000] with 9 points and [10000, 80000] with 9 points.
- *Moderator density* grid: sub-divided into 2 intervals, each sub-interval has 5 points.
- *Fuel temperature* grid: 5 points.
- *Boron concentration* grid: 7 points.
- *Xenon level*: 5 points.

In the Tucker grids used on the extended domain, subdivisions are applied to both the *burnup* axis and the *moderator density* axis. This comes from the method discussed in section 4.4.3.1 in order to capture the variation of cross-sections. The great variations (for the UOX and the MOX assemblies) as a function of *burnup* (presented in figure 4.9a) or *moderator density* (presented in figure 4.9b) explain why subdivisions on these two axes are needed.



(a) Variation of $\Sigma_{s0}^{1\to2}$ as a function of *burnup* (case of UOX assembly on extended domain).



(b) Variation of $\nu\Sigma_f^2$ as a function of *moderator density* (case of MOX assembly on extended domain).

Figure 4.9 – Variations of some cross-sections lead to subdivisions on the extended domain for the *burnup* axis and the *moderator density* axis.

4.5.4.1 UOX assembly on the extended domain

Once again, the UOX assembly, 3.7% enrichment is considered but on the extended domain. In this case, the reference grid contains 14700 points.

Evaluated errors for some cross-sections and for the reactivity is shown in table 4.4. We see that the relative errors are about 100 pcm, except for $\Sigma_{s0}^{2\to1}$ (464 pcm). These accuracies are better than those of the multilinear method for all cases shown in the table 4.4.

Cross Section	$\mathbf{e_{Tucker}}$ (pcm) formula (4.9) for cross-sections formula (4.15) for reactivity
Σ_t^1	9
Σ_t^2	86
Σ_a^1	64
Σ_a^2	97
$\nu\Sigma_f^1$	47
$\nu\Sigma_f^2$	104
$\Sigma_{s0}^{1\to1}$	8
$\Sigma_{s0}^{1\to2}$	23
$\Sigma_{s0}^{2\to1}$	464
$\Sigma_{s0}^{2\to2}$	84
Σ_f^1	43
Σ_f^2	102
Reactivity	179

Table 4.4 – Accuracy of the Tucker decomposition over 14700 points of the reference grid for the UOX assembly on the extended domain.

The comparisons of the Tucker decomposition with the multilinear interpolation on the number of APOLLO2 calculations and on the storage size are respectively shown in table 4.5 and table 4.6. The number of APOLLO2 calculations (3060) used for the multilinear interpolation is already optimized in the code COCAGNE. We see that the number of APOLLO2 calculations is reduced by 14.7% and the storage size is reduced by a factor of 1.65 to 11.16 by using the Tucker decomposition.

Number of APOLLO2 calculations	
Tucker decomposition	Multilinear Interpolation
$\underbrace{720}_{\textit{for 5 Tucker grids}} + \underbrace{1890}_{\text{for the system (4.7)}} = \mathbf{2610}$	**3060**

Table 4.5 – Comparison of the number of APOLLO2 calculations between the Tucker decomposition and the multilinear interpolation for the UOX assembly on the extended domain.

Cross section	**Storage (number of floats)**	
	Tucker	Multilinear
Σ_t^1	343	3060
Σ_t^2	343	idem
Σ_a^1	527	idem
Σ_a^2	866	idem
$\nu\Sigma_f^1$	537	idem
$\nu\Sigma_f^2$	636	idem
$\Sigma_{s0}^{1\to1}$	343	idem
$\Sigma_{s0}^{1\to2}$	854	idem
$\Sigma_{s0}^{2\to1}$	1849	idem
$\Sigma_{s0}^{2\to2}$	274	idem
Σ_f^1	537	idem
Σ_f^2	636	idem

Table 4.6 – Comparison of the storage for each cross-section, between the Tucker decomposition and the multilinear interpolation for the UOX assembly on the extended domain.

4.5.4.2 UOX-Gd assembly on the extended domain

This benchmark studies an UOX-Gd assembly for a 1300MWe-PWR. The reference grid contains 13300 points.

The accuracy of the Tucker decomposition is shown in table 4.7. In this test case, the relative errors of the cross-sections as well as the absolute errors of the reactivity are the worst compared to the other cases (UOX, MOX (see later)). But in these results, we did not perform any improvement

(e.g., change the discretization, add more points on some axis, etc., as shown in the section 4.4.3.2) in order to get better results. We see in the table 4.7 that the relative errors are smaller than 200 pcm in general, except for $\Sigma_{s0}^{2\to1}$ (550 pcm) and Σ_a^2 (327 pcm). (However, these results are all better than accuracies obtained by the multilinear interpolation).

Cross Section	$\mathbf{e_{Tucker}}$ (pcm) formula (4.9) for cross-sections formula (4.15) for reactivity
Σ_t^1	54
Σ_t^2	93
Σ_a^1	77
Σ_a^2	327
$\nu\Sigma_f^1$	76
$\nu\Sigma_f^2$	175
$\Sigma_{s0}^{1\to1}$	38
$\Sigma_{s0}^{1\to2}$	150
$\Sigma_{s0}^{2\to1}$	550
$\Sigma_{s0}^{2\to2}$	87
Σ_f^1	79
Σ_f^2	175
Reactivity	535

Table 4.7 – Accuracy of the Tucker decomposition over 13300 points of the reference grid used for UOX-Gd assembly on the extended domain.

In order to maintain the order of accuracy, the multilinear model must take more points for its grid (3060 → 3960 points), compared to previous benchmarks. Table 4.8 and 4.9 show that, by using the Tucker decomposition, the number of APOLLO2 calculations is reduced by 29.4% and the storage size is reduced by a factor of 3.6 to 20.84, compared to the multilinear interpolation.

Number of APOLLO2 calculations	
Tucker decomposition	Multilinear Interpolation
$\underbrace{720}_{\textit{for 5 Tucker grids}} + \underbrace{1440}_{\text{for the system (4.7)}} = \mathbf{2610}$	**3960**

Table 4.8 – Comparison of the number of APOLLO2 calculations between the Tucker decomposition and the multilinear interpolation for UOX-Gd assembly on the extended domain.

Cross section	**Storage (number of floats)**	
	Tucker	Multilinear
Σ_t^1	478	3960
Σ_t^2	190	idem
Σ_a^1	1070	idem
Σ_a^2	537	idem
$\nu\Sigma_f^1$	1098	idem
$\nu\Sigma_f^2$	410	idem
$\Sigma_{s0}^{1\to1}$	478	idem
$\Sigma_{s0}^{1\to2}$	537	idem
$\Sigma_{s0}^{2\to1}$	791	idem
$\Sigma_{s0}^{2\to2}$	190	idem
Σ_f^1	1098	idem
Σ_f^2	410	idem

Table 4.9 – Comparison of the storage for each cross-section, between the Tucker decomposition and the multilinear interpolation for UOX-Gd assembly on the extended domain.

4.5.4.3 MOX assembly on the extended domain

In this case, the MOX assembly (a mixture of uranium enriched with 2.5% and plutonium) is considered. For this benchmark, we use a reference grid containing 14000 points.

Cross Section	$\mathbf{e_{Tucker}}$ (pcm) formula (4.9) for cross-sections formula (4.15) for reactivity
Σ_t^1	15.05
Σ_t^2	58.19
Σ_a^1	20.23
Σ_a^2	28.61
$\nu\Sigma_f^1$	11.92
$\nu\Sigma_f^2$	24.80
$\Sigma_{s0}^{1\to1}$	12.45
$\Sigma_{s0}^{1\to2}$	11.45
$\Sigma_{s0}^{2\to1}$	482.94
$\Sigma_{s0}^{2\to2}$	62.02
Σ_f^1	11.21
Σ_f^2	25.11
Reactivity	93.00

Table 4.10 – Accuracy of the Tucker decomposition over 14000 points of the reference grid for MOX assembly on the extended domain

We show in table 4.10 the evaluated errors of the Tucker decomposition for some cross-sections

and for the reactivity. We see that the relative errors are smaller than 100 pcm, except for the worst case $\Sigma_{s0}^{2\to1}$ (482 pcm). The absolute errors of the reactivity are smaller than 93 pcm. This is the best approximation results that we have (on the extended domain), not only for the accuracy but also for the number of APOLLO2 calculations and the storage size in the neutron library.

Number of APOLLO2 calculations	
Tucker decomposition	Multilinear Interpolation
$\underbrace{720}_{\textit{for 5 Tucker grids}} + \underbrace{720}_{\text{for the system (4.7)}} = \mathbf{1440}$	**3060**

Table 4.11 – Comparison of the number of APOLLO2 calculations between the Tucker decomposition and the multilinear interpolation for MOX assembly on the extended domain.

The number of APOLLO2 calculations in table 4.11 and the storage size in table 4.12 are compared to those of the multilinear interpolation. These results show that, by using the Tucker decomposition, the number of APOLLO2 calculations is reduced by 52.9% and the storage size is reduced by a factor of 3.3 to 17.2.

Cross section	**Storage (number of floats)**	
	Tucker	Multilinear
Σ_t^1	178	3060
Σ_t^2	223	idem
Σ_a^1	446	idem
Σ_a^2	446	idem
$\nu\Sigma_f^1$	286	idem
$\nu\Sigma_f^2$	343	idem
$\Sigma_{s0}^{1\to1}$	178	idem
$\Sigma_{s0}^{1\to2}$	446	idem
$\Sigma_{s0}^{2\to1}$	920	idem
$\Sigma_{s0}^{2\to2}$	223	idem
Σ_f^1	286	idem
Σ_f^2	343	idem

Table 4.12 – Comparison of the storage for each cross-section, between the Tucker decomposition and the multilinear interpolation for MOX assembly on the extended domain.

4.5.5 Discussion

With the results obtained in the proposed benchmarks, we showed that the Tucker decomposition allows us to reduce the pre-calculated data as well as the storage size (compared to the multilinear model) while achieving high accuracy.

The subdivision (described in section section 4.4.3.1) in order to capture the variations of cross-sections is applied to some axes, e.g. *burnup* axis is subdivided into three sub-intervals. Hence, for the *burnup* axis, we have three Lagrange polynomials (one for each sub-interval) which lead to the global reconstructed cross-sections are not of class C^1. This problem can be solved if for each segment, we choose a higher degree polynomial and absorb the remaining degree of freedom with the continuity of the function and first derivative.

We can also improve the accuracy of the Tucker decomposition by taking more tensor directional basis functions for some axes (see the section 4.4.3.2 with the example of the cross-section Σ_t^1 where we added tensor directional basis functions more for the burnup axis and the moderator density axis). Unfortunately, since the tensor directional basis functions are only the approximations of the eigenfunctions issued from the Karhunen-Loève decomposition, this solution could be unsuccessful in some cases. Hence, the method to find automatically the optimal number of tensor directional basis functions needs to be investigated.

4.6 Conclusion and perspective

The Tucker decomposition described in this paper allows us to efficiently reconstruct the macroscopic neutron cross-sections. It also allows us to analyze the results obtained as a function of each parameter in order to improve the accuracy.

Using the Tucker decomposition, we reduce the number of APOLLO2 calculations (from 15% to 50%), the storage size in the neutron libraries (from 1.5 times to 20 times), compared to the multilinear interpolation. While pre-calculated data are significantly reduced, we still achieve high accuracy for the reconstructed cross-sections: around 100 pcm in relative errors, in general.

