Fast hierarchical algorithms for the low-rank approximation of matrices, with applications to materials physics, geostatistics and data analysis

Pierre Blanchard

université de BORDEAUX

THÈSE

PRÉSENTÉ À

L'UNIVERSITÉ DE BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Pierre Blanchard**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : MATHÉMATIQUES APPLIQUÉES ET CALCUL SCIENTIFIQUE

Algorithmes hiérarchiques rapides pour l'approximation de rang faible des matrices, applications à la physique des matériaux, la géostatistique et l'analyse de données.

Fast hierarchical algorithms for the low-rank approximation of matrices, with applications to materials physics, geostatistics and data analysis.

Soutenue le : 16 Février 2017

Après avis des rapporteurs :

George Biros	Professeur - The University of Texas at Austin
Steffen Börm	Professeur - Christian-Albrechts-Universität zu Kiel

Devant la commission d'examen composée de :

George Biros	Professeur - The University of Texas at Austin	Rapporteur
Steffen Börm	Professeur - Christian-Albrechts-Universität zu Kiel	Rapporteur
Olivier Coulaud	Directeur de recherche - Inria Bordeaux - Sud-Ouest	Directeur de Thèse
Eric Darve	Professeur associé - Stanford University	Co-Directeur de Thèse
Laurent Dupuy	Docteur/Ingénieur de recherche - CEA Saclay	Examinateur
Alain Franc	Directeur de recherche - INRA	Examinateur/Président

– 2017 –

To Manon.

Acknowledgments

I would like to acknowledge the HiePACS team for providing such a peaceful and inspiring atmosphere of research during my stay at Inria Bordeaux. A huge thanks to the ScalFMM group, in particular Bérenger Bramas, who has always been understanding and has always made himself available for nice enriching discussions. I am very grateful to my team mates as well, namely Jean-Marie, Arnaud, Maria, Yuval, Grégoire, Abdou, Manu, with whom I shared wonderful times. Thanks Thomas Traub and Martin Schanz for there support before and during the thesis. My experience at TU Graz definitely had a huge impact on this work. Thank you Franz, Michaël, Matthias and Bernhard as well!

I wish to thank my supervisor Olivier Coulaud for allowing me to address such a wide variety of scientific issues. Thank you Olivier for being always so supportive and patient. Thank you Eric Darve for always suggesting novel ideas and for raising my knowledge to randomized numerical linear algebra and many other great topics in scientific computing. Thank you for hosting me at Stanford University and allowing to meet great people there.

Thank you Laurent Dupuy for introducing me to the world of dislocations and the beautiful community of multiscale material modeling. Finally, many thanks to Alain Franc for the wonderful exchanges we had around randomized algorithms, geometry and biodiversity. Thank you especially for trusting me in the amazing project we started building.

To everyone thank you for those small attentions that made this research experience so delightful, I wish success to all of you!

Abstract

Advanced techniques for the low-rank approximation of matrices are crucial dimension reduction tools in many domains of modern scientific computing. Hierarchical approaches like $\mathcal{H}^2$-matrices, in particular the Fast Multipole Method (FMM), benefit from the block low-rank structure of certain matrices to reduce the cost of computing n-body problems to $O(n)$ operations instead of $O(n^2)$. In order to better deal with kernels of various kinds, kernel independent FMM formulations have recently arisen such as polynomial interpolation based FMM. However, they are hardly tractable to high dimensional tensorial kernels, therefore we designed a new highly efficient interpolation based FMM, called the Uniform FMM, and implemented it in the parallel library ScalFMM. The method relies on an equispaced interpolation grid and the Fast Fourier Transform (FFT). Performance and accuracy were compared with the Chebyshev interpolation based FMM. Numerical experiments on artificial benchmarks showed that the loss of accuracy induced by the interpolation scheme was largely compensated by the FFT optimization. First of all, we extended both interpolation based FMM to the computation of the isotropic elastic fields involved in Dislocation Dynamics (DD) simulations. Second of all, we used our new FMM algorithm to accelerate a rank-r Randomized SVD and thus efficiently generate multivariate Gaussian random variables on large heterogeneous grids in $O(n)$ operations. Finally, we designed a new efficient dimensionality reduction algorithm based on dense random projection in order to investigate new ways of characterizing the biodiversity, namely from a geometric point of view.

Keywords: fast multipole method, fast Fourier transform, random projection, dislocation dynamics, covariance matrix, multidimensional scaling.

Résumé

Les techniques avancées pour l'approximation de rang faible des matrices sont des outils de réduction de dimension fondamentaux pour un grand nombre de domaines du calcul scientifique. Les approches hiérarchiques comme les matrices $\mathcal{H}^2$, en particulier la méthode multipôle rapide (FMM), bénéficient de la structure de rang faible par bloc de certaines matrices pour réduire le coût de calcul de problèmes d'interactions à n-corps en $O(n)$ opérations au lieu de $O(n^2)$. Afin de mieux traiter des noyaux d'interaction complexes de plusieurs natures, des formulations FMM dites "kernel-independent" ont récemment vu le jour, telles que les FMM basées sur l'interpolation polynomiale. Cependant elles deviennent très coûteuses pour les noyaux tensoriels à fortes dimensions, c'est pourquoi nous avons développé une nouvelle formulation FMM efficace basée sur l'interpolation polynomiale, appelée Uniform FMM. Cette méthode a été implémentée dans la bibliothèque parallèle ScalFMM et repose sur une grille d'interpolation régulière et la transformée de Fourier rapide (FFT). Ses performances et sa précision ont été comparées à celles de la FMM par interpolation de Chebyshev. Des simulations numériques sur des cas tests artificiels ont montré que la perte de précision induite par le schéma d'interpolation était largement compensées par le gain de performance apporté par la FFT. Dans un premier temps, nous avons étendu les FMM basées sur grille de Chebyshev et sur grille régulière au calcul des champs élastiques isotropes mis en jeu dans des simulations de Dynamique des Dislocations (DD). Dans un second temps, nous avons utilisé notre nouvelle FMM pour accélérer une factorisation SVD de rang r par projection aléatoire et ainsi permettre de générer efficacement des champs Gaussiens aléatoires sur de grandes grilles hétérogènes. Pour finir, nous avons développé un algorithme de réduction de dimension basé sur la projection aléatoire dense afin d'étudier de nouvelles façons de caractériser la biodiversité, à savoir d'un point de vue géométrique.

Mots clés: méthode multipole rapide, transformé de Fourier rapide, projection aléatoire, dynamique des dislocations, matrices de covariance, positionnement multidimensionnel.

Résumé étendu

Introduction Les techniques avancées pour l'approximation de rang faible des matrices sont des outils d'algèbre linéaire numérique cruciaux pour la réduction de dimension. Elles interviennent dans un nombre important de domaines du calcul scientifique, dont certains sont abordés dans cette thèse. Des variantes hiérarchiques comme les matrices dites $\mathcal{H}^2$, en particulier la méthode multipôle rapide (FMM), peuvent en plus bénéficier de la structure de rang faible par bloc de certains opérateurs matriciels. Pendant les deux dernières décennies, la FMM a été utilisée de manière intensive dans de nombreuses applications allant de la simulation de problèmes à n-corps à la résolution d'équations aux dérivées partielles (EDP). De manière générale, la FMM est une méthode hiérarchique d'algèbre linéaire numérique permettant d'accélérer les calculs mettant en oeuvre des matrices noyaux (ou matrices d'interactions), comme défini ci-après

$$\mathbf{K} = \{k(\|\mathbf{x}_i - \mathbf{x}_j\|_2)\}_{i,j=1\ldots n}$$

ou $\{\mathbf{x}_i\}_{i=1\ldots n}$ est une distribution spatiale de points et k est une fonction noyaux (fonction de 2 variables d'espace) suffisamment régulière mais pouvant présenter une singularité à l'origine. La FMM permet de réduire le coût de calcul des problèmes d'interactions à n-corps ou d'application de produits matrice-vecteur à $\mathcal{O}(n)$ opérations au lieu de $\mathcal{O}(n^2)$. D'autre part, de plus en plus d'applications nécessitent de mettre en oeuvre des noyaux d'interaction très complexes, en particulier de natures différentes et de dimensions différentes (potentiellement grandes). Afin de gérer ce vaste ensemble de fonctions, des formulations FMM dites indépendantes du noyaux (ou *kernel-independent*) ont récemment vu le jour. Par exemple, les formulations multipôles rapides basées sur l'interpolation polynomiale reposent sur l'interpolation du noyau d'interaction en $p+1$ points d'une grille notée $\{\bar{\mathbf{x}}_a\}_{a=1\ldots p}$ par un polynôme S d'ordre p. Cette technique revient à approximer le noyau d'interaction par la formule suivante

$$\mathbf{k}(\mathbf{x}, \mathbf{y}) \approx \sum_{a \leq p} S(\mathbf{x}, \bar{\mathbf{x}}_a) \sum_{b \leq p} \mathbf{k}(\bar{\mathbf{x}}_a, \bar{\mathbf{y}}_b) S(\bar{\mathbf{y}}_b, \mathbf{y})$$

Cependant, de telles approches s'étendent difficilement à des noyaux d'interactions tensoriels lorsque leur dimension est élevée, en particulier parce que l'approximation de rang faible est sous optimale. Nous nous restreignons dans cette thèse au cas des noyaux non-oscillants, mais mentionnons dans ce manuscrit les récentes avancées qui permettent d'étendre notre approche aux noyaux oscillants. L'avantage de cette approche par rapport aux autres méthodes dites *kernel-independent* est qu'elle ne repose sur aucune hypothèse concernant la nature du noyau, en particulier son lien éventuel avec une EDP. D'autre part, son implémentation est relativement simple car elle repose sur des outils mathématiques relativement basiques et peut bénéficier des outils optimisés pour l'algèbre linéaire numérique dense tels que *blas* et *lapack*.

Fast interpolation based FMM Dans cette thèse, nous avons développé une nouvelle formulation FMM hautement performante basée sur l'interpolation polynomiale, appelée Uniform FMM ou *ufmm*, qui permet de réduire le coût de calcul de l'étape la plus coûteuse de l'algorithme FMM, à savoir l'étape de transfert M2L. Cette technique repose simplement sur une reformulation de l'opérateur M2L sans utiliser d'hypothèse supplémentaire sur le

noyau et sans introduire une nouvelle approximation matricielle. Cette approche a été implémentée dans la bibliothèque ScalFMM*, où elle bénéficie des dernières évolutions en terme de vectorisation, de paradigmes pour le calcul parallèle et de moteur d'exécutions. Notre méthode repose sur une grille d'interpolation régulière, ce qui implique une structure dites Toeplitz à l'opérateur M2L et permet de réduire son coût de stockage. De plus, la conversion de l'étape M2L dans le domaine de Fourier par transformée de Fourier rapide (FFT) permet de réduire significativement la complexité de l'étape de transfert M2L. Les performances et la précision de l'approche ont été comparé aux méthodes existantes telles que la FMM basée sur l'interpolation de Chebyshev et ses variantes optimisées par recompression algébrique des opérateurs. Des simulations numériques sur des cas tests artificiels ont montré que la perte de précision introduite par le schéma d'interpolation était largement compensé par l'accélération FFT. D'autre part, l'empreinte mémoire de l'étape M2L a été réduite de manière significative au prix de stocker des développements multipolaires de taille plus importante.