With a simplified hypothesis about an infinite medium and the two-group energy theory, the reactivity is analytically calculated. We can therefore verify the reactivity accuracy without using the flux solver. In general, the maximal absolute error for the reactivity are about 100 pcm to 600 pcm (depending on the benchmark). It is worth noting that the accuracy obtained using the Tucker decomposition is the same for the standard and the extended domain. No deterioration have been observed when moving from the standard to the extended domain.

However, the Tucker decomposition has some limitations. The evaluation step performed at the core code level could be expensive in CPU time because the Lagrange interpolation is used to reconstruct the tensor directional basis functions from the eigenvectors. Moreover, the Tucker decomposition does not ensure the positivity of cross-sections. The linearity relation, e.g. $\Sigma_t^1 = \Sigma_a^1 + \Sigma_{s0}^{1\to 2} + \Sigma_{s0}^{1\to 1}$ is also not exactly preserved (except of course if we define the approximation of Σ_t^1 as being the sum of the three approximations).

In future, we plan to study criteria which allow us to eliminate (*a priori* and *a posteriori*) the less important coefficients $\mathbf{a}$ in the representation of the Tucker decomposition. It means that the accuracy of the Tucker decomposition could be of the same order while many coefficients could be eliminated. This idea is feasible because the Tucker decomposition is constructed from "sorted" tensor directional basis functions (via the order of eigenvalues, e.g. decreasing order). Hence, there *a priori* exist less important coefficients, e.g. coefficients associated with the product of the last tensor directional basis functions. Therefore, such an elimination could lead to a similar accuracy while the number of APOLLO2 calculations and the storage size in the libraries would be further reduced.

Chapter 5

Sparse representation in Tucker decomposition applied to the reconstruction of neutron cross-sections

This is a submitted paper with Y.Maday, M.Guillo and P.Guérin. Its reference in the manuscript is [Luu et al., 2016c].

Abstract

In a previous work, we have presented and used the Tucker decomposition for the approximate reconstruction of neutron cross-sections from look up tables. We have shown that the Tucker decomposition achieves high accuracy while reducing significantly pre-calculated data as well as their storage, in comparison to the multi-linear interpolation that is classically used in this community. However, the number of coefficients in the Tucker decomposition still suffers the so-called *curse of dimensionality*. This becomes the major limiting factor for dealing with the increase of data in our approach. Therefore, reducing the number of these coefficients is a natural quest and the goal of this paper is to provide a solution in this direction. We propose some techniques to get a sparse representation in the Tucker decomposition, applied to the problem of reconstructing neutron cross-sections. These techniques are classified into two categories: *a priori* and *a posteriori*. Using these techniques, we can reduce the amount of required data while keeping a similar accuracy of our initial approach.

Keywords: Tucker decomposition, low-rank tensor approximation, sparse representation/sparsity, sparse grid, cross-sections reconstruction, neutronics

5.1 Introduction

In our previous works [Luu et al., 2016b] [Luu et al., 2016a], the Tucker decomposition [Tucker, 1966], [Hackbusch, 2012] [De Lathauwer et al., 2000] is used in order to evaluate neutron cross-sections which are multi-variate functions in rather high dimension $d \geq 5$. The Tucker decomposition is a low-rank tensor approximation method where a multi-variate function is approximated by a limited linear combination of well chosen tensor products of one-dimensional functions.

Let $f(\mathbf{x}) = f(x_1, \ldots, x_d)$ be a d-variate function defined on $\Omega = \Omega_1 \times \ldots \times \Omega_d \subset \mathbb{R}^d$, the Tucker decomposition applied to this function is expressed as follows:

$$f(\mathbf{x}) \approx \tilde{f}(x_1, \ldots, x_j, \ldots, x_d) = \sum_{i_1=1}^{r_1} \ldots \sum_{i_j=1}^{r_j} \ldots \sum_{i_d=1}^{r_d} \underbrace{\mathbf{a}_i}_{\text{coefficient}} \prod_{j=1}^{d} \underbrace{\varphi_{i_j}^{(j)}(x_j)}_{\text{tensor directional basis function}} \tag{5.1}$$

where:

(i) The multi-integer i is a short hand notation for $i = i_1 \ldots i_j \ldots i_d$

(ii) $\varphi_{i_j}^{(j)}$ are one-dimensional functions and called *tensor directional basis functions*. Here, we have a total of r_j tensor directional basis functions for each direction j, $1 \leq j \leq d$, that are chosen from eigenfunctions of the following Hilbert-Schmidt operator $K_f^{(j)}$:

$$\begin{aligned} & K_f^{(j)}\Phi(x_j) = \int_{\Omega_j} \int_{\Omega_{y_j}} f(x_j, y_j) f(x'_j, y_j) \Phi(x'_j) dx'_j dy_j, \ \forall x_j \in \Omega_j \\ \text{with} \quad & y_j \in \Omega_{y_j} := \Omega_1 \times \ldots \times \Omega_{j-1} \times \Omega_{j+1} \times \ldots \times \Omega_d \end{aligned} \tag{5.2}$$

If we denote by $\{\varphi_k^{(j)}\}_k$ these eigenfunctions, they are the solutions of the following eigenproblem:

$$\boxed{K_f^{(j)} \varphi_k^{(j)}(x_j) \equiv \int_{\Omega_j} \int_{\Omega_{y_j}} f(x_j, y_j) f(x'_j, y_j) \varphi_k^{(j)}(x'_j) dx'_j dy_j = \lambda_k^{(j)} \varphi_k^{(j)}(x_j), \forall x_j \in \Omega_j} \tag{5.3}$$

where $\lambda_k^{(j)}$ are eigenvalues associated with eigenfunctions $\varphi_k^{(j)}$ of the operator $K_f^{(j)}$. A particular property of the eigenvalues is that they decrease quickly in general: $\lambda_1^{(j)} \geq \lambda_2^{(j)} \geq \ldots \geq \lambda_N^{(j)} \geq \ldots \geq 0$.

(iii) $\mathbf{a}_i = \mathbf{a}_{i_1 \ldots i_j \ldots i_d}$ are called *coefficient*. We have a total of $R = \prod_{j=1}^{d} r_j$ coefficients. Each coefficient associates with a tensor product $\prod_{j=1}^{d} \varphi_{i_j}^{(j)}$ of tensor directional basis functions.

In our original papers, we have retained all the coefficients in (5.1), their set

$$I_{\text{full}} = \{i |\ i \text{ is index of } \mathbf{a}_i \text{ employed in (5.1)}\} \tag{5.4}$$

is of cardinal R. The associated coefficients are determined by solving the following system of R

linear equations:

$$\begin{cases} \sum_{i=1}^{R} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{1_j}) = f(\mathbf{x}_1) \\ \dots \\ \sum_{i=1}^{R} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{t_j}) = f(\mathbf{x}_t) \\ \dots \\ \sum_{i=1}^{R} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{R_j}) = f(\mathbf{x}_R), \end{cases} \tag{5.5}$$

where $\mathbf{x}_t = (x_{t_1}, \dots, x_{t_j}, \dots, x_{t_d}) \in \Omega$, $1 \leq t \leq R$. A greedy algorithm (see [Maday et al., 2013]) is used to construct the above set of points $\{\mathbf{x}_t\}_{t=1}^{t=R}$.

We can rewrite this system under a matrix form:

$$PA = B \tag{5.6}$$

where $B = (f(\mathbf{x}_t)) \in \mathbb{R}^R$, $P = (\prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{t_j})) \in \mathcal{M}_R(\mathbb{R})$ and $A = (\mathbf{a_i}) \in \mathbb{R}^R$.

We see that on the right hand side of the system (5.5), we need R values of the function f on the set $\{\mathbf{x}_t\}_{t=1}^{R}$. In our applications to neutron cross-sections, the right hand side, consisting of $(f(\mathbf{x}_t)) \in \mathbb{R}^R$ is built from the APOLLO2 code [Sanchez et al., 2010]. Such a calculation (a 2D transport problem on a fine geometry) may take a rather small CPU time but for classical methods like bilinear interpolation, $R \simeq 1000$ calculations are required in practice, leading an expensive CPU time in total. Therefore, we want to limit the number of APOLLO2 calculations as much as possible.

In order to understand the complexity of the Tucker algorithm, let us assume, for simplicity that the number of tensor directional basis functions is the same for all directions, i.e. $r_j = r$, $1 \leq j \leq d$, we obtain:

— The cardinal of coefficients ($\#\{\mathbf{a}_i\}$) is equal to $R = \prod_{j=1}^{d} r$, which is of the order of $\mathcal{O}(r^d)$ (exponential function of dimension).

— The number of APOLLO2 calculations required for cross-section values on the right hand side of the system (5.5) is also $O(r^d)$.

— The major factor in the storage size is the number of coefficients, still $O(r^d)$.

In our previous papers [Luu et al., 2016b], [Luu et al., 2016a], we have shown that the Tucker decomposition allows to diminish notably the number of calls to APOLLO2, however, the complexity still suffers from the "curse of dimensions" since it still scales like $O(r^d)$. From the construction of the tensor directional basis functions provided by a Karhunen-Loeve (KL) expansion and the ordering of the eigenpair in (5.3), we expect that the coefficients of the tensors $\prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_j)$ should decay rapidly for large values of the indices. Thus, eliminating the less important coefficients is one solution which allows us to reduce APOLLO2 calculations as well as storage size for the Tucker decomposition. This procedure is referred to as *sparsity* or *elimination*.

In the present work, we introduce two categories of approaches: *a posteriori* and *a priori* sparsity/elimination, applied to coefficients in the Tucker decomposition. *A posteriori* sparsity means

that we eliminate the coefficients based on their actual values already determined by the system (5.5). *A priori* sparsity implies that we eliminate some coefficients without computing them because we predict that they are small.

A posteriori sparsity is very simple because once we computed values of all coefficients, we can eliminate the smallest ones by some criteria. This method reduces both the storage size and the reconstruction of the approximation but not the number of APOLLO2 calculations, because we need to solve the complete system (5.5) of size R.

A priori sparsity is based on the expected decay of the eigenvalues of the eigenproblem (5.3). It reduces at the same time the number of APOLLO2 calculations required for the system (5.5) and the storage size.

The paper is organized in the following manner:

Section 2 describes the methodology used for *a posteriori sparsity* and *a priori sparsity*. Some mathematical analyses of the Tucker approximation will be shown in order to estimate the error of this approach and propose some sparsity criteria. In section 3, we show applications of our sparsity methods to the problem of reconstructing neutron cross-sections on some proposed benchmarks. In section 4 we draw some conclusions and perspectives.

5.2 Methodology

5.2.1 Mathematical analyses of the Tucker approximation

Let us remind that, out of the eigenproblem for the Hilbert-Schmidt operator $K_f^{(j)}$, comes a set of orthonormal eigenfunctions $(\varphi_k^{(j)})$, that constitute a basis of $L^2(\Omega_j)$. We then can write $f(\mathbf{x}) = \sum_{k=1}^{\infty} \sqrt{\lambda_k^{(j)}} \varphi_k^{(j)}(x_j) \psi_k^{(j)}(y_j)$ which is the Karhunen-Loève expansion). Hence $||f||_{L^2}^2 = \sum_{k=1}^{\infty} \lambda_k^{(j)}$.

Let us remind that we assume that the $\lambda_k^{(j)}$ are ranked in decreasing order: $\lambda_1^{(j)} \geq \lambda_2^{(j)} \geq \ldots \geq \lambda_k^{(j)} \geq \ldots 0$.

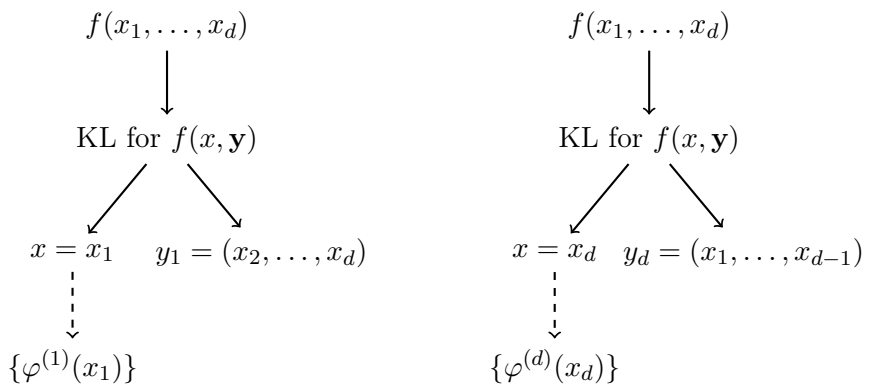
The Karhunen-Loève decomposition is iteratively used for each j and leads to d integral problems, each one is written as follows:

$$\int_{\Omega_x} \int_{\mathbf{\Omega}_{y_j}} f(x, y_j) f(x', y_j) \varphi(x') dx' dy_j = \lambda \varphi(x), \forall x \in \Omega_x \tag{5.7}$$

Solving these integral problems allows us to determine tensor directional basis functions $\{\varphi^{(j)}(x_j)\}$ for each direction j $(1 \leq j \leq d)$. We only keep from this decomposition, the family of eigenvectors in the x_j variable. The extension process is illustrated in figure (5.1).

If we denote by

$$X_K^{(j)} = span\{\varphi_k^{(j)}(x_j),\ 1 \leq k \leq K\} = \{\sum_{k=1}^{K} \alpha_k \varphi_k^{(j)}(x_j) : \alpha_k \in \mathbb{R}\}$$



(a) Construction of tensor directional basis functions for the direction $x = x_1$.

(b) Construction of tensor directional basis functions for the direction $x = x_d$.

Figure 5.1 – Construction of tensor directional basis functions for all directions using an extension of the Karhunen-Loève decomposition (KL).

and

$$\mathcal{X}_K^{(j)} = X_K^{(j)} \otimes L^2(\mathbb{R}^{d-1}) = \{\sum_{k=1}^{K} \alpha_k(y_j)\varphi_k^{(j)}(x_j) : \alpha_k \in L^2(\Omega_{y_j})\} \subset L^2(\mathbb{R}^d)$$

and by $\Pi_K^{(j)}$ the L^2-projection over $X_K^{(j)}$ (that can be identified with the L^2-projection over $\mathcal{X}_K^{(j)}$), we have:

$$\Pi_K^{(j)} f = \sum_{k=1}^{K} \sqrt{\lambda_k^{(j)}}\varphi_k^{(j)}(x_j)\psi_k^{(j)}(y_j)$$

Hence

$$||f - \Pi_K^{(j)} f||_{L^2}^2 = \sum_{k=K+1}^{\infty} \lambda_k^{(j)} = \frac{\sum_{k=K+1}^{\infty} \lambda_k^{(j)}}{\sum_{k=1}^{\infty} \lambda_k^{(j)}} ||f||_{L^2}^2$$

Our Tucker approximation consists in finding an approximation of f into $\bigotimes_{j=1}^{d} X_R^{(j)} = X_R$. In order to guide our understanding of the approximation in X_R, let us introduce, for any finite dimensional space $\mathbf{Y}_L$ in $L^2(\mathbb{R})$, the L^2 projection operator denoted as $\Pi_{\mathbf{Y}_L}$ on $\mathbf{Y}_L$. The first lemma is straightforward:

Lemma 5.2.1. *We have* $\cap_j \mathcal{X}_R^{(j)} = \bigotimes_j X_R^{(j)}$, *and*

$$\Pi_{\boldsymbol{X}_R} = \Pi_{\boldsymbol{X}_R^{(1)}} \circ \Pi_{\boldsymbol{X}_R^{(2)}} \circ \ldots \circ \Pi_{\boldsymbol{X}_R^{(d)}}$$

Lemma 5.2.2. *The best approximation of* f *in* X_R *for the* L^2*-norm is obtained by the* L^2 *projection*

$\Pi_{X_R}(f)$ *and it satisfies:*

$$||f - \Pi_{X_R}(f)||^2_{L^2} \leq \sum_j \Big[\frac{\sum_{k=R+1}^{\infty} \lambda_k^{(j)}}{\sum_{k=1}^{\infty} \lambda_k^{(j)}} \Big] ||f||^2_{L^2}$$

Proof. It is an easy matter to deduce from Lemma 5.2.1 that:

$$\begin{aligned} I_d - \Pi_{\mathbf{X}_R} = (I_d - \Pi^{(1)}_{\mathbf{X}^{(1)}_R}) + \Pi^{(1)}_{\mathbf{X}^{(1)}_R} \circ (I_d - \Pi^{(2)}_{\mathbf{X}^{(2)}_R}) + \dots \\ + \Pi^{(1)}_{\mathbf{X}^{(1)}_R} \circ \Pi^{(2)}_{\mathbf{X}^{(2)}_R} \circ \dots \circ \Pi^{(d-1)}_{\mathbf{X}^{(d-1)}_R} \circ (I_d - \Pi^{(d)}_{\mathbf{X}^{(d)}_R}) \end{aligned}$$

Since each projection operator $\Pi^{(j)}_{\mathbf{X}^{(j)}_R}$ is stable in the L^2 norm, we have:

$$\begin{aligned} ||f - \Pi_{\mathbf{X}_R} f||^2_{L^2(\Omega)} &\leq \sum_j ||f - \Pi^{(j)}_{\mathbf{X}^{(j)}_R}||^2_{L^2(\Omega)} \\ &\leq \sum_j \frac{\sum_{k=R+1}^{\infty} \lambda_k^{(j)}}{\sum_{k=1}^{\infty} \lambda_k^{(j)}} ||f||^2_{L^2(\Omega)} \end{aligned}$$

□

5.2.2 *A posteriori* sparsity for coefficients

5.2.2.1 Proposed criteria for *a posteriori* sparsity

Let us assume that all R coefficients $\mathbf{a}_i$ of the Tucker decomposition have already been determined by solving the system (5.5). Our *a posteriori*-criteria for sparsifying the coefficients is to keep only the largest coefficients and eliminate the smallest ones like in classical "non-linear" approximation. Concretely, these criteria are normalized and based on some norms (via value of n) as follows:

$$\text{keeping only } \mathbf{a}_i \text{ if } \frac{|\mathbf{a}_i|^n}{\sum_{i=1}^{R} |\mathbf{a}_i|^n} > \epsilon, \ n \in \mathbb{N}^* \tag{5.8}$$

where ϵ and n are user's parameter chosen either after a "learning" process or proposed by some analysis.

Noting that, the smaller ϵ is, the more $\mathbf{a}_i$ are kept, leading to a smaller reduction of the number of coefficients.

We denote by $I_{\text{posteriori}}$ the set of indexes of coefficients kept after *a posteriori* sparsity:

$$I_{\text{posteriori}} = \{i |\ i \text{ is the index of } \mathbf{a}_i \text{ satisfying the criterion (5.8)}\} \tag{5.9}$$

5.2.2.2 Values of coefficients after *a posteriori* elimination

Applying this heuristic, the Tucker decomposition (5.1) contains only the terms associated with the coefficients satisfying the criterion (5.8). The coefficients appearing in this sparsified expansion have the same values as the initial solution (while the eliminated coefficients are now equal to zero). Therefore, the *a posteriori* Tucker decomposition becomes:

$$f(\mathbf{x}) \approx \tilde{f}(x_1, \ldots, x_d) = \sum_{i \in I_{\text{posteriori}}} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_j) \tag{5.10}$$

and we can evaluate the error due to the sparsification because we know the size of the coefficients that have been eliminated.

5.2.3 *A priori* sparsity for coefficients

5.2.3.1 Proposed criteria for *a priori* sparsity

On the contrary to the *a posteriori* approach, with *a priori* sparsity we do not compute all the coefficient values. The idea is that now, we only compute those coefficients that are solutions of the system (5.5) where the components of the product of the tensor directional basis functions $\varphi_{i_j}^{(j)}$ are not all big. Indeed, in each direction, the relevance of a tensor directional basis function *in a direction* j is quantified by the value of the associated eigenvalue, this results in Ω the relevance of a product of tensor directional basis functions $\varphi_{i_j}^{(j)}$ is related to the product of the associated eigenvalues.

In order to understand why this idea of sparseness is working, let us consider what we loose by replacing the L^2 approximation of f over X_R (obtained by the product $\Pi_{\mathbf{X}_R} f = \Pi_{\mathbf{X}_R^{(1)}} \Pi_{\mathbf{X}_R^{(2)}} \ldots \Pi_{\mathbf{X}_R^{(d)}} f$) where we additionally get rid of the part

$$[Id - \Pi_{\mathbf{X}_{R/2}^{(1)}}] \circ [Id - \Pi_{\mathbf{X}_{R/2}^{(2)}}] \circ \ldots \circ [Id - \Pi_{\mathbf{X}_{R/2}^{(d)}}] f$$

Lemma 5.2.3. *We have the following bound*

$$||\Pi_{\boldsymbol{X}_R} f - [Id - \Pi_{\boldsymbol{X}_{R/2}^{(1)}}] \circ [Id - \Pi_{\boldsymbol{X}_{R/2}^{(2)}}] \circ \ldots \circ [Id - \Pi_{\boldsymbol{X}_{R/2}^{(d)}}] f||_{L^2} \leq \prod_j \Big[\frac{\sum_{k=R/2+1}^{\infty} \lambda_k^{(j)}}{\sum_{k=1}^{\infty} \lambda_k^{(j)}} \Big]^{1/2} ||f||_{L^2}$$

Proof. Let us consider the case where $d = 2$, the extension to larger values being straightforward. We thus consider $[Id - \Pi_{\mathbf{X}_{R/2}^{(1)}}] \circ [Id - \Pi_{\mathbf{X}_{R/2}^{(2)}}] f$ where $f = \sum_{k=1}^{\infty} \sqrt{\lambda_k^{(2)}} \varphi_k^{(2)}(x_2) \psi_k^{(2)}(y_2)$. We this have

$$[Id - \Pi_{\mathbf{X}_{R/2}^{(2)}}] f = \sum_{k=R/2+1}^{\infty} \sqrt{\lambda_k^{(2)}} \varphi_k^{(2)}(x_2) \psi_k^{(2)}(y_2)$$

The natural question is now to understand what is $[Id - \Pi_{\mathbf{X}_{R/2}^{(1)}}]\psi_k^{(2)}(y_2)$? The answer is provided by recalling that

$$\psi_k^{(2)}(y_2) = \int_{\Omega_2} f(x_2, y_2)\varphi_k^{(2)}(x_2)dx_2$$

hence

$$[Id - \Pi_{\mathbf{X}_{R/2}^{(1)}}]\psi_k^{(2)}(y_2) = [Id - \Pi_{\mathbf{X}_{R/2}^{(1)}}] \int_{\Omega_2} f(x_2, y_2)\varphi_k^{(2)}(x_2)dx_2$$

using the fact that the operator $\Pi_{\mathbf{X}_{R/2}^{(1)}}$ commutes with any linear action in the x_2 direction, in particular with the integration over Ω_2 yielding

$$[Id - \Pi_{\mathbf{X}_{R/2}^{(1)}}]\psi_k^{(2)}(y_2) = \int_{\Omega_2} \left[[Id - \Pi_{\mathbf{X}_{R/2}^{(1)}}]f(x_2, y_2)\right] \varphi_k^{(2)}(x_2)dx_2$$

Using the orthonormality of the $\varphi_k^{(2)}$ we get the result. □

Concretely, we have proposed two criteria for the a priori elimination of coefficients: one based on the following idea described in section 7.6.4 of the book [Hackbusch, 2012] about sparse grid approach (page 212 - 216), related to the value of the indexes. The second one, in the spirit of the analysis of the previous lemma based on the values of the eigenvalues.