Dynamique des dislocations Dans la première partie de cette thèse, nous avons étendue les formulations FMM par interpolation à l'évaluation des champs isotropiques élastiques mis en oeuvre dans les simulations de Dynamique des Dislocations (DD). Le modèle DD repose sur un représentation intégrale aux frontières du champs de force élastique créé par un ensemble de défauts linéiques en interactions élastiques, ces défauts sont aussi connus sous le nom de *dislocations*. La simulation d'ensemble massifs de dislocations, par exemple pour des discrétisations allant jusqu'à $\mathcal{O}(10^7)$ segments, permet de comprendre de manière plus précise les lois de comportement plastique des matériaux cristallins irradiés. Entreprendre de telles simulations a nécessité d'adapter les routines d'interpolation de la bibliothèque ScalFMM aux éléments intégraux et à la nature tensorielle des interactions propres aux dislocations. En utilisant ScalFMM comme moteur d'exécution de la FMM, nous avons étendu les capacités du code DD parallèle OptiDis† en fournissant une évaluation multipôle rapide performante des champs élastiques isotropiques. Notre méthode a permis de réaliser des simulations DD avec $n = \mathcal{O}(10^7)$ segments en prenant en compte les contributions de champs lointain de la contrainte, de la force et de l'énergie élastique, tout en maintenant une précision de l'ordre de 10^{-4} et minimisant le coût de ce champs lointain. D'autre part l'utilisation d'une approche *kernel-independent* a permis de facilement mettre en oeuvre des formulations optimisées du champs d'interaction pour son évaluation multipôle rapide, et facilitera la mise en oeuvre de noyaux anisotropiques. D'autre part, l'interpolation polynomiale a permis de diminuer la dimensionnalité du noyaux et d'évaluer efficacement les opérateurs intégraux. Enfin, nos travaux ont permis de mener à bien des simulations clés dans le domaine des dislocations dans un temps raisonnable et avec la précision attendue, en particulier nous avons pu simuler les mécanismes associés à l'apparition de *bandes claires* dans les alliages de Zirconium (Figure 1).

Matrices de covariance Dans la seconde partie de cette thèse, nous avons mis en oeuvre notre nouvel algorithme FMM afin de générer de manière efficace des variables aléatoires Gaussiennes multidimensionnelles sur des grilles arbitraires et en particulier très hétérogènes. L'approche standard consiste à appliquer une racine carrée de la matrice de covariance, calculée à partir de la donnée d'une fonction de corrélation et d'une grille de n points, à

*La bibliothèque *ScalFMM* et sa documentation complète sont accessibles gratuitement à l'adresse suivante: https://gitlab.inria.fr/solverstack/ScalFMM.

†Une présentation de la bibliothèque *Optidis* ainsi que des vidéos de démonstration sont disponibles en ligne à l'adresse suivante: http://optidis.gforge.inria.fr/.

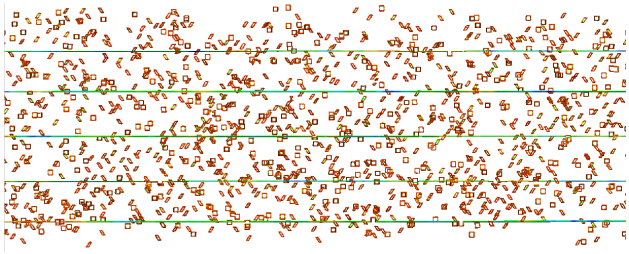

Figure 1: Apparitions de bandes claires dans les alliages de Zirconium (Zr) simulées avec *OptiDis*.

un bruit blanc généré sur cette grille. Cette méthode est particulièrement coûteuse, en effet $\mathcal{O}(n^3)$ opérations sont nécessaire pour calculer de manière exacte une racine carrée de matrice par une factorisation de Cholesky ou SVD, de plus il est nécessaire d'assembler la matrice pour en extraire sa racine. L'approche que nous proposons repose sur l'accélération du calcul de la racine en combinant deux techniques avancées d'approximation de rang faible:

- une SVD de rang-r approchée par projection aléatoire, aussi connu sous le nom de *randomized SVD*,
- accélérée par des multiplications de matrices approchées grâce à la FMM.

Le coût de la *randomized SVD* étant gouverné par $2r$ produits matrice-vecteur, la FMM permet de réduire le coût de pré-calcul de la racine carré de rang faible de $\mathcal{O}(rn^2)$ à $\mathcal{O}(r^2n)$ opérations sans qu'il n'y ait besoin de stocker la matrice de covariance. Premièrement, nous avons optimisé notre implémentation *ufmm* afin de mieux gérer un nombre élevé de vecteur d'entrée, à savoir $r = \mathcal{O}(10^3)$. Dans un second temps, étant donné la régularité de certaines fonctions de corrélation utilisées dans ce type d'applications, *e.g.*, la corrélation Gaussienne, nous avons développé un algorithme d'approximation de rang faible par blocs basé sur l'interpolation polynomiale, en changeant simplement le critère d'admissibilité de la version standard de la FMM par interpolation. Notre approche hiérarchique a permis de générer des champs aléatoires Gaussiens sur des grilles allant jusqu'à $n = \mathcal{O}(10^6)$ points en quelques secondes. Pour finir, nous avons illustré que pour des distributions de points très hétérogènes, le caractère hiérarchique de notre approche permettait d'augmenter ces performances de manière significative, en effet le remplissage de l'arbre est d'autant plus creux que la grille est hétérogène.

Positionnement multidimensionel Dans la dernière partie de la thèse, nous avons mis au point un algorithme de réduction de dimension efficace pour étudier de nouvelles façons de décrire la biodiversité, à savoir d'un point de vue géométrique. Notre approche est basée sur une technique de visualisation appelée le *positionnement multidimensionnel* ou *multidimensional scaling* (MDS). Cette technique permet de représenter un ensemble de n points dans un espace euclidien à r dimensions avec $r << n$ grâce à la seule donnée des distances deux-à-deux entre ces points. Le coût de calcul de la MDS est en général largement dominé par l'étape de factorisation de la matrice de similarité associée à la matrice de distance, en effet les méthodes standards requièrent $\mathcal{O}(n^3)$ opérations et nécessitent l'assemblage et le stockage de la matrice. La plupart des méthodes rapides pour le *positionnement multidimensionnel* sont basées sur des techniques de factorisation par sélection aléatoire de colonnes (ou échantillonnage aléatoire), qui ne s'appliquent qu'à des matrices définie-positives, à savoir la méthode de Nyström. D'autre part, une discussion sur l'influence de la présence de valeurs propres négatives sur la distorsion du nuage a également été menée.

Notre nouvelle approche met en oeuvre un algorithme de SVD basé sur la projection Gaussienne dense connu sous le nom de *randomized SVD*. La projection aléatoire a de multiples avantages sur la sélection de colonnes aléatoire dans ce domaine bien que cette dernière soit moins coûteuse en terme de calcul. Premièrement, la projection aléatoire est par essence plus proche du concept de réduction de dimension, car une illustration plus directe du le lemme de Johnson-Lindenstrauss. Ensuite, elle permet une factorisation sous la forme SVD de manière rapide, stable et précise, en fournissant des informations cruciales sur la structure de la matrice, à savoir une approximation de l'image et du rang numérique. D'autre part, la *randomized SVD* présentent des avantages significatifs en terme d'implémentation et de performance. La projection aléatoire a permis de faire tourner l'algorithme de positionnement multidimensionnel sur de grands échantillons fournis par la nouvelle génération de technologies de séquençage d'ADN dites *ngs*, et ce grâce à une implémentation C++ efficace de la *randomized SVD* (mais aussi sur la décomposition de Nyström) dans la bibliothèque *fmr*[‡].

Des méthodes de visualisations basiques ont été utilisées pour représenter les nuages de points dans un nombre relativement faible de dimensions. Nous avons pu traiter de manière fluide et précise des échantillons jusqu'à $\mathcal{O}(10^5)$ séquences (ou *reads*) provenant du Lac Léman, et ainsi illustrer deux phénomènes:

- la réunion en cluster des *reads* associés à des espèces identifiées
- et la forte concentration des *reads* dans des régions très localisées et bien séparées entre elles.

Une étude plus avancée de ce dernier phénomène devrait permettre la définition de nouveaux *Operational Taxonomic Units* (OTUs) et ainsi une caractérisation de la biodiversité sur un critère purement géométrique.

Conclusion Nous avons montré sur deux applications l'importance de développer des algorithmes efficaces qui exploitent la structure de rang faible par bloc des matrices d'interaction et prennent en compte l'hétérogénéité du domaine. Une application en physique des matériaux nous a permis de souligner la nécessité de développer des formulations peu coûteuses et générique pour traiter des interactions très complexes. Une application en géostatistique nous a permis d'illustrer les bénéfices d'une représentation hiérarchiques de la matrice de covariance sur une technique efficace pour la factorisation globale des matrices (ou plus précisément dans ce cas le calcul de racines carrés), reposant sur la projection aléatoire. Enfin, une approche plus exploratoire nous a permis d'étendre la factorisation par projection aléatoire aux matrices de similarités mises en oeuvre pour la classification de grands jeux de données en biodiversité. Cette dernière a fourni des résultats prometteurs qui doivent permettre une réduction supplémentaire du coût de calcul en exploitant plus finement la structure des matrices.

[‡]La bibliothèque *fmr* est disponible en ligne à l'adresse suivante: https://gitlab.inria.fr/piblanch/fmr.

Contents

Introduction

Introduction

Challenges in Scientific Computing Applied physicists, biologists, geoscientists, climatologists, economists and many other kind of computational scientists keep developing new models in order to simulate real-life phenomenon more accurately and at more realistic scales. However, because of their complex nature, large dimensions and extreme refinement, these models are more and more demanding in terms of computational resources. Therefore, the challenges faced by the scientific computing community for the next few decades is huge, in particular when it comes to tackling the *curse of dimensionality* induced by the use of high dimensional models or projection spaces. For instance, new approaches developed by researchers in mechanical engineering and materials sciences in order to better understand the behavior of complex structures or materials involve a growing number of parameters, which leads to very intensive computations and often requires advanced HPC solutions and, what is called in this domain, *reduced order models* [128]. On the other hand, Molecular Dynamics (MD) and Multiscale Materials Modeling (MMM) as a whole have pushed the barriers of scientific computing by addressing arbitrary high order models for the interactions at a given scale and also by coupling computations at various scales [48, 38]. A large part of scientific computing is dedicated to the resolution of problems formulated in term of n-body interactions or relying on the discretization of continuous operators on a mesh. Therefore, they usually implement relatively basic matrix computations such as multiplication, inverse, square root or more evolved matrix functions. Over the last two decades many algorithms have already been designed to reduce the complexity of these fundamentals operations. A significant part of these algorithms belong to the framework of sparse Numerical Linear Algebra (NLA) as it makes use of the sparsity of the matrix operators, but they will not be discussed in this thesis. On the other hand, the framework of dense NLA addresses the computation of more general dense matrices, which rapidly involve prohibitive resources consumptions. In particular, since the cost of applying such operators is quadratic in the problem size and factorization in low rank form as well as inversion are cubic, the challenge in this field has been the design of linear scaling algorithm. Among the notable algorithms that contributed most to the field of scientific computing, we would like to emphasize two kinds of algorithms. First of all, hierarchical algorithms such as the Fast Multipole Method (FMM) and $\mathcal{H}$ matrices as they allow for fast matrix multiplications as well as fast inversion. Then, randomized algorithms have recently brought important contributions to the fast factorization of matrices in standard forms that are oblivious of the matrix sparsity.

Hierarchical Algorithms The computation of n-body interactions problems lead to the design of the very popular FMM [59], in order to apply discrete matrix operators at a linear cost. Although the FMM terminology is highly oriented towards physical applications, it has proven its efficiency and relevance in many other areas of computational sciences. In particular, the FMM became very popular in the mechanical engineering community for the resolution of boundary value problems, namely by accelerating the matrix multiplications

involved in the iterative solvers [94, 112, 31, 88, 110]. In fact, it probably even motivated the design of its algebraic counterpart, namely the more general framework of hierarchical matrices, *a.k.a.*, $\mathcal{H}$-matrices [11, 63]. Both methods benefit from the block low rank structure of certain discrete or integral operators and rely on a data sparse representation of the matrices. Furthermore, they involve approximations that need to be set *w.r.t.* to the target application. They have not yet been integrated into generic packages such as *lapack*, because their implementation need a very special care in order to achieve optimal performance. However, the formulation and implementation of the FMM and $\mathcal{H}$-matrices have become more and more generic in order to easily adapt to any application [127, 54]. Nonetheless, FMM algorithms still need to be improved in order to cope with the growing demand in high dimensional models and extremely refined simulations.