The approximation f_N of f (with f expressed by series expansion $f = \sum_{i\in\mathbb{N}^d} \mathbf{a}_i \bigotimes_{j=1}^d \varphi_{i_j}^{(j)}$) comes from a sparse grid with the following representation (see the page 213 of [Hackbusch, 2012]):

$$f = \sum_{i\in\mathbb{N}^d} \mathbf{a}_i \bigotimes_{j=1}^d \varphi_{i_j}^{(j)} \simeq f_N = \sum_{\{i \,|\, \prod_{j=1}^d i_j < N\}} \mathbf{a}_i \bigotimes_{j=1}^d \varphi_{i_j}^{(j)} \text{ with } i \equiv [i_1, \ldots, i_j, \ldots, i_d]$$

Here N is related to the final level of a sparse grid and only indexes i_j satisfying $\prod_{j=1}^d i_j < N$ are used in the sparse representation f_N of f.

Inspired by this relation with index i satisfying $\prod_{j=1}^d i_j < N$, we propose a normalized criterion (by dividing by r_j) for *a priori* sparsity for coefficients:

$$\text{keeping only } \mathbf{a}_i = \mathbf{a}_{i_1 \ldots i_j \ldots i_d} \text{ with } \prod_{j=1}^d \frac{i_j}{r_j} < \epsilon \tag{5.11}$$

Since the eigenvalues have the same role as the indexes in the order definition of tensor directional basis functions, we can therefore propose another criterion as follows:

$$\text{keeping only } \mathbf{a}_i = \mathbf{a}_{i_1 \ldots i_j \ldots i_d} \text{ with } \prod_{j=1}^d \frac{\lambda_{i_j}^{(j)}}{\lambda_0^{(j)}} > \epsilon \tag{5.12}$$

Here, ϵ ($\epsilon \leq 1$) is the user's parameter.

It should be noted that in (5.11), the smaller ϵ is, the less $\mathbf{a}_i$ are kept, leading to a high reduction score (contrary to the criteria (5.8) used for *a posteriori* sparsity and the criterion (5.12) based on eigenvalues).

We denote by I_{priori} the set of indexes of coefficients kept after *a priori* sparsity:

$$I_{\text{priori}} = \{i|\ i \text{ is the index of } \mathbf{a}_i \text{ satisfying the criterion (5.11) or (5.12}\} \tag{5.13}$$

5.2.3.2 Values of coefficients after *a priori* elimination

After realizing *a priori* sparsity for coefficients via the criterion (5.11) or (5.12), we have a total of $R^{'}$ coefficients for the Tucker decomposition (5.1) with $R^{'} < R = \prod_{j=1}^{d} r_j$.

We propose two ways to determine these R' coefficients:

— They are solutions of a new square system (the number of equations is equal to the number of unknowns). This system is similar to (5.5) but its size is R':

$$\begin{cases} \sum_{i\in I_{\text{priori}}} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{1_j}) = f(\mathbf{x}_1) \\ \dots \\ \sum_{i\in I_{\text{priori}}} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{t_j}) = f(\mathbf{x}_t) \\ \dots \\ \sum_{i\in I_{\text{priori}}} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{R'_j}) = f(\mathbf{x}_{R'}) \end{cases} \tag{5.14}$$

where $\mathbf{x}_t = (x_{t_1}, \dots, x_{t_j}, \dots, x_{t_d}) \in \Omega \subset \mathbb{R}^d$, $1 \le t \le R'$.

By this way, we reduced the number of calculations required for the right hand side of the system (5.5). However, how to choose the new set of points $\{\mathbf{x}_t\}_{t=1}^{R'}$ (used on the right hand side of the new system (5.14)) such that we obtain a similar accuracy to our initial approach? Until now, we have not found a satisfactory solution. The greedy algorithm (employed to construct the points on the left hand side of the initial system (5.5)) could be again considered.

— These coefficients are solutions of a new system which is similar to the initial one (5.5) but we have less unknown coefficients (R' instead of R) and the matrix P in (5.6) is now rectangular with smaller size ($P \in \mathcal{M}_{R,R'}(\mathbb{R})$ instead of $P \in \mathcal{M}_R(\mathbb{R})$). The new system aims at solving:

$$\begin{cases} \sum_{i\in I_{\text{priori}}} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{1_j}) = f(\mathbf{x}_1) \\ \dots \\ \sum_{i\in I_{\text{priori}}} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{t_j}) = f(\mathbf{x}_t) \\ \dots \\ \sum_{i\in I_{\text{priori}}} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{R_j}) = f(\mathbf{x}_R) \end{cases} \tag{5.15}$$

at best, where $\mathbf{x}_t = (x_{t_1}, \dots, x_{t_j}, \dots, x_{t_d}) \in \Omega \subset \mathbb{R}^d$, $1 \le t \le R$.

The system (5.15) implies that we keep all points of the set $\{\mathbf{x}_t\}_{t=1}^{R}$ used on the right hand side of (5.5) while on the left hand side of (5.5), the eliminated coefficients are equal to zero. Using this way, we do not reduce the number of calculations required on the right hand side of the initial system (5.5) and the system (5.15) is an overdetermined system

(more equations than unknowns). Therefore, we propose to compute the coefficients $\mathbf{a}_i$ by adding auxiliary condition based on least square methods [Miller, 2006], [Williams, 1990], [Lawson and Hanson, 1995]. The coefficients $\mathbf{a}_i$ kept after *a priori* sparsity are now determined by solving the following system with a constraint that minimizes errors in 2-norm (sum of square errors):

$$\begin{cases} \sum_{i\in I_{\text{priori}}} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{1_j}) = f(\mathbf{x}_1) \\ \dots \\ \sum_{i\in I_{\text{priori}}} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{t_j}) = f(\mathbf{x}_t) \\ \dots \\ \sum_{i\in I_{\text{priori}}} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{R_j}) = f(\mathbf{x}_R) \\ \\ \textit{With the constraint:} \\ \min_{\mathbf{a}_i} \sum_{t=1}^{R} [f(\mathbf{x}_t) - \sum_{i\in I_{\text{priori}}} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_{t_j})]^2 \end{cases} \tag{5.16}$$

In this case, the *a priori* Tucker decomposition becomes:

$$f(\mathbf{x}) \approx \tilde{f}(x_1, \dots, x_d) = \sum_{i\in I_{\text{priori}}} \mathbf{a}_i \prod_{j=1}^{d} \varphi_{i_j}^{(j)}(x_j) \tag{5.17}$$

with the least square constraint described in (5.16) for $\mathbf{a}_i$.

5.2.4 Sparsity criteria in practice

In practice, with the proposed criteria (5.8) for *a posteriori* sparsity and (5.11) or (5.12) for *a priori* sparsity, we do:

— Either we choose directly the value of the parameter ϵ;

— Or we impose a desired percentage for the number of eliminated coefficients for which the value of ϵ is implicitly deduced.

The second way seems to be more practical than the first one since the accuracy is probably more sensitive to the criteria using the parameter ϵ (a bit variation of ϵ can lead to an important change of storage and accuracy). Although both methods are mathematically equivalent, we prefer to use the second one for *a priori* sparsity. The reason is that, with *a priori* sparsity, we have less available information than *a posteriori* sparsity and the parameter ϵ is often small (< 1) which is more difficult to control than the parameter percentage.

In order to get better understanding about the second way, we describe it with more details here. Considering the criteria (5.11) for an example, we do:

— Computing the product $\prod_{j=1}^{d} \frac{i_j}{r_j}$ for each coefficient $\mathbf{a}_i = \mathbf{a}_{i_1 \dots i_j \dots i_d}$.

— Sorting these products, e.g. in increasing order.

— Applying the imposed percentage on the sorted products $\prod_{j=1}^{d} \frac{i_j}{r_j}$ in order to eliminate the less important products.

— Eliminating the coefficients corresponding to the eliminated products.

Concretely, we know that a coefficient $\mathbf{a}_i$ is *a priori* kept if its associated product $\prod_{j=1}^{d} \frac{i_j}{r_j}$ is small. Therefore, for example, if we want to reduce 40% of the number of coefficients, we just eliminate the coefficients corresponding to the last 40% of the products $\prod_{j=1}^{d} \frac{i_j}{r_j}$ sorted in the increasing order.

5.3 Applications to the reconstruction of neutron cross-sections

5.3.1 Context of applications

At EDF-R&D (Électricité De France-Recherche & Développement), a multi-linear interpolation model is currently employed to reconstruct neutron cross-sections. Two main steps are used in this reconstruction:

— First, neutron cross-sections are pre-calculated on a sample of pre-selected points. Those values are stored in files, called *neutron libraries*.

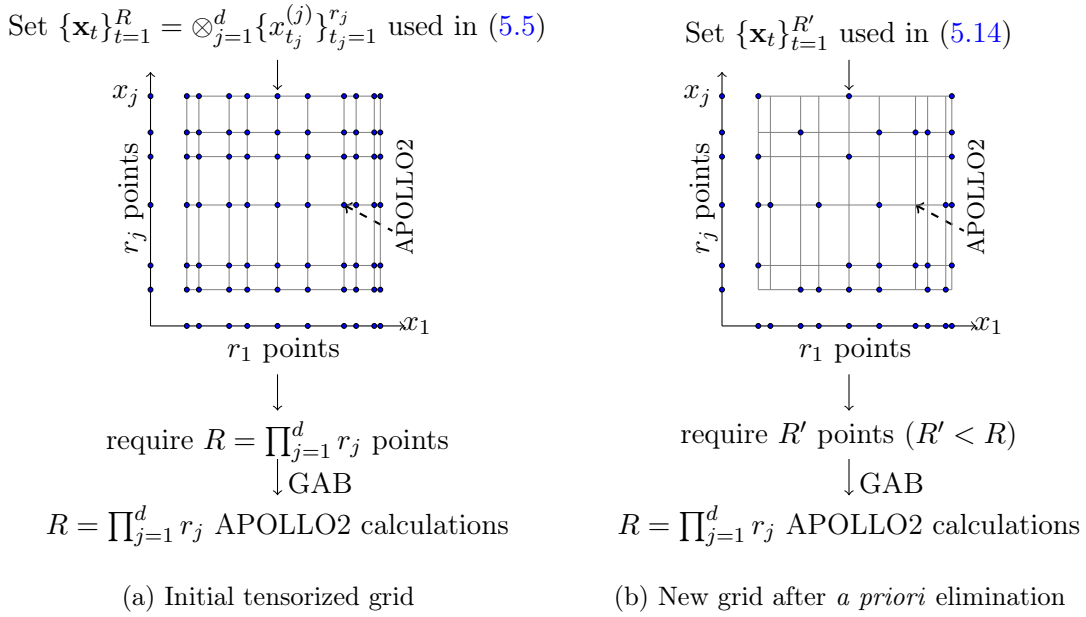
— Next, neutron cross-sections are reconstructed from pre-calculated data in neutron libraries by an evaluation method (currently, multi-linear interpolation).

Neutron cross-sections at a given point $\mathbf{x}$ are computed using APOLLO2 code [Sanchez et al., 2010] developed at CEA. APOLLO2 requires an input deck describing the geometry, material, solver options, the scheme for calculation... EDF-R&D has developed a code, GAB, that generates automatically APOLLO2 input decks for different points $\mathbf{x}_i$. GAB computes cross sections for all those $\mathbf{x}_i$ using distributed CPUs on a cluster, and finally gather all results inside hierarchical library files. One important thing is that the user does not indicate the $\mathbf{x}_i$; rather it defines d discretized axis, and the $\mathbf{x}_i$ are all nodes of the tensorized grid obtained with those axes. This is exactly what we want for the multi-linear interpolation but this is not optimal for other models which require only a subset of this grid. Therefore, we had to optimize the construction of the set of points $\{\mathbf{x}_t\}_{t=1}^{R}$ (used on the left hand side of the system (5.5)) such that this set is a tensor product of 1D-sets $\{x_{t_j}^{(j)}\}_{t_j=1}^{r_j}$:

$$\{\mathbf{x}_t\}_{t=1}^{R} = \otimes_{j=1}^{d} \{x_{t_j}^{(j)}\}_{t_j=1}^{r_j} \tag{5.18}$$

Where $\{x_{t_j}^{(j)}\}_{t_j=1}^{r_j}$ is a 1D-set containing r_j points chosen in the direction j (see the illustration in figure 5.2a).

Now, with *a priori* sparsity methods discussed in section 5.2.3.2, if we want to reduce the number of APOLLO2 calculations, we need to determine values of kept coefficients ($\{\mathbf{a}_i\}_{i \in I_{\text{priori}}}$) by solving



(a) Initial tensorized grid

(b) New grid after *a priori* elimination

Figure 5.2 – Using the calculation scheme of GAB, the number of APOLLO2 calculations is not reduced after *a priori* procedure if the new grid does not correspond to a tensor structure.

the new system (5.14). This system depends on a new set of points $\{\mathbf{x}_t\}_{t=1}^{R'}$ on its right hand side, therefore, we must also determine this set. Assuming that this new set $\{\mathbf{x}_t\}_{t=1}^{R'}$ is included in the initial set $\{\mathbf{x}_t\}_{t=1}^{R}$ defined by (5.18), we need to eliminate $R - R'$ points in the grid corresponding to $\{\mathbf{x}_t\}_{t=1}^{R}$ points. However, in general, this elimination does not assure that the new grid $\{\mathbf{x}_t\}_{t=1}^{R'}$ (see figure 5.2b) has a tensor structure. In that case, the number of APOLLO2 calculations is not reduced because of GAB restriction.

Taking advantage of *a priori* sparsity method to save the APOLLO2 calculations, GAB would be modified so that we can define only the points we are interested in.

5.3.2 Description of benchmarks

5.3.2.1 Cross-section notion and reactivity

In nuclear physics, neutron cross-sections are denoted by Σ_r^g where r designates the reaction kind and g designates the energy group. We refer here to some cross-sections that are used in the core code COCAGNE like the `macro totale` - Σ_t^g, the `macro absorption` - Σ_a^g, `macro fission` - Σ_f^g and the `macro nu*fission` - $\nu\Sigma_f^g$. A particular case, the `macro scattering` also depends on anisotropy order o, departure energy group g and arrival energy group g', denoted by $\Sigma_{so}^{g\to g'}$. In our work, two energy groups are considered: fast group ($g = 1$) and thermal group ($g = 2$). For more explanation on reactor physics, we refer to the books [Marguet, 2013] and [Lewis and Miller, 1984].

Cross-sections are merely inputs for the neutron flux solver. We are more interested in the output of the flux solver, for instance: k_{eff} or reactivity. For a two-groups diffusion theory and infinite medium, flux and cross-sections are spatially constant and k_{eff} is denoted by k_∞. Therefore, with this simplified hypothesis, the following analytic formula can be applied (see page 1172, 1173, 1221 of the book [Marguet, 2013]):

$$\text{reactivity} = 1 - \frac{1}{k_{eff}}, \text{with } k_{eff} = k_\infty = \frac{\nu\Sigma_f^1 * (\Sigma_t^2 - \Sigma_{s0}^{2\to2}) + \nu\Sigma_f^2 * \Sigma_{s0}^{1\to2}}{(\Sigma_t^1 - \Sigma_{s0}^{1\to1}) * (\Sigma_t^2 - \Sigma_{s0}^{2\to2}) - \Sigma_{s0}^{1\to2} * \Sigma_{s0}^{2\to1}} \tag{5.19}$$

5.3.2.2 Evaluated errors

In order to measure the accuracy of our reconstruction method, we use a tensorized grid, referred to as *reference grid*. This grid is very fine and its points are constructed relatively randomized in order to have a lot of different points compared to points of grids used in the Tucker decomposition. Cross-section values performed by APOLLO2 code on the reference grid are *exact values* (Σ_r^g) and cross-section values reconstructed using Tucker decomposition and Lagrange interpolation are *approximated values* ($\widetilde{\Sigma}_r^g$).

We use the following indicators expressed in *pcm* unit (1 pcm = 10^{-5}) to measure the final evaluated error of the Tucker decomposition:

— For the reconstruction of a cross-section Σ_r^g:

$$e_{\Sigma_r^g}(pcm) = \max_{\mathbf{x}\in reference\ grid} |e_{\Sigma_r^g,\,\text{Relative}}(\mathbf{x})| \tag{5.20}$$

where:

$$e_{\Sigma_r^g,\,\text{Relative}}(\mathbf{x}) = \underbrace{\frac{\Sigma_r^g(\mathbf{x}) - \tilde{\Sigma}_r^g(\mathbf{x})}{\max_{\mathbf{x}} |\Sigma_r^g(\mathbf{x})|} * 10^5}_{\text{Relative Error (pcm)}}, \quad \mathbf{x} \in reference\ grid$$

— For the reconstruction of the reactivity:

$$\mathbf{e}_{\text{Reactivity}}(pcm) = \max_{\mathbf{x}\in reference\ grid} |e_{\text{Reactivity, Absolute}}(\mathbf{x})| \tag{5.21}$$

where:

$$e_{\text{Reactivity, Absolute}}(\mathbf{x}) = \underbrace{(\frac{1}{k_\infty(\mathbf{x})} - \frac{1}{\tilde{k}_\infty(\mathbf{x})}) * 10^5}_{\text{Absolute Error (pcm)}} \tag{5.22}$$

5.3.2.3 Proposed benchmarks

In this paper, we use the same benchmarks as described in a previous paper [**?**] (where we showed the results of the Tucker decomposition without sparse representation for coefficients). In

those benchmarks, cross-sections depend on 5 parameters: *burnup, boron concentration, moderator density, fuel temperature* and *xenon level*.

The benchmarks were proposed for different fuel assemblies: UOX, UOX-Gd, MOX. They are re-used here in order to test the capacity of data reduction by using our sparsity methods. The number of points in the reference grid (to measure reconstruction errors) for each benchmark is respectively: 14700 points for UOX, 13300 points for UOX-Gd and 14000 points for MOX.

5.3.3 Results with *a posteriori* sparsity

We present here results of *a posteriori* sparsity obtained with three benchmarks UOX, UOX-Gd, MOX. We used the criterion (5.8) with $n = 1$ and $\epsilon = 10^{-6}$ for UOX and MOX while ϵ for UOX-Gd is slightly smaller: $\epsilon = 5.5 * 10^{-7}$ (in order to get a similar ratio reduction/accuracy as other cases). The reduction of the number of coefficients is shown in table 5.1. We see that the reduction (in percentage) is around 50% in most cases except for some cross-sections (Σ_t^2, $\Sigma_{s0}^{2\to1}$, $\Sigma_{s0}^{2\to2}$) in the case of the assembly UOX-Gd. In fact, the UOX-Gd is a difficult test case in general because of the complexity of the variation of cross-sections, compared to other cases (UOX, MOX). This can lead to coefficients that do not decrease fast. In this case, we can only remove a small part of these coefficients with *a posteriori* elimination (17.5% for $\Sigma_{s0}^{2\to1}$,≈ 30% for Σ_t^2 and $\Sigma_{s0}^{2\to2}$ in UOX-Gd case).

	#{a_i}-UOX			#{a_i} - UOX-Gd			#{a_i}-MOX		
	initial	after elimination $\epsilon = 10^{-6}$ (in (5.8))	***reduction*** (%)	initial	after elimination $\epsilon = 5.5 * 10^{-7}$ (in (5.8))	***reduction*** (%)	initial	after elimination $\epsilon = 10^{-6}$ (in (5.8))	***reduction*** (%)
Σ_t^1	180	75	***58***	324	93	***71***	72	37	***48***
Σ_t^2	192	62	***67***	72	49	***31***	96	48	***50***
Σ_a^1	360	125	***65***	864	183	***78***	288	74	***74***
Σ_a^2	675	138	***79***	360	155	***56***	288	88	***69***
$\nu\Sigma_f^1$	360	115	***68***	900	288	***68***	144	51	***64***
$\nu\Sigma_f^2$	450	129	***71***	240	121	***49***	192	73	***61***
$\Sigma_{s0}^{1\to1}$	180	73	***59***	324	90	***72***	72	37	***48***
$\Sigma_{s0}^{1\to2}$	630	161	***74***	360	206	***42***	288	85	***70***
$\Sigma_{s0}^{2\to1}$	1620	210	***87***	600	495	***17***	720	146	***79***
$\Sigma_{s0}^{2\to2}$	128	60	***53***	72	48	***33***	96	49	***48***
Σ_f^1	360	164	***54***	900	283	***68***	144	51	***64***
Σ_f^2	450	123	***72***	240	121	***49***	192	72	***62***

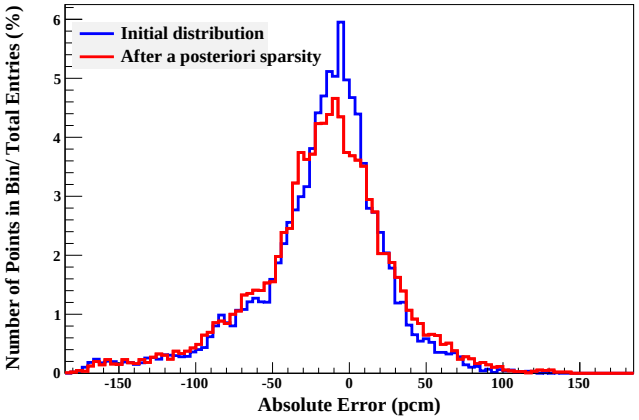
Table 5.1 – Reduction of the number of coefficients by *a posteriori* sparsity method with the criterion (5.8) (where $n = 1$), for different benchmarks: UOX, UOX-Gd, MOX.