Randomized Algorithms In the last few years, a new domain of NLA, called randomized NLA, has gained popularity thanks to its potential benefits to data analysis applications, *e.g.*, machine learning or data mining. This sudden interest for randomized algorithms lead to highly efficient methods for the Low-Rank Approximation (LRA) of matrices with application to a wide range of domains [78]. Randomized LRA are random in the sense that they involve sketching of a matrix using random procedures [123], but most importantly because they rely on error bounds that hold with probability deriving from the theory of the *concentration of measure* [3]. All randomized algorithms more or less derive from the beautiful idea, illustrated by *Johnson-Lindenstrauss lemma* [68], that for any given set of point in an Euclidean space there exists an arbitrary low-distortion embedding into a lower dimensional Euclidean space. Despite relying on a relatively involved mathematical background [65], the general idea of randomized LRA algorithms remain very intuitive and their implementation straightforward. Moreover, as they rely on very basic matrix computations [64], they can further benefit from the structure and the sparsity of the input matrix and they parallelize well. In this thesis we will focus on dense random projection based approaches [115] and there low-rank variants [75].

Scope of study First of all, our work aim at proposing new variants of the FMM that are better suited for the requirements of modern scientific applications, such as Multiscale Materials Modeling through Dislocation Dynamics (DD) simulations. Second of all, we want to highlight the interest of using the hierarchical algorithms in a larger variety of domains, in particular on geostatistical applications, that involve large grids and non-oscillatory interaction kernels of various nature. Furthermore, we want to benefit from this framework to illustrate the benefits of combining hierarchical algorithms with randomized LRA in terms of performance. In this work, we focus on developing efficient sequential algorithms that fully benefit from the recent advances in NLA and that can easily be ported on parallel applications. Although no HPC solution is actually proposed in this thesis, all our contributions are developed in commonly used and maintained open-source parallel libraries that implement the state-of-the-art parallel paradigms.

Overview

The present manuscript is organized in 3 main parts, focusing on various aspects of dense NLA for computationally demanding modern scientific applications. In particular, it describes the implementation of new efficient hierarchical algorithms, randomized algorithms or combination of both.

Preliminary Part I introduces the various algorithms studied in this thesis and the current state-of-the-art. In particular, Chapter 1 describes the main concepts and features of the FMM and interpolation based variants. Then, Chapter 2 presents the framework of our first application, namely Dislocation Dynamics. Finally, Chapter 3 introduces the fundamental concepts of randomized NLA for the low-rank approximation of dense matrices.

n-body interactions Part II gathers our various contributions to the computation of n-body interaction problems. First of all, Chapter 5 describes a highly efficient interpolation based FMM that applies to any non-oscillatory kernel. Then, Chapter 6 implements this algorithm along with other interpolation schemes for computing isotropic elastic interactions in large dislocation networks.

Covariance matrix Part III focuses on our contributions to covariance matrix computations. In Chapter 7 we describe a linear time algorithm for performing randomized LRA of matrices using our interpolation based FMM for performing fast matrix multiplications with application to the generation of Gaussian Random Fields (GRFs). Finally, Chapter 8 addresses the design of a geometric view on biodiversity by mean of random projection aided dimensionality reduction.

Part I

Scientific background & Related works

1

The Fast Multipole Method

Contents

1.1 The Fast Multipole Method

Here we recall the fundamentals of the Fast Multipole Method (FMM) and introduce generic notations. Various tools associated with the FMM are presented along with the algorithm itself as well as detailed asymptotic complexities.

1.1.1 Introduction

The simulation of n-body problems remains a crucial issue in a wide variety of scientific domains involving elastic, acoustic, electric, gravitational interactions,...These problems usually boil down to evaluating sums of the form

$$\phi(\mathbf{x}_i) = \sum_{j=1}^{n} k(\mathbf{x}_i, \mathbf{x}_j) w(\mathbf{x}_j) \tag{1.1}$$

where $\mathbf{x} \in \mathbb{R}^{n \times d}$ denote the points of a d-dimensional grid, *i.e.*, the set of n bodies/particles represented in d dimensions and $w(\mathbf{x}_i)$, respectively $\phi(\mathbf{x}_i)$, denotes the density, respectively the potential, associated to the particle i located at $\mathbf{x}_i$. Originally introduced by Greengard and Rokhlin [59], the FMM provides an efficient method for approximating $\phi(\mathbf{x}_i)$, *i.e.*, the potential created by all particles on the particle i, for all $i \in \{1, \dots, n\}$ up to a given accuracy ε_{FMM} in $\mathcal{O}(n)$ operations instead of the $\mathcal{O}(n^2)$ operations required by the naïve direct computation of the sums. From an algebraic point of view, it is equivalent to performing an approximate product between a *kernel matrix* and a vector, where the input and output vectors are respectively $\mathbf{w} = \{w(\mathbf{x}_i)\}_{i=1,\dots,n}$ and $\boldsymbol{\phi} = \{\phi(x_i)\}_{i=1,\dots,n}$ and the *kernel matrix* reads as

$$\mathbf{K} = \{k(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j \in \{1,\dots,n\}} .$$

The FMM relies on 2 main ingredients:

- the hierachical partitioning of the input grid using a binary tree (*e.g.*, quadtree in 2D, octree in 3D),
- the separation or decoupling of variables $\mathbf{x}$ and $\mathbf{y}$ in $k(\mathbf{x}, \mathbf{y})$, *e.g.*, using analytic expansions of the kernel k.

If the input kernel matrix has a block low-rank structure, then hierachical partitioning of the grid allows for efficiently building those blocks. On the other hand, analytical expansions are used for computing the smooth block-to-block interactions in low-rank form.

1.1.2 The Fast Multipole Algorithm

The Fast Multipole Algorithm (FMA) is a 2-stage algorithm that produces approximate potentials $\bar{\boldsymbol{\phi}}$ given a set of particles $\mathbf{x}$, a kernel $k(\cdot, \cdot)$ and densities $\mathbf{w}$ using a cluster tree, an admissibility criterion and a low-rank approximation scheme. Let us describe the algorithm in full details in this subsection.

Octree First of all, the FMA relies on a hierarchical partitioning of the domain using a cluster tree structure. Since only 3D grids (or spatial grids) are considered in this thesis, the name *octree* will be used to denote the tree structure. The root cluster of the tree is the smallest cube enclosing all particles. At the level L of the octree we subdivide all *parent* cells in 8 cubes of equal size to get all *child* cells at level $L+1$. This recursive partition is stopped at the *leaf* level $\bar{L}$ once a suitable criterion is reached, *e.g.*, minimum or average number of particles per leaf cell, minimum leaf cell size. We denote $\mathcal{C}_{\mathbf{x}}^{(L)}$ the level-L cell containing the particle located at $\mathbf{x}$. Its center and its width are denoted respectively $\mathbf{c}_{\mathbf{x}}^{(L)}$ and $\omega^{(L)}$. In particular, $\omega_{\mathbf{x}}^{(L)} = \omega^{(0)}/2^L$ with $\omega^{(0)}$ the width of the root cluster. Finally, the number of particles contained in $\mathcal{C}_{\mathbf{x}}^{(L)}$ is denoted $n_{\mathbf{x}}$.

Admissibility Once the domain is partitioned into clusters we need to define admissible cluster pairs $\mathcal{T}_{\mathbf{xy}}^{(L)} = (\mathcal{C}_{\mathbf{x}}^{(L)}, \mathcal{C}_{\mathbf{y}}^{(L)})$. At a given level L and for a fixed target cell $\mathcal{C}_{\mathbf{x}}^{(L)}$, the pair $\mathcal{T}_{\mathbf{xy}}^{(L)}$ is uniquely defined by its transfer vector

$$\mathbf{t}_{\mathbf{xy}}^{(L)} = \frac{\mathbf{c}_{\mathbf{x}}^{(L)} - \mathbf{c}_{\mathbf{y}}^{(L)}}{\omega^{(L)}}$$

If $\mathcal{T}_{\mathbf{xy}}^{(L)}$ is admissible, then the matrix of interactions

$$\mathbf{K}(\mathcal{T}_{\mathbf{xy}}^{(L)}) = \left\{k(\mathbf{x}_i, \mathbf{y}_j)\right\}_{i<n_{\mathbf{x}}, j<n_{\mathbf{y}}} \in \mathbb{R}^{n_{\mathbf{x}} \times n_{\mathbf{y}}}$$

can be approximated at a given precision using a low-rank method, *i.e.*,

$$\mathbf{K}(\mathcal{T}_{\mathbf{xy}}^{(L)}) \approx \mathbf{U}(\mathcal{C}_{\mathbf{x}}^{(L)})\mathbf{C}(\mathcal{T}_{\mathbf{xy}}^{(L)})\mathbf{V}^T(\mathcal{C}_{\mathbf{y}}^{(L)})$$

with $\mathbf{U} \in \mathbb{R}^{n_{\mathbf{x}} \times r}$, $\mathbf{C} \in \mathbb{R}^{r \times r}$, $\mathbf{V} \in \mathbb{R}^{n_{\mathbf{y}} \times r}$ and $r = r_{\mathbf{xy}} << n_{\mathbf{x}}, n_{\mathbf{y}}$. A pair of clusters $\mathcal{T}_{\mathbf{xy}}^{(L)}$ is said admissible if it satisfies

$$\|\mathbf{t}_{\mathbf{xy}}^{(L)}\| \geq \gamma \qquad \text{(admissibility criterion)}$$

where γ is a prescribed parameter. While the original FMA [59] enforces $\gamma = 2$, *i.e.*, the well-separation of cells or strong admissibility, general $\mathcal{H}^2$-matrices can also enforce $\gamma = 1$, *i.e.*, the adjacency of cells, *a.k.a.*, *weak admissibility*, *e.g.*, the HSS structure [125, 34]. See Bebendorf [11] for a comprehensive introduction to the concept of $\mathcal{H}$-matrix.

Nesting A particular aspect of the FMM that produces a $\mathcal{H}^2$-format is the nesting of the basis between levels. In fact, the basis U resp. V at a given level can be expressed in terms of the basis at a lower level, resp. upper level, by a relation of the following kind

$$U_{i\alpha}(\mathcal{C}_{\mathbf{x}}^{(L-1)}) = U_{i\alpha'}(\mathcal{C}_{\mathbf{x}}^{(L)})U'_{\alpha'\alpha}(\mathcal{C}_{\mathbf{x}}^{(L)})$$
$$V_{\beta i}^T(\mathcal{C}_{\mathbf{x}}^{(L-1)}) = V'^T_{\beta\beta'}(\mathcal{C}_{\mathbf{x}}^{(L)})V_{\beta' i}^T(\mathcal{C}_{\mathbf{x}}^{(L)})$$

for all $i = 1, \ldots, n_{\mathbf{x}}$, where $\mathbf{U}', \mathbf{V}' \in \mathbb{R}^{r \times r}$ denote the operators used to transfer the basis, and ultimately the expansions, from one level to another.

Neighbor and Interaction lists Let $\mathcal{C}_{\mathbf{x}}^{(L)}$ denote a target cell at level L. Cells that form admissible pairs with $\mathcal{C}_{\mathbf{x}}^{(L)}$ are also said to be in farfield interactions with $\mathcal{C}_{\mathbf{x}}^{(L)}$, while the other are in nearfield interactions. The neighbor list $\mathcal{N}(\mathcal{C}_{\mathbf{x}}^{(L)})$ is defined as the set of cells in nearfield interactions with $\mathcal{C}_{\mathbf{x}}^{(L)}$. For $\gamma = 2$, $\mathcal{N}(\mathcal{C}_{\mathbf{x}}^{(L)})$ contains at most 3^d cells for any L, *i.e.*, 27 in 3D. The interaction list $\mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})$ is defined as the set of non-empty cells $\mathcal{C}_{\mathbf{y}}^{(L)}$ in farfield interactions with $\mathcal{C}_{\mathbf{x}}^{(L)}$ such that the parent of $\mathcal{C}_{\mathbf{x}}^{(L)}$ and $\mathcal{C}_{\mathbf{y}}^{(L)}$ are in nearfield interactions.

$$\mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)}) = \left\{ \mathcal{C}_{\mathbf{y}}^{(L)} \in tree : \mathcal{T}_{\mathbf{xy}}^{(L)} \text{ is admissible and } \mathcal{C}_{\mathbf{y}}^{(L-1)} \in \mathcal{N}(\mathcal{C}_{\mathbf{x}}^{(L)}) \right\}$$

The latter condition ensures that all contributions are only computed once during the hierarchical summation scheme. For $\gamma = 2$, $\mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})$ contains at most $6^d - 3^d$ cells for any L, *i.e.*, 189 in 3D. Fig. 1.2 shows the neighbor list (colored in red) and interaction list (colored in green) on a portion of a quadtree for a given target cell $\mathcal{C}_{\mathbf{x}}^{(L)}$. On the other hand, two brother cells do not share the same interaction list, therefore there are $7^d - 3^d$ possible configurations of interacting pairs, *i.e.*, 316 in 3D. Given a fictitious target cell $\mathcal{C}_{\mathbf{x}}^{(L)}$, we denote the set of all possible interactions $\mathcal{I}_{full}(\mathcal{C}_{\mathbf{x}}^{(L)})$.