The change of accuracy is presented in table 5.2. These results show that we obtain a similar accuracy for the reconstruction of cross-sections after reducing by around 50% of the coefficients in the Tucker decomposition. We observe that accuracy after the elimination is sometimes a bit better than before, e.g. Σ_a^1 in the UOX case (before: 64.92 pcm, after: 62.15 pcm). One of the reasons should be that the final error is measured by the infinite norm which returns us a punctual value (defined by "max"). Results could be different when using another norm, e.g. L^2-norm.

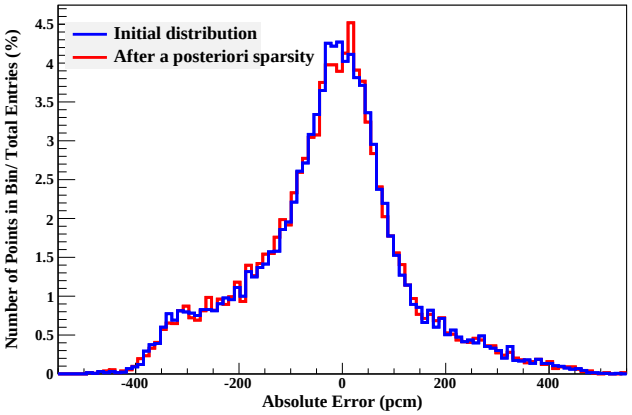
	e_{Tucker} (pcm) (formula (5.20) for cross-sections) (formula (5.21) for the reactivity)					
	UOX		**UOX-Gd**		**MOX**	
	initial	after elimination	initial	after elimination	initial	after elimination
Σ_t^1	9.09	9.48	54.20	54.23	15.05	16.48
Σ_t^2	86.67	87.87	93.32	93.47	58.19	58.15
Σ_a^1	64.92	62.15	77.53	112.40	20.23	22.07
Σ_a^2	97.09	98.07	327.35	321.08	28.61	28.94
$\nu\Sigma_f^1$	47.41	46.78	76.43	154.93	11.92	11.64
$\nu\Sigma_f^2$	104.35	105.98	175.61	176.32	24.80	25.30
$\Sigma_{s0}^{1\to1}$	8.56	9.10	38.71	61.25	12.45	13.65
$\Sigma_{s0}^{1\to2}$	23.98	30.91	150.04	149.69	11.45	13.49
$\Sigma_{s0}^{2\to1}$	464.06	464.22	550.81	550.65	482.94	483.78
$\Sigma_{s0}^{2\to2}$	84.72	84.60	87.11	87.33	62.02	62.11
Σ_f^1	43.50	43.35	79.38	140.83	11.20	11.18
Σ_f^2	102.70	104.67	175.35	176.29	25.11	25.78
Reactivity	**179.02**	**182.45**	**535.93**	**549.21**	**93.00**	**93.13**

Table 5.2 – The change of accuracy of the Tucker decomposition after *a posteriori* elimination with the reduction presented in table 5.1.

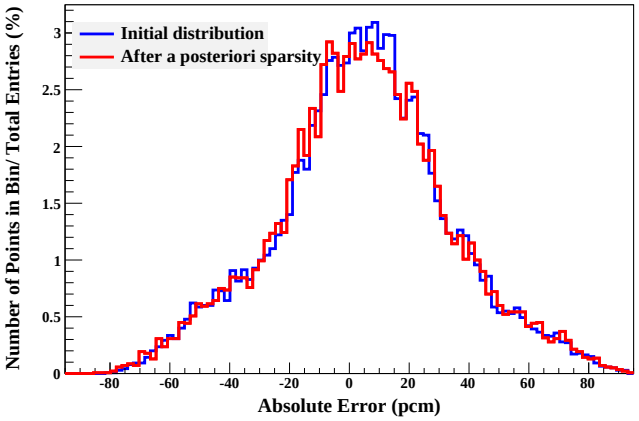
We illustrate also in figure 5.3 the distribution of absolute error for different $\mathbf{x}$ (defined by (5.22)) of the reactivity for the three benchmarks: UOX (figure 5.3a), UOX-Gd (figure 5.3b) and MOX (figure 5.3c). In these figures, one can see that distributions are almost identical before and after *a posteriori* elimination.



(a) Evaluated errors of the reactivity - UOX case



(b) Evaluated errors of the reactivity - UOX-Gd case



(c) Evaluated errors of the reactivity - MOX case

Figure 5.3 – Distribution of absolute errors of the reactivity for three benchmarks: UOX, UOX-Gd and MOX. Before and after *a posteriori* elimination, the distributions are almost identical while the number of coefficients used in the Tucker decomposition for cross-sections are reduced by around 50 % (see table 5.1).

5.3.4 Results with *a priori* sparsity

In the results presented here, the coefficients in *a priori* sparsity of the Tucker decomposition are determined by the system (5.16) with the least square method. Results obtained by using the criterion (5.11) and (5.12) where ϵ is implicitly defined via an imposed percentage for the number of eliminated coefficients (see section 5.2.4).

With the number of initial coefficients presented in table 5.1, we show in table 5.3 the elimination percentage imposed for each cross-section of the three benchmarks: UOX, UOX-Gd, MOX. Imposed percentage is around 60% for UOX and MOX, 40% for UOX-Gd, see table 5.3.

	Reduction of $\#\{\mathbf{a}_i\}$ (%) (by *a priori* sparsity methods (5.11 and (5.12)))		
	UOX	**UOX-Gd**	**MOX**
Σ_t^1	40	40	61
Σ_t^2	50	20	60
Σ_a^1	60	40	60
Σ_a^2	40	40	60
$\nu\Sigma_f^1$	60	40	60
$\nu\Sigma_f^2$	60	40	60
$\Sigma_{s0}^{1\to1}$	40	40	61
$\Sigma_{s0}^{1\to2}$	60	40	60
$\Sigma_{s0}^{2\to1}$	60	20	60
$\Sigma_{s0}^{2\to2}$	60	30	60
Σ_f^1	60	40	60
Σ_f^2	60	40	60

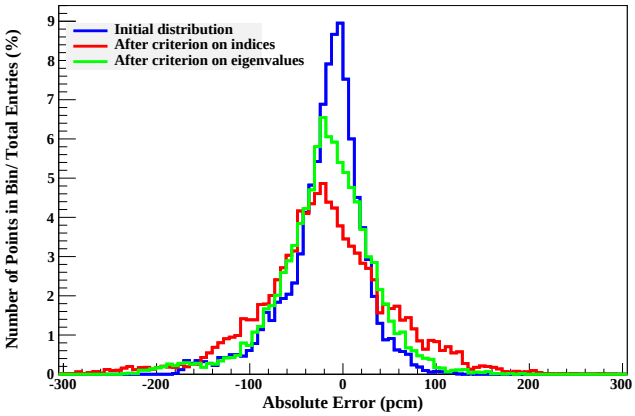
Table 5.3 – Imposed elimination percentages for the criterion (5.11) and (5.12) for *a priori* sparsity of coefficients, realized on different benchmarks: UOX, UOX-Gd, MOX.

	e_{Tucker} (pcm) (formula (5.20) for cross-sections) (formula (5.21) for the reactivity)								
	UOX			**UOX-Gd**			**MOX**		
	initial	elimination with (5.11)	elimination with (5.12)	initial	elimination with (5.11)	elimination with (5.12)	initial	elimination with (5.11)	elimination with (5.12)
Σ_t^1	9.09	10.72	10.52	54.20	54.01	54.11	15.05	15.57	17.39
Σ_t^2	86.67	87.02	86.72	93.32	93.72	93.68	58.19	63.26	59.94
Σ_a^1	64.92	56.76	55.90	77.53	60.15	77.53	20.23	20.33	21.66
Σ_a^2	97.09	97.09	97.09	327.35	322.58	325.10	28.61	30.43	30.17
$\nu\Sigma_f^1$	47.41	45.62	48.54	76.43	73.67	75.03	11.92	17.74	13.00
$\nu\Sigma_f^2$	104.35	106.21	105.56	175.61	175.17	176.18	24.80	32.46	30.02
$\Sigma_{s0}^{1\to1}$	8.56	9.98	9.87	38.71	38.95	39.30	12.45	15.38	16.74
$\Sigma_{s0}^{1\to2}$	23.98	23.92	23.99	150.04	151.75	152.67	11.45	16.93	12.47
$\Sigma_{s0}^{2\to1}$	464.06	464.05	464.28	550.81	535.28	535.20	482.94	482.79	483.01
$\Sigma_{s0}^{2\to2}$	84.72	81.73	86.88	87.11	86.26	87.45	62.02	71.90	63.57
Σ_f^1	43.50	39.24	39.97	79.38	76.33	77.82	11.20	17.75	12.23
Σ_f^2	102.70	103.39	103.30	175.35	173.79	175.81	25.11	32.64	28.16
Reactivity	**179.02**	**301.74**	**233.71**	**535.93**	**534.42**	**519.24**	**93.00**	**108.77**	**140.23**

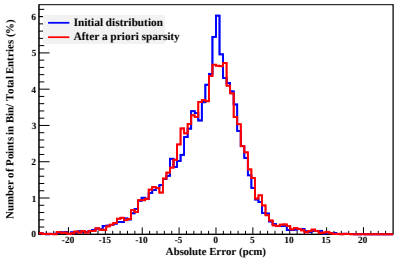
Table 5.4 – The change of accuracy of the Tucker decomposition after *a priori* elimination with the reduction presented in table 5.3.

The change of accuracy is presented in table 5.4. These results show that we obtain a similar accuracy for the reconstruction of cross-sections. However, for the UOX case, the reactivity accuracy is decreased by 60pcm (criterion (5.12)) and 120 pcm (criterion (5.11)) while the max errors of each reconstructed cross-section is very close before and after *a priori* elimination.

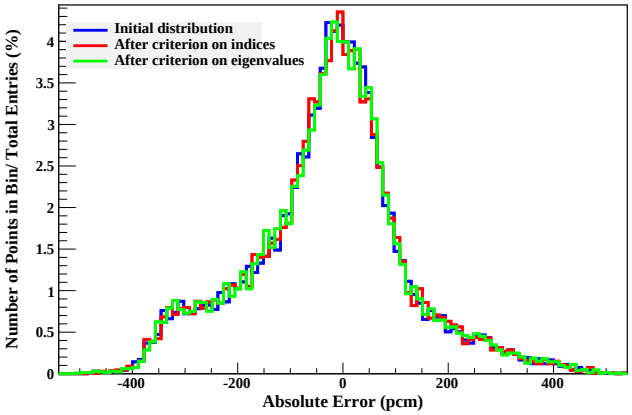
We illustrate in figure 5.5 the distribution of absolute error (defined by (5.22)) of the reactivity for the three benchmarks: UOX (figure 5.4a), UOX-Gd (figure 5.5a), MOX (5.5b). In these figures, the distributions look similar before and after *a priori* elimination for both UOX-Gd and MOX cases. This is consistent with the table 5.4. However, the error distribution for the reactivity is quite different for UOX case. Not only the max has increased a lot (that is what we have in table 5.4) but the dispersion has also increased. In order to understand these increases, we need more information than the max of errors given in the table 5.4. Looking at the error distribution of cross-sections in the UOX case, we see that the increasing dispersion for the reactivity accuracy can be caused by the degradation of some reconstructed cross-sections, even their max accuracy are almost identical. For example, the cross-section $\Sigma_{s0}^{1\to 2}$ has the same max accuracy before and after *a priori* elimination (they are around 23 pcm in table 5.4) but the dispersion increased after this elimination (see figure 5.4b). In order to improve accuracy, we should have probably reduced less coefficients for some cross-sections.



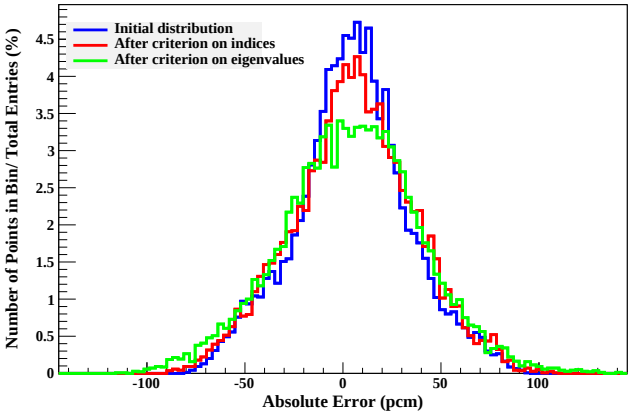
(a) Evaluated errors of the reactivity - UOX case



(b) Evaluated errors of `macro scattering` $\Sigma_{s0}^{1\to 2}$- UOX case: ***dispersion increased after a priori elimination even max accuracy is preserved (23 pcm).***



(a) Evaluated errors of the reactivity - UOX-Gd case



(b) Evaluated errors of the reactivity - MOX case

Figure 5.5 – Distribution of absolute errors of the reactivity for three cases: UOX, UOX-Gd and MOX. Before and after *a priori* elimination, the distributions are similar while the number of coefficients used in the Tucker decomposition for cross-sections are reduced by around 40%-60% for UOX, 20%-40% for UOX-Gd and 60% for MOX (see table 5.3).

5.4 Conclusion and perspective

Sparsity methods for coefficients in the Tucker decomposition are applied to the reconstruction of neutron cross-sections. Two methods are employed: *a posteriori* and *a priori*. These methods allow us to reduce (around 50%) the number of coefficients stored in neutron libraries while the quality of the reconstruction was not changed a lot.

In general, *a posteriori* method is easier than *a priori* one and often has a better ratio reduction/accuracy. Nevertheless, *a posteriori* sparsity does not allow us to reduce the number of expensive APOLLO2 calculations needed to determine the coefficients in the Tucker decomposition.

In our present applications, *a priori* method does not reduce the number of APOLLO2 calculations either, this is mainly due to the acquisition of the neutron libraries through GAB.

We have for the moment resorted to a least squares method for solving an overdetermined system to determine the coefficients kept after *a priori* elimination. By this way, we show that the proposed criteria can predict less important coefficients in the Tucker decomposition without calculating them: accuracy is similar before and after a significantly reduction of the number of coefficients. This point also needs to be confirmed in the case where we use less APOLLO2 calculations, by solving a smaller square system (the number of equation is equal to the number of coefficients kept after *a priori* sparsity).

The results obtained from our applications demonstrate that we can reduce data in the Tucker decomposition by using sparsity methods. In future, an optimal way to define the points used after *a priori* elimination for one(all) cross-section(s) should help us to break the "curse of dimensionality" problem in the Tucker decomposition. These improvement will be investigated in our next works.

Conclusion and perspectives

Conclusion and perspectives

The goal of this work was to find a new model for the reconstruction of neutron cross-sections in the new core code COCAGNE developed at EDF-R&D. To deal with this problem, we have proposed the Tucker decomposition, a model based on the Tucker format and an extension of the Karhunen-Loève decomposition. Using this model, each cross-section is approached by a limited linear combination of tensor products of one-variate functions which are called "tensor directional basis functions" in our work. These functions are constructed by the extension of the Karhunen-Loève decomposition.

We have tested our model on two calculation domains: a standard domain $\mathcal{D}$ where parameter values are close to the nominal values (the default values on which the reactor operates normally) and an extended domain $\mathcal{D}'$ where $\mathcal{D}'$ is larger than $\mathcal{D}$ ($\mathcal{D}' \supsetneq \mathcal{D}$, which could correspond to an incident situation).

With the results obtained through benchmarks on the two calculation domains, we have demonstrated that the Tucker decomposition model achieves the purposes required by EDF. In the offline step, our new model significantly reduces the number of APOLLO2 calculations (from 15% to 50%) and the storage size of the neutron libraries (from 1.5 times to 20 times, depending on the cross-section type), compared to the multilinear interpolation model (currently used in COCAGNE). We have not only achieved these reductions in the offline step, but we have also obtained a better accuracy for reconstructed cross-sections in the online step.

In a reactor core simulation, high accuracy for the reconstructed cross-sections could not lead to high accuracy for the resolution of the neutron transport equation since cross-sections are merely inputs of this equation. We therefore need to measure the final accuracy by using outputs of the flux solver which is used to solve the neutron transport equation. One of those outputs is the reactivity.

In our work, the reactor core is simplified and supposed to be a homogeneous infinite medium where the two-group diffusion equation is solved. This simplification allows us to analytically compute the reactivity and thus to measure the accuracy the reconstruction method in terms of reactivity. In general, the maximal absolute error for this reactivity is about 100 pcm to 600 pcm (depending on the benchmark), better than the reactivity error provided by the multilinear interpolation. An

important observation is that there is no accuracy deterioration when we move from the standard to the extended domain using the Tucker decomposition. This is not the case for many other reconstruction methods, such as: methods using perturbations around the nominal point, the multilinear interpolation method, ...

Since the Tucker decomposition is based on tensor directional basis functions, we can analyze the results obtained as a function of each parameter. Such analysis is employed to improve the accuracy of our approach. We proposed to use some statistical techniques to determine the major factors that cause bad accuracy zones. We have shown how we can improve results by adding more basis functions to some directions and/or discretizing with more points on some parameter value zones. This efficient method allows us to locally improve the accuracy.

Using the Tucker decomposition, in offline steps, we can further reduce the storage size of neutron libraries by sparse representation techniques applied to the coefficients in the Tucker decomposition. We employed two methods: *a posteriori* and *a priori*, reducing (around 50%) the number of coefficients stored in neutron libraries with no significant loss of accuracy, which is still better than the multilinear interpolation.

In the future, our method should be implemented in COCAGNE in order to couple it with the flux solver and verify its final accuracy. We should also measure the time used to evaluate cross-sections, and compare it with the multilinear one. On the other hand, some points should be considered in order to improve the results of this method:

- Currently, in order to ensure the accuracy of our approach on some axes (e.g. burnup), we subdivide it, leading to piece-wise polynomials basis functions that are of class C^0 but not C^1. The Tucker decomposition is thus not class C^1. Because this C^1 property is required to calculate some coefficients in the core simulation, this problem therefore needs to be solved. One can notice that the same problem occurs with the multilinear interpolation.
- In our work, we use quadrature rules based on a polynomial basis to solve the integral equation issued from the Karhunen-Loève decomposition. Using the polynomial basis has forced us to divide some axes into sub-intervals in order to capture the variations of cross-sections. To avoid this sub-division step, which requires physical knowledges about the parameters, other methods should be investigated, for example: the change of integral variables, the change of integral weights at variations zones, or the change the representation by non polynomial bases. This would also be useful to ensure the C^1-basis functions for the Tucker decomposition.
- In the determination of tensor directional basis functions for each direction, the Karhunen-Loève decomposition is used via a matricization process (a condensation of a parameter group into two parameter sub-groups). For the moment, in the offline steps, these processes are performed on tensorized grids (called Tucker grids in our work), each grid having $n2^{d-1}$ points

(where $n \sim 10, d \sim 5$). Randomized points in each grid should be tested, which would allow us to get more accuracy for the resolution of integral equations while using less points for each grid (also meaning less APOLLO2 calculations).

- For a final desired accuracy of the Tucker decomposition, a criterion to predict the best truncation in the Karhunen-Loève decomposition (in each direction) needs to be studied. It could be based on a posteriori error of the truncation since this error is determined by the sum of the remaining and smallest eigenvalues of the Karhunen-Loève decomposition. Such a criterion would allow us to identify when a tensor directional basis function is no more a good approximation of the corresponding eigenfunction in the Karhunen-Loève decomposition. This would also help us to determine the optimal number of tensor directional basis functions that needs to be used in the Tucker decomposition for given discretizations of the integral equations.
- For the *a priori* sparse representation method, we need to solve a new system of linear equations to determine the coefficients obtained after this *a priori* sparsity. Such a new system depends on new points in the parameters-phase space. We therefore need to find the best way to construct these new points. This could get us the best ratio [accuracy for each cross-section]/[number of APOLLO2 calculations used for all cross-sections] and it would help us to break the "curse of dimensionality" problem in the Tucker decomposition.

Bibliography

[Abramowitz and Stegun, 1964] Abramowitz, M. and Stegun, I. A. (1964). *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*, volume 55. Courier Corporation.

[Antonio and Wolfgang, 2012] Antonio, F. and Wolfgang, H. (2012). On minimal subspaces in tensor representations. *Foundations of Computational Mathematics*, 12:765–803.

[Areva company, 2016] Areva company (2016). Arcadia code system. http://de.areva.com/EN/customer-2821/arcadia-core-design-core-monitoring-and-safety-analysis.html. Accessed: 2016-10-15.

[Atkinson, 1997] Atkinson, K. E. (1997). *The numerical solution of integral equations of the second kind*, volume 4. Cambridge university press.

[Bellman, 1957] Bellman, R. (1957). *Dynamic Programming*. Rand Corporation research study. Princeton University Press.

[Bellman, 1961] Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour*. Princeton Legacy Library. Princeton University Press.

[Berkooz et al., 1993] Berkooz, G., Holmes, P., and Lumley, J. L. (1993). The proper orthogonal decomposition in the analysis of turbulent flows. *Annual review of fluid mechanics*, 25(1):539–575.

[Boltzmann, 1970] Boltzmann, L. (1970). *Weitere studien über das wärmegleichgewicht unter gasmolekülen*. Springer.

[Botes and Bokov, 2011] Botes, D. and Bokov, P. M. (2011). Hierarchical, multilinear representation of few-group cross sections on sparse grids. In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2011), Rio de Janeiro, Brazil*.

[Boyd, 2001] Boyd, J. P. (2001). *Chebyshev and Fourier spectral methods*. Courier Dover Publications.

[Bungartz and Griebel, 2004] Bungartz, H.-J. and Griebel, M. (2004). Sparse grids. *Acta numerica*, 13:147–269.

[Carroll and Chang, 1970] Carroll, J. and Chang, J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of Eckart-Young decomposition. *Psychometrika*, 35(3):283–319.

[Clenshaw and Curtis, 1960] Clenshaw, C. W. and Curtis, A. R. (1960). A method for numerical integration on an automatic computer. *Numerische Mathematik*, 2(1):197–205.