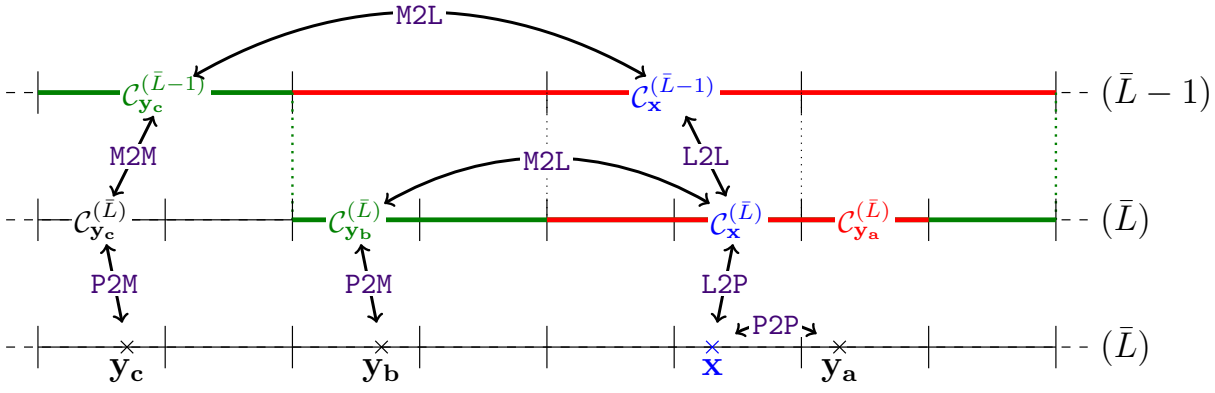

Figure 1.1: Portion of a 1D cluster tree at level $\bar{L}$ and $\bar{L}-1$. The interaction lists $\mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})$ (green) and neighbor lists $\mathcal{N}(\mathcal{C}_{\mathbf{x}}^{(L)})$ (red) of a target particle $\mathbf{x}$ is represented for $\gamma = 2$. The contribution of the source particle $\mathbf{y_a}$ is computed analytically as part of the nearfied while contributions of $\mathbf{y_b}$ and $\mathbf{y_c}$ are approximated as part of the farfield at level $\bar{L}$ and $\bar{L}-1$ respectively.

Fast Multipole Algorithm The standard FMA computes approximate particle interactions in a hierarchical fashion, *i.e.*, considering multiple levels of the tree. The multi-level fast multipole summation is presented in algorithm 1. It consists in 2 stages, namely 2 passes through the binary tree: an upward pass followed by a downward pass. Figure 1.1 illustrates the interaction and neighbor lists on the last two levels of a simple 1D tree structure. It also schematically shows how the FMA transfers nearfield and farfield contributions by mean of `X2Y` operations, where `X` and `Y` can be equal to

- `P`, when contributions are expressed on the source or target particles contained in leaf cells,
- `M`, when contributions are expressed as multipole expansions $\mathcal{M}$ in source cells,
- `L`, when contributions are expressed as local expansions $\mathcal{L}$ in target cells.

For a general d-dimensional grid, the FMA consists in successively performing the following operations:

- The Upward Pass consists in performing the following operations at each level from bottom ($L = \bar{L}$) to top ($L = 2$)
  - `P2M`: Compute all multipole expansions ($\boldsymbol{\mathcal{M}}$) at the leaf level, *i.e.*, accumulate contributions from source particles $\mathbf{y}$ to their containing leaf cell $\mathcal{C}_{\mathbf{y}}^{(\bar{L})}$ in $\boldsymbol{\mathcal{M}}(\mathcal{C}_{\mathbf{y}}^{(\bar{L})})$

$$\mathcal{M}_\beta(\mathcal{C}_{\mathbf{y}}^{(\bar{L})}) = \sum_{j=1}^{n_{\mathbf{y}}} V_{\beta j}^T(\mathcal{C}_{\mathbf{y}}^{(\bar{L})}) w(\mathbf{y}_j),\ \forall \beta \leq r \tag{1.2}$$

  - `M2M`: If $L < \bar{L}$, transfer the multipole expansions from 2^d level-$(L+1)$ child cells to their level-(L) parent cell

$$\mathcal{M}_\beta(\mathcal{C}_{\mathbf{y}}^{(L)}) = \sum_{\beta' \leq r} V_{\beta\beta'}^T(\mathcal{C}_{\mathbf{y}}^{(L)}) \mathcal{M}_{\beta'}(\mathcal{C}_{\mathbf{y}}^{(L+1)}),\ \forall \beta \leq r \tag{1.3}$$

- The Downward Pass consists in performing the following operations at each level from top ($L = 2$) to bottom ($L = \bar{L}$)
  - `M2L`: Transfer multipole expansions ($\boldsymbol{\mathcal{M}}$) from a maximum of $6^d - 3^d$ (189 in 3D) source cells $\mathcal{C}_{\mathbf{y}}^{(L)} \in \mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})$ into local expansions ($\boldsymbol{\mathcal{L}}$) of target cell $\mathcal{C}_{\mathbf{x}}^{(L)}$

$$\mathcal{L}_\alpha(\mathcal{C}_{\mathbf{x}}^{(L)}) = \sum_{\mathcal{C}_{\mathbf{y}}^{(L)} \in \mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})} \sum_{\beta \leq r} C_{\alpha\beta}(\mathcal{T}_{\mathbf{xy}}^{(L)}) \mathcal{M}_\beta(\mathcal{C}_{\mathbf{y}}^{(L)}),\ \forall \alpha \leq r \tag{1.4}$$

  - `L2L`: If $L < \bar{L}$, transfer all the local expansions from level-(L) parent cells to their 2^d level-$(L+1)$ child cells

$$\mathcal{L}_\alpha(\mathcal{C}_{\mathbf{x}}^{(L+1)}) += \sum_{\alpha' \leq r} U_{\alpha\alpha'}(\mathcal{C}_{\mathbf{x}}^{(L)}) \mathcal{L}_{\alpha'}(\mathcal{C}_{\mathbf{x}}^{(L)}),\ \forall \alpha \leq r \tag{1.5}$$

  - `L2P`: Aggregate local expansions ($\boldsymbol{\mathcal{L}}$) at the leaf level on the target particles $\mathbf{x}$ for all target cells $\mathcal{C}_{\mathbf{x}}^{(\bar{L})}$

$$\bar{\phi}(\mathbf{x}_i) = \sum_{\alpha \leq r} U_{i\alpha}(\mathcal{C}_{\mathbf{x}}^{(\bar{L})}) \mathcal{L}_\alpha(\mathcal{C}_{\mathbf{x}}^{(\bar{L})}) \tag{1.6}$$

– P2P: Add the nearfield contributions for each target cell $\mathcal{C}_{\mathbf{x}}^{(\bar{L})}$ at the leaf level

$$\bar{\phi}(\mathbf{x}_i) += \sum_{\mathbf{y}_j \in \mathcal{N}(\mathcal{C}_{\mathbf{x}}^{(\bar{L})})} k(\mathbf{x}_i, \mathbf{y}_j) w(\mathbf{y}_j) \tag{1.7}$$

Algorithm 1: A generic Fast Multipole Algorithm (FMA)

Input: kernel $k(\cdot,\cdot)$, densities $\mathbf{w}$, positions $\mathbf{x}$, order p, leaf level $\bar{L}$, octree *tree*

Output: Potentials $\bar{\phi}$

```
// Upward pass
for level L = L̄, ..., 2 do
    for source cell C_y^(L) ∈ tree do
        if L = L̄ then
            // P2M: from source Particle to leaf Multipole expansion
            M_β(C_y^(L̄)) = Σ_{j=1}^{n_y} V^T_{βj}(C_y^(L̄)) w(y_j), ∀β ≤ r
        else
            // M2M: from 2^d childs to parent
            M_β(C_y^(L)) = Σ_{β'≤r} V^T_{ββ'}(C_y^(L)) M_β'(C_y^(L+1)), ∀β ≤ r
// Downward pass
for level L = 2, ..., L̄ do
    for target cell C_x^(L) ∈ tree do
        // M2L: from Multipole to Local expansions
        for source cell C_y^(L) ∈ I(C_x^(L)) do
            L_α(C_x^(L)) += Σ_{β≤p} C_αβ(T_xy^(L)) M_β(C_y^(L)), ∀α ≤ r
        if L < L̄ then
            // L2L: from parent to 2^d childs
            L_α(C_x^(L+1)) += Σ_{α'≤r} U_αα'(C_x^(L)) L_α'(C_x^(L)), ∀α ≤ r
        else
            // L2P: from leaf Local Expansion to target Particles
            φ̄(x_i) = Σ_{α≤r} U_iα(C_x^(L̄)) L_α(C_x^(L̄))
            // P2P: direct computation of the nearfield
            for source cell C_y^(L̄) ∈ N(C_x^(L̄)) do
                φ̄(x_i) += Σ_{j=1}^{n_y} k(x_i, y_j) w(y_j)
```

1.1.3 Computational cost and asymptotic complexity

Although the linear asymptotic complexity of the FMM and $\mathcal{H}^2$-methods (its algebraic counterpart) is well-established, in this subsection we recall the computational cost of the FMA in

details regardless of the expansion technique (series expansion, polynomial interpolation,...) and regardless of the interaction kernel k. Therefore, we denote

- t_k, the time required to compute $k(\mathbf{x}, \mathbf{y})$ for one couple $(\mathbf{x}, \mathbf{y})$ multiply by the density $w(\mathbf{y})$ and add to the potential $\phi(\mathbf{x})$
- $t_{U/V^T} = \mathcal{O}(r)$, the time required to compute a multipole expansion for a given particle
- $t_C = \mathcal{O}(r^2)$, the time required to compute the basis of the expansion U or V^T at a given level

Consequently, the time required to transfer an expansion from child to parent or the other way around reads as $r \times t_{U/V^T} = \mathcal{O}(r^2)$. For the sake of simplicity, we consider the case, where all cells at all levels are non-empty and the number of particles per leaf is a constant $n_0 = n/2^{d\bar{L}}$, *e.g.*, particles uniformly distributed in a cube.

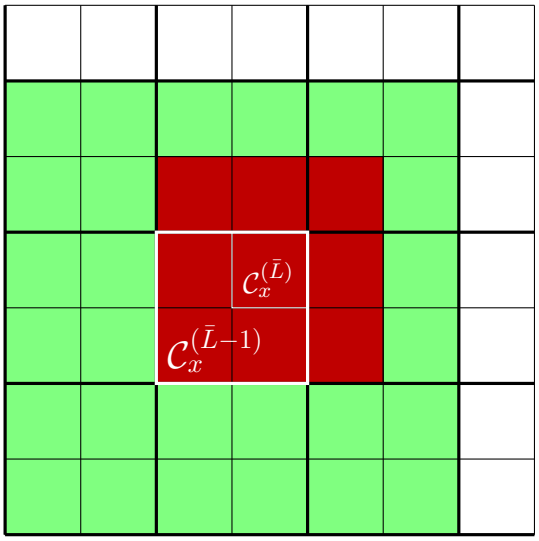

Figure 1.2: The portion of a quadtree at level $\bar{L}$ and $\bar{L}-1$ used to precompute the M2L operators. The interaction list $\mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})$ (green) and neighbor list $\mathcal{N}(\mathcal{C}_{\mathbf{x}}^{(L)})$ (red) of a target particle $\mathbf{x}$ are represented at the leaf level for $\gamma = 2$.