[Comon et al., 2009] Comon, P., ten Berge, J. M., De Lathauwer, L., and Castaing, J. (2009). Generic and typical ranks of multi-way arrays. *Linear Algebra and its Applications*, 430(11-12):2997–3007.

[Danniël and Pavel, 2014] Danniël, B. and Pavel, M. B. (February 2014). Polynomial interpolation of few-group neutron cross sections on sparse grids. *Annals of Nuclear Energy*, 64:156–168.

[De Lathauwer et al., 2000] De Lathauwer, L., De Moor, B., and Vandewalle, J. (2000). A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4):1253–1278.

[de Silva and Lim, 2008] de Silva, V. and Lim, L.-H. (2008). Tensor rank and the ill-posedness of the best low-rank approximation problem. *SIAM J. Matrix Anal. Appl.*, 30(3):1084–1127.

[Dellnitz et al., 1994] Dellnitz, M., Golubitsky, M., and Nicol, M. (1994). *Symmetry of attractors and the Karhunen-Loeve decomposition*. Springer.

[Dufek, 2011] Dufek, J. (2011). Building the nodal nuclear data dependences in a many-dimensional state-variable space. *Annals of Nuclear Energy*, 38(7):1569–1577.

[Eckart and Young, 1936] Eckart, C. and Young, G. (September 1936). The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218.

[Edenius et al., 1986] Edenius, M., Smith, K., Ver Planck, D., Ahlin, A., and Jernberg, P. (1986). New data and methods for CASMO and SIMULATE. Technical report.

[Fujita et al., 2014] Fujita, T., Endo, T., and Yamamoto, A. (2014). A macroscopic cross-section model for BWR pin-by-pin core analysis. *Journal of Nuclear Science and Technology*, 51(3):282–304.

[Grasedyck, 2010] Grasedyck, L. (2010). *Polynomial approximation in hierarchical Tucker format by vector-tensorization*. Inst. für Geometrie und Praktische Mathematik.

[Hackbusch, 2012] Hackbusch, W. (2012). *Tensor spaces and numerical tensor calculus*. Springer.

[Hackbusch and Kühn, 2009] Hackbusch, W. and Kühn, S. (2009). A new scheme for the tensor representation. *Journal of Fourier Analysis and Applications*, 15(5):706–722.

[Halmos et al., 1957] Halmos, P. R., Halmos, P. R., Halmos, P. R., Halmos, P. R., and Mathematician, H. (1957). *Introduction to Hilbert space and the theory of spectral multiplicity*. Chelsea New York.

[Harris, 2004] Harris, S. (2004). *An introduction to the theory of the Boltzmann equation*. Courier Corporation.

[Harshman, 1970] Harshman, R. A. (1970). Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis.

[Hébert, 2006] Hébert, A. (2006). Towards DRAGON version4. In *Workshop at International Mtg. on the Physics of Fuel Cycles and Advanced Nuclear Systems: Advances in Nuclear Analysis and Simulation PHYSOR*.

[Hébert, 2010] Hébert, A. (2010). Multigroup neutron transport and diffusion computations. In *Handbook of Nuclear Engineering*, pages 751–911. Springer.

[Hobson et al., 2013] Hobson, G., Bolloni, H.-W., Breith, K.-A., van Geemert, R., Haase, H., and Hartmann, B. (2013). ARTEMISTM Core Simulator: Latest developments. In *International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013 (SNA + MC 2013)*.

[Hobson et al., 2008] Hobson, G., Merk, S., Bolloni, H.-W., Breith, K.-A., Curca-Tivig, F., Van Geemert, R., Heinecke, J., Hartmann, B., Porsch, D., Tiles, V., et al. (2008). ARTEMIS: The core simulator of AREVA NP's next generation coupled neutronics/thermal-hydraulics code system ARCADIA. *PHYSOR 2008 Proceedings*, pages 14–19.

[Hotelling, 1933] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441 and 498–520.

[Karhunen, 1946] Karhunen, K. (1946). *Zur Spektraltheorie stochastischer Prozesse*. Annales Academiae scientiarum Fennicae. Series A. 1, Mathematica-physica.

[Kressner and Tobler, 2012] Kressner, D. and Tobler, C. (2012). htucker–a matlab toolbox for tensors in hierarchical Tucker format. *MATHICSE, EPF Lausanne*.

[Lawson and Hanson, 1995] Lawson, C. L. and Hanson, R. J. (1995). *Solving least squares problems*, volume 15. SIAM.

[LeVeque, 2007] LeVeque, R. J. (2007). *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*, volume 98. Siam.

[Lewis, 2010] Lewis, E. (2010). Second-order neutron transport methods. In *Nuclear Computational Science*, pages 85–115. Springer.

[Lewis and Miller, 1984] Lewis, E. and Miller, W. (1984). *Computational methods of neutron transport*. John Wiley and Sons, Inc., New York.

[LLC, 2016] LLC, M. (2016). MS Windows NT kernel description. http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm. Accessed: 2016-10-15.

[Loève, 1955] Loève, M. (1955). *Probability Theory: Foundations, Random Sequences*. University series in higher mathematics. Van Nostrand.

[Luu et al., 2015] Luu, T. H., Maday, Y., Guillo, M., and Guérin, P. (2015). Reconstruction of neutronic macroscopic cross-sections using Tucker decomposition. In *International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method, Nashville, Tennessee, USA*.

[Luu et al., 2016a] Luu, T. H., Maday, Y., Guillo, M., and Guérin, P. (2016a). Benchmarking of Tucker decomposition method for reconstruction of neutronic macroscopic cross-sections. *Nuclear Science and Engineering (submitted)*.

[Luu et al., 2016b] Luu, T. H., Maday, Y., Guillo, M., and Guérin, P. (2016b). A new method for reconstruction of cross-sections using Tucker decomposition. *Journal of Computational Physics (submitted)*.

[Luu et al., 2016c] Luu, T. H., Maday, Y., Guillo, M., and Guérin, P. (2016c). Sparse representation in Tucker decomposition applied to the reconstruction of neutronic cross-sections. *(submitted)*.

[Lyons, 1982] Lyons, S. W. (1982). Empirical orthogonal function analysis of Hawaiian rainfall. *Journal of Applied Meteorology*, 21(11):1713–1729.

[Maday et al., 2013] Maday, Y., Mula, O., Turinici, G., et al. (2013). A priori convergence of the generalized empirical interpolation method. In *10th international conference on Sampling Theory and Applications (SampTA 2013)*, pages 168–171.

[Maday et al., 2007] Maday, Y., Nguyen, N. C., Patera, A. T., and Pau, G. S. (2007). A general, multipurpose interpolation procedure: the magic points.

[Madenci and Guven, 2015] Madenci, E. and Guven, I. (2015). *The finite element method and applications in engineering using ANSYS*. Springer.

[Marguet, 2013] Marguet, S. (2013). *La physique des réacteurs nucléaires, 2ème édition*. Tec& Doc, Lavoisier.

[Mayhue et al., 2006] Mayhue, L., Zhang, B., Sato, D., and Jung, D. (2006). PWR core modeling using the NEXUS once-through cross section model. In *PHYSOR-2006, ANS Topical Meeting on Reactor Physics, Vancouver, Canada*.

[McClarren, 2010] McClarren, R. G. (2010). Theoretical aspects of the simplified Pn equations. *Transport Theory and Statistical Physics*, 39(2-4):73–109.

[Miller, 2006] Miller, S. J. (2006). The method of least squares. *Mathematics Department Brown University*, pages 1–7.

[Müller et al., 2007] Müller, E., Mayhue, L., and Zhang, B. (2007). Reactor physics methods development at Westinghouse. In *Proc. Int. Conf. Nuclear Energy for New Europe 2007*.

[Nouy, 2016] Nouy, A. (2016). Low-rank tensor methods for model order reduction. pages 1–26.

[Oseledets, 2011] Oseledets, I. V. (2011). Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317.

[Pearson, 1901] Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

[Plagne and Ponçot, 2005] Plagne, L. and Ponçot, A. (2005). Generic programming for deterministic neutron transport codes. In *Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and biological Applications*.

[Polytechnique Montréal, 2016] Polytechnique Montréal (2016). DRAGON code. http://www.polymtl.ca/nucleaire/en/logiciels/index.php. Accessed: 2016-10-15.

[Pomraning, 1993] Pomraning, G. (1993). Asymptotic and variational derivations of the simplified Pn equations. *Annals of Nuclear Energy*, 20(9):623–637.

[Reuss, 2003] Reuss, P. (2003). *Précis de neutronique*. Collection Génie atomique. EDP Sciences.

[Rhodes et al., 2006] Rhodes, J., Smith, K., and Lee, D. (2006). CASMO-5 development and applications. In *Proc. ANS Topical Meeting on Reactor Physics (PHYSOR-2006)*, pages 10–14.

[Sanchez et al., 1988] Sanchez, R., Mondot, J., Cossic, A., Zmijarevic, I., et al. (1988). APOLLO II: A user-oriented, portable, modular code for multigroup transport assembly calculations. *Nuclear Science and Engineering*, 100(3):352–362.

[Sanchez et al., 2010] Sanchez, R., Zmijarevic, I., Coste-Delclaux, M., Masiello, E., Santandrea, S., Martinolli, E., Villate, L., Schwartz, N., and Guler, N. (2010). APOLLO2 year 2010. *Nuclear Engineering and Technology*, 42:474 – 499.

[Schmidt, 1989] Schmidt, E. (1989). Zur theorie der linearen und nichtlinearen integralgleichungen. In *Integralgleichungen und Gleichungen mit unendlich vielen Unbekannten*, pages 188–233. Springer.

[Šimša, 1992] Šimša, J. (1992). The best L2-approximation by finite sums of functions with separable variables. *Aequationes mathematicae*, 43(2-3):248–263.

[Stålek and Demazière, 2008] Stålek, M. and Demazière, C. (2008). Development and validation of a cross-section interface for PARCS. *Annals of Nuclear Energy*, 35(12):2397–2409.

[Studsvik, 2016] Studsvik (2016). SIMULATE code. http://www.studsvik.com/sv/Verksamhetsomraden/Bransle--och-materialteknik/Programvara-for-bransleoptimering/In-Core-Fuel-Management/SIMULATE5/. Accessed: 2016-10-15.

[Trefethen, 2008] Trefethen, L. N. (2008). Is Gauss quadrature better than Clenshaw-Curtis? *SIAM review*, 50(1):67–87.

[Tucker, 1966] Tucker, L. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311.

[Turski et al., 1997] Turski, R. B., Morris, E. E., Taiwo, T. A., and Cahalan, J. E. (1997). Macroscopic cross section generation and application for coupled spatial kinetics and thermal hydraulics analysis with SAS-DIF3DK.

[Watson and Ivanov, 2002] Watson, J. K. and Ivanov, K. N. (2002). Improved cross-section modeling methodology for coupled three-dimensional transient simulations. *Annals of Nuclear Energy*, 29(8):937–966.

[Weisstein, 2002] Weisstein, E. W. (2002). Fredholm integral equation of the second kind.

[Williams, 1990] Williams, G. (1990). Overdetermined systems of linear equations. *The American Mathematical Monthly*, 97(6):511–513.

[Zabreyko et al., 2013] Zabreyko, P., Koshelev, A., Krasnosel'skii, M., Mikhlin, S., Rakovshchik, L., and Stet'senko, V. Y. (2013). *Integral equations: A reference text*. Springer.

List of Figures

List of Tables