Nearfield interactions Each cell is in nearfield interaction with at most 27 cells (including itself) in 3D. At the leaf level, nearfield interactions are actually computed in a direct fashion, therefore the time t_{P2P} required to compute the P2P step is equal to the time required to compute interactions between n_0 source and $27n_0$ target particles time the number of leaf cells:

$$t_{\text{P2P}} = \sum_{\mathcal{C}_{\mathbf{x}}^{(\bar{L})} \in tree} (n_0 \times 27n_0)t_k = 2^{d\bar{L}} \times n_0 \times 27n_0 \times t_k = n \times 27n_0 \times t_k$$

Hence, the P2P step has a linear complexity in n, *i.e.*, $t_{\text{P2P}} = \mathcal{O}(n)$. In order to maintain this linear complexity, n_0 has to remain constant or at least bounded as n grows. For a given geometry, if particles are uniformly distributed (*i.e.*, homogeneous density of particles) then increasing the tree depth by one level every time the problem size is multiplied by 2^d ensures a constant n_0.

Farfield interactions On the other hand, since the size of the interaction list is bounded for a given level and the number of cells scales like $2^{d\bar{L}} = n/n_0$ then the cost of the farfield is linear in n. Indeed, the asymptotic cost of each step of the FMM can be estimated as follows:

- P2M/L2P
$$t_{\texttt{P2M/L2P}} = (2^{d\bar{L}} \times n_0) \times t_{U/V^T} = n \times t_{U/V^T} = \mathcal{O}(n)$$
- M2M/L2L
$$t_{\texttt{M2M/L2L}} = (\sum_{L=2}^{\bar{L}-1} 2^{dL} \times 2^d) \times (r \times t_{U/V^T}) \leq (n/n_0 \times 2^d) \times (r \times t_{transfer}) = \mathcal{O}(n)$$
- M2L
$$t_{\texttt{M2L}} = \sum_{L=2}^{\bar{L}} 2^{dL} \times 189 \times t_C \leq n/n_0 \times 189 \times t_C = \mathcal{O}(n)$$

Balance The fundamental idea behind FMM is to replace direct computations by low-rank approximations while using a binary tree and interaction lists in order to ensure an asymptotic linear complexity. The common procedure to setup an FMM algorithm is to experimentally determine the order of the expansion in order to setup the accuracy. At this point the cost of computing interactions between cells in farfield or nearfield interactions is fixed but not necessarily well-balanced. Increasing the number of level will increase the size of the farfield, *i.e.*, the number of interactions that are computed in a low-rank fashion, and decrease the number of nearfield interactions, *i.e.*, direct computations. Therefore, the optimal tree depth of any FMM algorithm is usually obtained when nearfield and farfield computations are perfectly balanced.

1.2 Interpolation based FMM

The original FMA as introduced by Greengard and Rokhlin [59] relies on expansions of the kernel $k(r) = 1/r$ in terms of spherical harmonics, *i.e.*, in the basis of Legendre polynomials expressed in spherical coordinates. Since then many algorithms have been derived from this one using the same terminology but implementing other strategies for the expansion of the kernel. They are usually divided in 2 classes, whether the expansion technique depends on the kernel or not.

- A kernel dependent approach implementing
  - Spherical Harmonics expansions for $k(r) = 1/r$ was introduced in the original paper [59].
  - Taylor series expansions for Dislocation Dynamics (elastostatics) was introduced by Arsenlis et al. [8].

  - Hankel series expansions for frequency domain elastodynamics $k(r) = e^{ikr}/r$ was introduced by Chaillat et al. [31].

- Kernel independent approaches like
  - the *kifmm*, introduced by Ying et al. [127], relies on Boundary integral representation.
  - the *skeletonization*-based FMM proposed by Martinsson and Rokhlin [83].
  - the *bbfmm*, introduced by Fong and Darve [54], relies on Chebyshev polynomial interpolation.

In this thesis we focus on polynomial interpolation based variants like the *bbfmm*. First of all, we present the general framework of polynomial interpolation based FMM. Then, we discuss some useful features of polynomial interpolation. Finally, we present the *bbfmm* algorithm using a polynomial basis independent terminology.

1.2.1 Chebyshev polynomial interpolation

As discussed in the thesis of Matthias Messner [86] or Michael Messner [89], the Chebyshev polynomial interpolation is usually a good alternative to the Chebyshev series expansion. For instance, as shown here, the computation of the coefficient c_n takes the basic form of a discrete summation over $p+1$ Chebyshev nodes instead of an integral over $[-1, 1]$.

Basics The most simple way of constructing a polynomial approximation of order p of a continuous function f on $[-1, 1]$ is to interpolate f at $p+1$ equidistant points $\{\bar{x}_i\}_{i=0,\dots,p}$ by

$$f(x) \approx f_p(x) = \sum_{n=0}^{p} c_n x^n$$

The interpolation procedure leads to the resolution of a system of $p+1$ equations. This method is time consuming and becomes numerically unstable as p grows. Moreover, since the set of interpolation points is chosen equally spaced, the interpolating polynomials does not converge uniformly on $[-1, 1]$ (Runge phenomenon). A better choice for $\{\bar{x}_i\}_{i=0,\dots,p}$ is the set of Chebyshev points, *i.e.*, the root of the Chebyshev polynomials of the first kind T_{p+1}, namely

$$\bar{x}_m = \cos\left(\frac{2m+1}{p+1}\frac{\pi}{2}\right), \text{ for } m = 0, \dots, p$$

The resulting interpolating polynomials converge exponentially on $[-1, 1]$ for analytic functions. Moreover, there is a quasi-best approximation result in ∞-norm, namely

$$\|f - f_p\|_\infty \approx \max_{x\in[a,b]} |f(x) - f_p(x)|,$$

that involves the Lebesgue constants, see Rivlin [103].

Lebesgue constants In order to quantify the quality of an interpolation technique, *i.e.*, the choice of interpolation nodes, we often use the Lebesgue constants. These quantities grow only logarithmically with p when interpolating at Chebyshev nodes, whereas they grow exponentially when interpolating on equispaced nodes.

Chebyshev interpolation formula Let us write the polynomial f_p of order p interpolating function f on Chebyshev nodes as a sum of Chebyshev polynomials of the first kind

$$f(x) \approx f_p(x) = \sum_{n=0}^{p} c'_n T_n(x) = \frac{1}{2}c_0 + \sum_{n=1}^{p} c_n T_n(x), \; \forall x \in [-1, 1] \tag{1.8}$$

then the coefficients c_n read as

$$c_n = \frac{2}{p+1} \sum_{m=0}^{p} f(\bar{x}_m) T_n(\bar{x}_m)$$

As shown in [54] and recalled in [86] exponential convergence is also ensured for functions $f(\cdot)$ and kernels $k(\cdot, \cdot)$ when using Chebyshev polynomial interpolation.

A generic interpolation formula Let us now re-write the interpolation formula (1.8) in a more practical and generic form

$$f(x) \approx \sum_{m=0}^{p} S(p; x, \bar{x}_m) f(\bar{x}_m), \; \forall x \in [-1, 1] \tag{1.9}$$

where the interpolation operator S is defined as

$$S(p; x, \bar{x}_m) = \frac{1}{p+1} + \frac{2}{p+1} \sum_{n=1}^{p} T_n(x) T_n(\bar{x}_m) \tag{1.10}$$

The interpolation formula (1.9) is generic in the sense that changing the interpolation polynomials (or nodes) only affects the definition of S. In the following developments we will always consider (1.9) when doing polynomial interpolation. Furthermore, for the sake of clarity we will omit the interpolation order p in the definition of the order-p interpolation operator, *i.e.*,

$$S(x, \bar{x}_m) \equiv S(p; x, \bar{x}_m)$$

1.2.2 Useful features of polynomial interpolation

Interpolating on an arbitrary interval In the general case, we may need to interpolate a function f on an arbitrary $[a, b]$. This can be done provided the mapping function Φ defined

as follows

$$\begin{cases} \Phi : [-1,1] \to [a,b] \\ \qquad\qquad x \mapsto \Phi(x) = \dfrac{a+b}{2} + \dfrac{b-a}{2}x \\ \Phi^{-1} : [a,b] \to [-1,1] \\ \qquad\qquad x \mapsto \Phi^{-1}(x) = \dfrac{2x-b-a}{b-a} \end{cases} \tag{1.11}$$

The interpolation formula of order p then reads as

$$f(x) \approx \sum_{m=0}^{p} S(\Phi^{-1}(x), \bar{x}_m) f(\Phi(\bar{x}_m)),\ \forall x \in [a,b]$$

, *i.e.*,

$$f(x) \approx \sum_{m=0}^{p} S(\frac{2x-b-a}{b-a}, \bar{x}_m) f(\frac{a+b}{2} + \frac{b-a}{2}\bar{x}_m),\ \forall x \in [a,b]$$

The multi-index notation Let f denote a function mapping from $\mathbb{R}^3$ to $\mathbb{C}$, a similar interpolation formula as (1.10) can be derived for f that reads as

$$f(\mathbf{x}) \approx \sum_{|\boldsymbol{\alpha}| \leq p} S(\mathbf{x}, \bar{\mathbf{x}}_{\boldsymbol{\alpha}}) f(\bar{\mathbf{x}}_{\boldsymbol{\alpha}}),\ \forall \mathbf{x} \in [-1,1]^3$$

This formula uses the third-dimensional multi-index notation defined as follows

$$\boldsymbol{\alpha} := (\alpha_1, \alpha_2, \alpha_3), \text{ where } \forall i \in \{1,2,3\},\ \alpha_i = 0, \ldots, p.$$

The summation over all combinations of α_1, α_2 and α_3 can be denoted

$$\sum_{|\boldsymbol{\alpha}| \leq p} \equiv \sum_{\alpha_1=0}^{p} \sum_{\alpha_2=0}^{p} \sum_{\alpha_3=0}^{p}$$

where

$$|\boldsymbol{\alpha}| := \max_{i \leq 3}(\alpha_i)$$

Please note that the interpolation points are represented in a tensorial way such that

$$\bar{\mathbf{x}}_{\boldsymbol{\alpha}} := (\bar{x}_{\alpha_1}, \bar{x}_{\alpha_2}, \bar{x}_{\alpha_3}) \in [-1,1]^3$$

Using this notation, the interpolation operators can be expressed in a shorter form

$$S(\mathbf{x}, \bar{\mathbf{x}}_{\boldsymbol{\alpha}}) := S(x_1, \bar{x}_{\alpha_1}) S(x_2, \bar{x}_{\alpha_2}) S(x_3, \bar{x}_{\alpha_3})$$

which makes them easier to read and manipulate.

Shifting derivatives Polynomial interpolation techniques allow for approximating the derivatives of f at almost no extra cost by simply deriving the interpolation formula. This feature was described and mathematically analyzed in [17, 18] and relies on a stability result, namely a generalization of *Markov's inequality*. Additionally, since derivation decreases the order of the polynomial basis by one, this technique provides an order $p-1$ polynomial approximation of the derivative of f from the interpolation of f at $p+1$ points. Let f denote a univariate analytic function defined on $[-1,1]$, then for all $x \in [-1,1]$ we have

$$f'(x) \approx (f_p(x))' \tag{1.12}$$

This result does not hold in general but holds for analytic functions. Therefore, incorporating the expression of f_p given in (1.9) into the (1.12) yields the following approximation of f'

$$f'(x) \approx \sum_{m=0}^{p} \frac{\partial S}{\partial x}(x, \bar{x}_m) f(\bar{x}_m),\ \forall x \in [-1,1]$$

Hence, approximating f' boils down to shifting the derivative to the interpolating polynomial S in (1.9). This trick is particularly convenient, since S' is known explicitly for most interpolation polynomials. For a function f defined on an arbitrary interval $[a,b]$ the approximation reads as

$$f'(x) \approx \sum_{m=0}^{p} \frac{\partial \Phi^{-1}}{\partial x}(x) \frac{\partial S}{\partial x}(\Phi^{-1}(x), \bar{x}_m) f(\Phi(\bar{x}_m)),\ \forall x \in [a,b] \tag{1.13}$$

If we replace Φ by its definition (1.11), the approximation formula (1.13) becomes

$$f'(x) \approx \sum_{m=0}^{p} \frac{2}{b-a} \frac{\partial S}{\partial x}(\frac{2x-b-a}{b-a}, \bar{x}_m) f(\frac{a+b}{2} + \frac{b-a}{2}\bar{x}_m),\ \forall x \in [a,b]$$

Let f denote a function defined on a 3-dimensional interval $[-1,1]^3$, we have

$$\nabla_{\mathbf{x}} f(\mathbf{x}) \approx \sum_{|\boldsymbol{\alpha}| \leq p} \nabla_{\mathbf{x}} S(\mathbf{x}, \bar{\mathbf{x}}_{\boldsymbol{\alpha}}) f(\bar{\mathbf{x}}_{\boldsymbol{\alpha}}),\ \forall \mathbf{x} \in [-1,1]^d$$

where the gradient is defined as

$$\begin{aligned}
\frac{\partial S}{\partial x_1}(\mathbf{x}, \bar{\mathbf{x}}_{\boldsymbol{\alpha}}) &:= S'(x_1, \bar{x}_{\alpha_1}) S(x_2, \bar{x}_{\alpha_2}) S(x_3, \bar{x}_{\alpha_3}) \\
\frac{\partial S}{\partial x_2}(\mathbf{x}, \bar{\mathbf{x}}_{\boldsymbol{\alpha}}) &:= S(x_1, \bar{x}_{\alpha_1}) S'(x_2, \bar{x}_{\alpha_2}) S(x_3, \bar{x}_{\alpha_3}) \\
\frac{\partial S}{\partial x_3}(\mathbf{x}, \bar{\mathbf{x}}_{\boldsymbol{\alpha}}) &:= S(x_1, \bar{x}_{\alpha_1}) S(x_2, \bar{x}_{\alpha_2}) S'(x_3, \bar{x}_{\alpha_3})
\end{aligned}$$

Applications of this approach to the FMM can be found in [86, 113]. It will often be used in this work in order to decrease the dimensionality of the kernel during the `M2L` step at the

cost of decreasing the accuracy of the approximation scheme.

1.2.3 The *bbfmm*, a Chebyshev interpolation based FMM

In this subsection, we present the interpolation formula of an arbitrary non-oscillatory kernel function and the fast multipole summation scheme associated to it. The resulting algorithm is an interpolation based FMM oblivious to the polynomial basis, that coïncides with the *bbfmm* for the basis of Chebyshev polynomials. Most fundamental mathematical results behind this method were covered 7 years prior the *bbfmm* in the framework of $\mathcal{H}^2$ matrices, see [63].

Interpolating interaction kernels First of all, any smooth (and non-oscillatory) kernel function k can be interpolated independently *w.r.t.* $\mathbf{x}$ and $\mathbf{y}$, thus providing a low-rank representation

$$k(\mathbf{x},\mathbf{y}) \approx \sum_{|\boldsymbol{\alpha}|\leq p} S(\mathbf{x},\bar{\mathbf{x}}_{\boldsymbol{\alpha}}) \sum_{|\boldsymbol{\beta}|\leq p} k(\bar{\mathbf{x}}_{\boldsymbol{\alpha}},\bar{\mathbf{y}}_{\boldsymbol{\beta}}) S(\mathbf{y},\bar{\mathbf{y}}_{\boldsymbol{\beta}}) \tag{1.14}$$

that holds for any $\mathbf{x}$ and $\mathbf{y}$. Then, for any target particle $\mathbf{x} \in \mathcal{C}_{\mathbf{x}}^{(L)}$ and source particle $\mathbf{y} \in \mathcal{C}_{\mathbf{y}}^{(L)}$ in farfield interaction, *i.e.*, such that $\mathcal{C}_{\mathbf{y}}^{(L)} \in \mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})$, the interaction $k(\mathbf{x},\mathbf{y})$ can be approximated within well-controlled error bounds (see [86] for further details). As a result, the farfield contributions to the potential created at position $\mathbf{x}_i$

$$\phi^{far}(\mathbf{x}_i) = \sum_{\mathbf{y}_j \in \mathcal{I}(\mathcal{C}_{\mathbf{x_i}}^{(L)})} k(\mathbf{x}_i,\mathbf{y}_j) w(\mathbf{y}_j)$$

can be approximated at a fixed precision by the following expression

$$\phi^{far}(\mathbf{x}_i) \approx \sum_{|\boldsymbol{\alpha}|\leq p} S(\mathbf{x}_i,\bar{\mathbf{x}}_{\boldsymbol{\alpha}}) \sum_{|\boldsymbol{\beta}|\leq p} k(\bar{\mathbf{x}}_{\boldsymbol{\alpha}},\bar{\mathbf{y}}_{\boldsymbol{\beta}}) \sum_{\mathbf{y}_j \in \mathcal{I}(\mathcal{C}_{\mathbf{x_i}}^{(L)})} S(\mathbf{y}_j,\bar{\mathbf{y}}_{\boldsymbol{\beta}}) w(\mathbf{y}_j)$$

using a low-rank representation of the interaction kernel, where $U_{i\boldsymbol{\alpha}}(\mathcal{C}_{\mathbf{x}}^{(L)}) = S(\mathbf{x}_i,\bar{\mathbf{x}}_{\boldsymbol{\alpha}})$, $V_{\boldsymbol{\beta}j}^T(\mathcal{C}_{\mathbf{y}}^{(L)}) = S(\mathbf{y}_j,\bar{\mathbf{y}}_{\boldsymbol{\beta}})$ and $C_{\boldsymbol{\alpha}\boldsymbol{\beta}}(\mathcal{T}_{\mathbf{xy}}^{(L)}) = k(\bar{\mathbf{x}}_{\boldsymbol{\alpha}},\bar{\mathbf{y}}_{\boldsymbol{\beta}})$. An important feature of polynomial interpolation is that it only requires the evaluation of k on a fixed grid, which ensures both kernel independence and modularity.

Compact notations For the sake of clarity we sometimes simplify notations by omitting the dependence in the grid points, *i.e.*, we denote

$$S_{\boldsymbol{\alpha}}(\mathbf{x}) = S(\mathbf{x},\bar{\mathbf{x}}_{\boldsymbol{\alpha}}),\ S_{\boldsymbol{\beta}}(\mathbf{y}) = S(\mathbf{y},\bar{\mathbf{y}}_{\boldsymbol{\beta}}),\ \bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}} = k(\bar{\mathbf{x}}_{\boldsymbol{\alpha}},\bar{\mathbf{y}}_{\boldsymbol{\beta}}) \tag{1.15}$$

Thus, the compact interpolation formula reads as

$$k(\mathbf{x},\mathbf{y}) \approx \sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) \sum_{|\boldsymbol{\beta}|\leq p} \bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}} S_{\boldsymbol{\beta}}(\mathbf{y}) \tag{1.16}$$

Fast summation scheme The Fast Multipole summation scheme associated with polynomial interpolation is presented in Algorithm 2, while Algorithm 3 defines the functions that are specific to the original *bbfmm* as introduced by Fong and Darve [54]. Here we recall the equations associated with the main steps of the algorithm.

- The Upward Pass consists in performing the following operations at each level from bottom ($L = \bar{L}$) to top ($L = 2$)
  - `P2M`: Aggregate all source contributions from the particles to the multipole expansion at the leaf level

$$\mathcal{M}_{\boldsymbol{\beta}}(\mathcal{C}_{\mathbf{y}}^{(\bar{L})}) = \sum_{j=1}^{n_{\mathbf{y}}} S(\mathbf{y}_j, \bar{\mathbf{y}}_{\boldsymbol{\beta}}) w(\mathbf{y}_j),\ \forall \boldsymbol{\beta}/|\boldsymbol{\beta}| \leq p \tag{1.17}$$

  - `M2M`: Transfer multipole expansions child to parent cells

$$\mathcal{M}_{\boldsymbol{\beta}}(\mathcal{C}_{\mathbf{y}}^{(L)}) = \sum_{|\boldsymbol{\beta'}|\leq p} S(\bar{\mathbf{y}}_{\boldsymbol{\beta}}, \bar{\mathbf{y}}_{\boldsymbol{\beta'}}) \mathcal{M}_{\boldsymbol{\beta'}}(\mathcal{C}_{\mathbf{y}}^{(L+1)}),\ \forall \boldsymbol{\beta}/|\boldsymbol{\beta}| \leq p \tag{1.18}$$

- The Downward Pass consists in performing the following operations at each level from top ($L = 2$) to bottom ($L = \bar{L}$)
  - `M2L`: Transfer all multipole expansions between all admissible cluster pairs

$$\mathcal{L}_{\boldsymbol{\alpha}}(\mathcal{C}_{\mathbf{x}}^{(L)}) += \sum_{|\boldsymbol{\beta}|\leq p} \bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}} \mathcal{M}_{\boldsymbol{\beta}}(\mathcal{C}_{\mathbf{y}}^{(L)}),\ \forall \boldsymbol{\alpha}/|\boldsymbol{\alpha}| \leq p \tag{1.19}$$

  - `L2L`: Transfer local expansions from parent to child cells

$$\mathcal{L}_{\boldsymbol{\alpha}}(\mathcal{C}_{\mathbf{x}}^{(L)}) = \sum_{|\boldsymbol{\alpha'}|\leq p} S(\bar{\mathbf{x}}_{\boldsymbol{\alpha}}, \bar{\mathbf{x}}_{\boldsymbol{\alpha'}}) \mathcal{L}_{\boldsymbol{\alpha'}}(\mathcal{C}_{\mathbf{x}}^{(L-1)}),\ \forall \boldsymbol{\alpha}/|\boldsymbol{\alpha}| \leq p \tag{1.20}$$

  - `L2P`: Aggregate all leaf-level local expansions $\boldsymbol{\mathcal{L}}(\mathcal{C}_{\mathbf{x}}^{(\bar{L})})$ on the target particles $\mathbf{x}$

$$\bar{\phi}(\mathbf{x}_i) += \sum_{|\boldsymbol{\alpha}|\leq p} S(\mathbf{x}_i, \bar{\mathbf{x}}_{\boldsymbol{\alpha}}) \mathcal{L}_{\boldsymbol{\alpha}}(\mathcal{C}_{\mathbf{x}}^{(\bar{L})}) \tag{1.21}$$

  - `P2P`: compute and add the nearfield contributions for each target cell $\mathcal{C}_{\mathbf{x}}^{(\bar{L})}$ at the leaf level

$$\bar{\phi}(\mathbf{x}_i) += \sum_{\mathbf{y}_j \in \mathcal{N}(\mathcal{C}_{\mathbf{x_i}}^{(\bar{L})})} k(\mathbf{x}_i, \mathbf{y}_j) w(\mathbf{y}_j) \tag{1.22}$$

Algorithm 2: A generic interpolation based FMM

Input: kernel $k(\cdot,\cdot)$, densities $\mathbf{w}$, positions $\mathbf{x}$, order p, leaf level $\bar{L}$, octree *tree*

Output: Potentials $\bar{\boldsymbol{\phi}}$

// Upward pass

for *level* $L = \bar{L}, \ldots, 2$ **do**

  for *source cell* $\mathcal{C}_{\mathbf{y}}^{(L)} \in$ *tree* **do**

    if $L = \bar{L}$ **then**

      // P2M: interpolation from source particles to leaf cell

      $\mathcal{M}_{\boldsymbol{\beta}}(\mathcal{C}_{\mathbf{y}}^{(\bar{L})}) = \sum_{j=1}^{n_{\mathbf{y}}} S(\mathbf{y}_j, \bar{\mathbf{y}}_{\boldsymbol{\beta}}) w(\mathbf{y}_j),\ \forall \boldsymbol{\beta}/|\boldsymbol{\beta}| \leq p$

    else

      // M2M:

      $\mathcal{M}_{\boldsymbol{\beta}}(\mathcal{C}_{\mathbf{y}}^{(L)}) = \sum_{|\boldsymbol{\beta'}|\leq p} S(\bar{\mathbf{y}}_{\boldsymbol{\beta}}, \bar{\mathbf{y}}_{\boldsymbol{\beta'}}) \mathcal{M}_{\boldsymbol{\beta'}}(\mathcal{C}_{\mathbf{y}}^{(L+1)}),\ \forall \boldsymbol{\beta}/|\boldsymbol{\beta}| \leq p$

    $\tilde{\boldsymbol{\mathcal{M}}}(\mathcal{C}_{\mathbf{y}}^{(L)}) \leftarrow$`transformM(`$\boldsymbol{\mathcal{M}}(\mathcal{C}_{\mathbf{y}}^{(L)})$`)`

// Downward pass

for *level* $L = 2, \ldots, \bar{L}$ **do**

  for *target cell* $\mathcal{C}_{\mathbf{x}}^{(L)} \in$ *tree* **do**

    // M2L: transfer expansions between interacting cells

    `applyM2L(`$\tilde{\boldsymbol{\mathcal{L}}}(\mathcal{C}_{\mathbf{x}}^{(L)}), \mathcal{C}_{\mathbf{x}}^{(L)}$`)`

    $\boldsymbol{\mathcal{L}}(\mathcal{C}_{\mathbf{x}}^{(L)}) \leftarrow$`transformL(`$\tilde{\boldsymbol{\mathcal{L}}}(\mathcal{C}_{\mathbf{x}}^{(L)})$`)`

    if $L < \bar{L}$ **then**

      // L2L: interpolation from parent to 2^d childs

      $\mathcal{L}_{\boldsymbol{\alpha'}}(\mathcal{C}_{\mathbf{x}}^{(L+1)}) = \sum_{|\boldsymbol{\alpha'}|\leq p} S(\bar{\mathbf{x}}_{\boldsymbol{\alpha}}, \bar{\mathbf{x}}_{\boldsymbol{\alpha'}}) \mathcal{L}_{\boldsymbol{\alpha'}}(\mathcal{C}_{\mathbf{x}}^{(L)}),\ \forall \boldsymbol{\alpha}/|\boldsymbol{\alpha}| \leq p$

    else

      // L2P: interpolation from leaf cell to target particles

      $\bar{\phi}(\mathbf{x}_i) += \sum_{|\boldsymbol{\alpha}|\leq p} S(\mathbf{x}_i, \bar{\mathbf{x}}_{\boldsymbol{\alpha}}) \mathcal{L}_{\boldsymbol{\alpha}}(\mathcal{C}_{\mathbf{x}}^{(\bar{L})})$

      // P2P: direct computation of the nearfield

      for *source cell* $\mathcal{C}_{\mathbf{y}}^{(L)} \in \mathcal{N}(\mathcal{C}_{\mathbf{x}}^{(L)})$ **do**

        $\bar{\phi}(\mathbf{x}_i) += \sum_{\mathbf{y}_j \in \mathcal{C}_{\mathbf{y}}^{(L)}} k(\mathbf{x}_i, \mathbf{y}_j) w(\mathbf{y}_j)$

Algorithm 3: *bbfmm* functions

Function `precomputeM2L()`

```
// For L̄ − 1 levels in tree, or only 1 level for homogeneous kernels
for L = 2, ..., L̄ do
    // Assemble all possible interactions, i.e., 316.
    for i-th source cell C_y^(L) ∈ I_full(C_x^(L)) do
        K̄^i_αβ(L) = k(x̄_α, ȳ_β), ∀α, β/|α|, |β| ≤ p
```

Function `applyM2L(`$\tilde{\mathcal{L}}(\mathcal{C}_{\mathbf{x}}^{(L)}), \mathcal{C}_{\mathbf{x}}^{(L)}$`)`

```
for i-th source cell C_y^(L) ∈ I(C_x^(L)) do
    L̃_α(C_x^(L)) += Σ_{|β|≤p} K̄^i_αβ(L) M̃_β(C_y^(L)), ∀α/|α| ≤ p
```

Function `transformM(`$\boldsymbol{\mathcal{M}}$`)`

```
return M
```

Function `transformL(`$\tilde{\boldsymbol{\mathcal{L}}}$`)`

```
return L̃
```

1.2.4 Computational cost and memory footprint

The computational cost and memory footprint of the M2L step is much larger than the other steps in most FMM variants, this subsection shows that this also applies to the interpolation based FMM. In order to accurately describe the cost of interpolation based FMM such as the *bbfmm*, Table 1.2 and Table 1.1 gather detailed theoretical estimations of its complexity and memory footprint.

Interpolators During the P2M/L2P steps, part of the expansion can be precomputed, *e.g.*, the polynomial basis, and stored at a $\mathcal{O}(p)$ cost. The rest of the expansion is computed on the fly at a $\mathcal{O}(p^3)$ cost. The M2M/L2L step can be slightly reworked in order to be performed in $\mathcal{O}(p^4)$ operations instead of the naive variant, that involves $\mathcal{O}(p^6)$ operations and a memory footprint of $\mathcal{O}(p^3)$. The optimized variant only requires the interpolator S to be assembled on a single fictitious pair of child and parent 1D interpolation grid at each level, which results in a memory footprint of $\mathcal{O}(p^2)$.

M2L operators In the framework of interpolation based FMM, the $(p+1)^3$-by-$(p+1)^3$ matrices $\bar{\mathbf{K}} = \left\{\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right\}_{|\boldsymbol{\alpha}|,|\boldsymbol{\beta}|\leq p}$ are called the M2L operators. These operators are used to transfer multipole expansions $\mathcal{M}$, *i.e.*, expansions *w.r.t.* the sources $\mathbf{y}$, to local expansions $\mathcal{L}$, *i.e.*, expansions *w.r.t.* the targets $\mathbf{x}$. The M2L operators are known explicitly, therefore they are precomputed at each level for each possible pair of interacting cells. Their number reduces to a maximum of 316 per level in 3D thanks to the translation invariance. The $\mathcal{O}(p^6)$ memory footprint of these operators represents the largest part of the memory requirements. As a result, computations involving large tree depth and $p > 10$ exceed the limits of standard desktop computers in term of storage. On the other hand, the cost of applying $\bar{\mathbf{K}}$ to the

multipole expansions scales like $\mathcal{O}(p^6)$. Moreover, these matrices must be applied at all levels and a potentially large number of times (max. 189 interactions per cell per level), which usually makes the M2L step the most computationally expensive step of any Fast Multipole scheme.

Homogeneity In the case of a homogeneous kernel k, *i.e.*,

$$\exists \alpha \in \mathbb{R} / \forall (\lambda, \mathbf{x}, \mathbf{y}) \in \mathbb{R} \times \mathbb{R}^d \times \mathbb{R}^d, k(\lambda \mathbf{x}, \lambda \mathbf{y}) = |\lambda|^\alpha k(\mathbf{x}, \mathbf{y}), \tag{1.23}$$

the expression of $\bar{\mathbf{K}}$ at a given level can be expressed easily in term of $\bar{\mathbf{K}}$ at another level. More precisely, if we store $\bar{\mathbf{K}}^{ref}$, an M2L operator at a fictitious level corresponding to a reference cell width of $\omega_{ref} = 2$, then a level-L M2L operator $\bar{\mathbf{K}}(L)$ can be applied using $\bar{\mathbf{K}}^{ref}$ and a scale factor of $\lambda(\alpha, L) = (\omega_{ref}/\omega_L)^\alpha$. Then, the level-$L$ M2L operations read as

$$\boldsymbol{\mathcal{L}}(\mathcal{C}_{\mathbf{x}}^{(L)}) += \lambda(\alpha, L) \bar{\mathbf{K}}^{ref} \boldsymbol{\mathcal{M}}(\mathcal{C}_{\mathbf{y}}^{(L)}) \tag{1.24}$$

This optimization results in a reduction of the memory footprint by a factor of $\bar{L} - 1 = 7$ for the largest tree depth considered in this thesis, namely $\bar{L} = 8$, but also reduces the precomputation time.

Step	P2M/L2P	Before Computation M2M/L2L	M2L homogeneous	M2L non-homogeneous	During Computation global
Operator	$T(x)$	$S(\bar{x}, \bar{x}')$	$\bar{\mathbf{K}}$	$\bar{\mathbf{K}}^{ref}$	$\boldsymbol{\mathcal{M}}/\boldsymbol{\mathcal{L}}$
# operators	$8^{\bar{L}}$	$3 \times 8(\bar{L} - 1)$	$316(\bar{L} - 1)$	316	n_{cells}
# entries	$(p+1)$	$(p+1)^2$	$(p+1)^6$	$(p+1)^6$	$(p+1)^3$

Table 1.1: Asymptotic memory footprint of each step of the *bbfmm* *w.r.t.* the interpolation order p. In order to get the exact number of bytes one needs to multiply by the value type size, *i.e.*, 4 for single floats and 8 for double floats. The complexities of the non-homogeneous M2L and the M2M/L2L is multiplied by the number of levels, namely $\bar{L} - 1$. For the P2M/L2P the complexities are multiplied by the number of leaves. Finally the order-p^3 expansions stored during the computation is multiplied by the number of non-leaf cells $n_{cells} = \sum_{L=2}^{\bar{L}-1} 2^{dL} = (8^{\bar{L}} - 8^2)/7$.

Step	P2P	P2M/L2P	M2M/L2L	M2L
# flops	$27 \times 8^{\bar{L}} \times n_0^2$	$n \times p^3$	$8 n_{cells} \times 3p^4$	$189(n_{cells} + 8^{\bar{L}}) \times p^6$

Table 1.2: Asymptotic complexity of each step of the *bbfmm* *w.r.t.* the interpolation order p. The complexity of the M2L is multiplied by the number of cells $n_{cells} + 8^{\bar{L}}$ and the complexity of the M2M/L2L by the number of non-leaf cells n_{cells}.

1.3 Related works on M2L-optimized FMM

The main contribution of this thesis, described in Section 5.1, consists in a new M2L-optimized FMM powered by Fast Fourier Transform (FFT). While optimization of the interpolation based M2L operators and conversion of the FMM to Fourier domain have already been investigated independently in the past, they have never been combined. Here,

we discuss those past developments in order to show how the method positions itself in this context.

1.3.1 M2L-optimization of the bbfmm

As investigated in [54], the rank of the interpolation-based M2L operators is not optimal, therefore compression of $\bar{K}$ should be considered. Messner et al. [87] describes several optimizations of the *bbfmm* based on compressing the M2L operators. These methods are briefly described here and will be used as references in our numerical benchmarks (see Section 5.4).

Compressed variant The first variant called the *compressed-bbfmm* and presented in Algo 4 relies on the global compression of the set of M2L operators. The resulting factorization of the M2L allows for shifting part of its application to the P2M/L2P step and store the result in transformed expansions. This method usually requires a significant amount of precomputation time and does not always provide optimal compression and application time as opposed to the second variant.

Algorithm 4: *compressed-bbfmm* functions

Function `precomputeM2L()`

```
// For all levels in tree, or 1 level if kernel is homogeneous
for L = 2, ..., L̄ do
    // Assemble all possible interactions, i.e., 316.
    for i-th source cell C_y^(L) ∈ I_full(C_x^(L)) do
        K̄^i_αβ(L) = k(x̄_α, ȳ_β), ∀α, β/|α|, |β| ≤ p
    // Compress the entire set of interactions.
    U_L, [C_L^1, ..., C_L^316], V_L ← SVD[K̄^1(L), ..., K̄^316(L)]
```

Function `applyM2L(`$\tilde{\mathcal{L}}(\mathcal{C}_{\mathbf{x}}^{(L)}), \mathcal{C}_{\mathbf{x}}^{(L)}$`)`

```
for i-th source cell C_y^(L) ∈ I(C_x^(L)) do
    L_α(C_x^(L)) += Σ_{|β|≤p} (C_L^i)_αβ M_β(C_y^(L)), ∀α/|α| ≤ p
```

Function `transformM(`$\mathcal{M}$`)`

```
// Apply V_L^T.
M̃ = V_L^T M
```

Function `transformL(`$\tilde{\mathcal{L}}$`)`

```
// Apply U_L.
L = U_L L̃(, ∀α/|α| ≤ p
```

Symmetric variant For kernels exhibiting symmetries, like $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{y}, \mathbf{x})$, the *symmetric-bbfmm* presented in Algo 5 allows for significantly reducing the memory footprint of the *bbfmm* while reducing both precomputation and running time. In the *symmetric-bbfmm*,

the symmetries of the kernel and the interpolation grid allow for reducing the size of the interaction list from $|\mathcal{I}_{full}(\mathcal{C}_{\mathbf{x}}^{(L)})| = 316$ to $|\mathcal{I}_{sym}(\mathcal{C}_{\mathbf{x}}^{(L)})| = 16$ for a given level L. Consequently, we need to store less `M2L` operators and they can be applied to a set of multipole expansions at once, however expansions have to be permuted properly before the `M2L` translation. On the other hand, individual compression of the operators becomes affordable, which results in a near-optimal low-rank compression and thus a faster application of each `M2L` operator. In particular, a fully or partially pivoted Adative Cross Approximation (ACA) is used to compress the `M2L` operators with or without assembling it. Then, recompression can be performed using a QR Decomposition and a subsequent Singular Value Decomposition (SVD). Although this method is kernel dependent it applies to a wide range of kernels, *e.g.*, RBF kernels and kernel satisfying $k(\mathbf{x}, \mathbf{y}) = k(|\mathbf{x} - \mathbf{y}|)$ in general.

Algorithm 5: *symmetric-bbfmm* functions

Function `precomputeM2L()`

```
// For all levels in tree, or 1 level if kernel is homogeneous
for L = 2, ..., L̄ do
    // Assemble all possible interactions, i.e., 16
    for i-th source cell C_y^(L) ∈ I_sym(C_x^(L)) do
        U_L^i, V_L^i ← ACA[K̄^i(L)]
```

Function `applyM2L(`$\tilde{\mathcal{L}}(\mathcal{C}_{\mathbf{x}}^{(L)}), \mathcal{C}_{\mathbf{x}}^{(L)}$`)`

```
for i-th source cell C_y^(L) ∈ I(C_x^(L)) do
    L_α(C_x^(L)) += Σ_{|β|≤p} Σ_r (U_L^i)_{αr} (V_L^i)_{rβ} M_β(C_y^(L)), ∀α/|α| ≤ p
```

1.3.2 Conversion of a `FMM` to Fourier domain

Principle An FFT conversion of the original Fast Multipole Algorithm was originally proposed in a technical report by Greengard et al. [60], and later implemented and improved by Elliott et al. [49]. For an efficient parallel implementation of this approach please refer to Pan et al. [97]. The general idea of the original FFT conversion proposed in these papers consists in rewriting the `M2L` operations in the form of a 2D linear convolution, then converting it to a 2D circular convolution (using zero-padding) in order to finally perform the `M2L` operations in Fourier domain at a lower cost using FFT.

Limits This approach differs from our method since it is based on expanding the kernel $1/r$ in spherical harmonics while our method relies on *kernel-independent* polynomial interpolations. On the other hand, both variants require zero-padding for the conversion to circular convolution. The FFT conversion of the FMA algorithm additionally suffers from *smearing effects*, *i.e.*, some entries are artificially not zeroed out due to the conversion to circular convolution form. While our method suffers from the Runge phenomenon due to the interpolation on an equispaced grid, the original variant suffers from various sources of instability. The first occurs when a coefficient with strong variations in magnitude is warped

in order to rewrite the expansion as a linear convolution. The second instability results from a scaling problem that amplifies during the FFT operations. Stabilization techniques were proposed in [60] and further discussed and improved in [49]. In Section 5.1, we also suggest methods in order to defeat the Runge phenomenon that may occur in our method.

2

Dislocation Dynamics simulations

Contents

2.1 Introduction to Dislocation Dynamics

Developing a deeper understanding of material's behaviour is crucial for the design of new structures. For instance, providing fine models for the plastic behaviour of complex alloys submitted to high gradients of temperature and high levels of radiations has become a key issue in nuclear safety.

A multiscale approach The general multiscale approach to material modeling is represented on Figure 2.1, it can be summarized in 3 main scales:

- Molecular Dynamics (MD) represents the material at the atomistic scale. MD simulations allows to identify elastic laws by analyzing atomistics interactions on very large sets of atoms.
- Dislocation Dynamics (DD) bridges the gap between the MD scale and the FEM scale by modeling the moving plans of atoms represented in terms of line defects called dislocations. DD can be used to simulate either a single grain or a large number of grains. Ultimately, DD produces empirical plasticity laws by simulating the propagation of massive ensembles of dislocations.
- Finite Element Methods (FEM) are the well-known and broadly used methods that aim at modeling elastoplastic or more complex media at the mesoscale. The internal stress or strain states generated by FEM is usually used for design purposes.

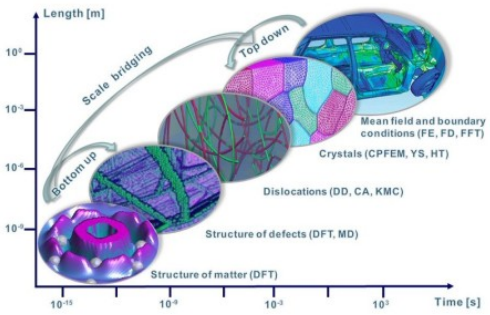

Figure 2.1: Illustrations of a material along the various steps of the multiscale modelling. The axis show the typical time and length scales associated to the methods.

The DD model Dislocations are by definition the result of an eigenstrain state in a medium, see Mura's book [92]. They are usually studied in the context of lattice defects in crystalline materials. In particular, the simulation of massive ensembles of dislocations allows for a better understanding of the mesoscopic plasticity laws. Several models have emerged in order to make such simulations computationally tractable to large ensembles. In this thesis we focus on the most widely spread approach, namely Dislocation Dynamics (DD). Here, we will only recall the fundamentals of the nodal approach to DD simulations, please refer to Bulatov and Cai [27] for an extensive introduction to the topic or see Kubin [72] for a more physics-oriented approach.

Topology Dislocations consist in closed line defects that are ubiquitous in crystalline materials. As shown on Figure 2.2, a dislocation line (in red) is characterized by a Burger's vector $\mathbf{b}$ representing the intensity and direction of the dislocation. At any point $\mathbf{x}$ of the line we also define the line tangent $\mathbf{t}(\mathbf{x})$. In the *nodal approach* to DD, dislocations lines are discretized by segments and nodes. In particular, segments are treated as straight dislocation lines with constant line tangent vectors $\mathbf{t}$, and all segments discretizing a dislocation loop share the same Burger's vector $\mathbf{b}$ as the loop. Finally, the segment length denoted L is given in Å(short for ångström, 1Å= 0.1nm= 10^{-10}m), typically 20Å$\leq L \leq$ 100Å. In this thesis, all lengths and distances involved in DD are given in Å.

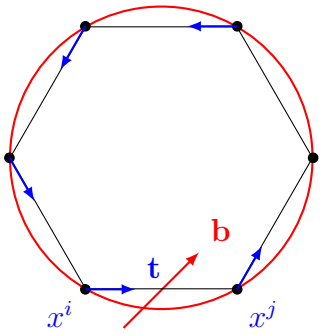

Figure 2.2: A dislocation loop discretized in 6 segments. All segments share the same Burger's vector $\mathbf{b}$ as the loop but they have various line tangents $\mathbf{t}$ depending on their orientations.

Dynamics and algorithm The motion of dislocations is usually initiated by an external elastic stress state applied to the material. This stress states acts on dislocation lines along with the internal stress state created by the dislocation network itself. The resulting forces applied on the lines create the motion according to advanced mobility laws. While moving, dislocations can collide, merge, annihilate or even multiply, creating a potentially significant modification of the network's topology. Please refer to Arnaud Etcheverry's thesis [51] for an extensive presentation of the algorithm and the efficient implementation in *OptiDis* code*.

2.2 Integral representations of the isotropic elastic fields

In this section we present expressions for the isotropic elastic stress, forces and energy fields in integral form. The elastic stress field is first represented in integral form using Mura's formula. Then, we recall the definition of the elastic nodal forces acting on the extremities of finite dislocation lines. Finally, we address the integral representation of the elastic energy.

*See *OptiDis* code's home page at http://optidis.gforge.inria.fr/

2.2.1 Isotropic elastic stress through Mura's formula

Mura's formula The elastic stress field created at point $\mathbf{x}$ by a dislocation loop $(\mathcal{C}')$ with Burger's vector $\mathbf{b'}$ is given in [92] by Mura's formula as

$$\sigma_{ij}(\mathbf{x}) = C_{ijkl} \oint_{(\mathcal{C}')} \varepsilon_{lnh} C_{pqmn} G_{kp,q}(|\mathbf{x} - \mathbf{x'}|) b'_m dx'_h, \tag{2.1}$$

where G_{ij} is the Green's function for the infinite elastic media characterized by the Hooke's tensor $\mathbf{C}$. We denote $d\mathbf{x'}(\mathbf{x'})$ the infinitesimal variation of $\mathbf{x'}$ and $\mathbf{t'}(\mathbf{x'})$ the line tangent vector at point $\mathbf{x'}$. Thus, if $dx'(\mathbf{x'}) = |d\mathbf{x'}|$ then $d\mathbf{x'} = \mathbf{t'}(\mathbf{x'})dx'(\mathbf{x'})$. Finally, $\boldsymbol{\varepsilon}$ denotes the permutation symbol and we omit the summation over repeated indices using Einstein's summation convention. A demonstration of Mura's formula can be found in Mura's book [92].

Isotropic elastic stress field In isotropic elasticity, the Hooke's tensor reduces to the 2 Lamé constants (μ, ν) and the Green's function reads as

$$G_{ij}(r) = \frac{1}{8\pi\mu}(\delta_{ij} r_{,pp} - \frac{1}{2(1-\nu)} r_{,ij}), \tag{2.2}$$

where $\mathbf{r} = \mathbf{x} - \mathbf{x'}$ and $r = |\mathbf{r}|$. The notation $r_{,i}$ denotes the partial derivatives of r *w.r.t.* x_i, *i.e.*,

$$r_{,i} = \frac{\partial r}{\partial x_i}.$$

Replacing the expression (2.2) of $\mathbf{G}$ in (2.1) yields

$$\sigma_{ij}((\mathcal{C}'), \mathbf{x}) = \frac{\mu b'_k}{8\pi} \oint_{(\mathcal{C}')} r_{,mpp}(\varepsilon_{jmk} dx'_i + \varepsilon_{imk} dx'_j) + \frac{2}{1-\nu} \varepsilon_{nmk}(r_{,ijm} - \delta_{ij} r_{,ppm}) dx'_n \tag{2.3}$$

Let us write (2.3) in a slightly different form and distinguish 2 main contributions

$$\sigma_{ij}((\mathcal{C}'), \mathbf{x}) = \frac{\mu}{8\pi}(\sigma^A_{ij} + \sigma^A_{ji})((\mathcal{C}'), \mathbf{x}) + \frac{2}{1-\nu}(\sigma^B_{ij} - \delta_{ij}\sigma^B_{pp})((\mathcal{C}'), \mathbf{x})) \tag{2.4}$$

where $\boldsymbol{\sigma}^{\mathbf{A}}$ and $\boldsymbol{\sigma}^{\mathbf{B}}$ are defined as follows

$$\sigma^A_{ij}((\mathcal{C}'), \mathbf{x}) = \oint_{(\mathcal{C}')} r_{,mpp}(\mathbf{x} - \mathbf{x'}) \varepsilon_{jmk} b'_k t'_i dx'$$
$$\sigma^B_{ij}((\mathcal{C}'), \mathbf{x}) = \oint_{(\mathcal{C}')} r_{,ijm}(\mathbf{x} - \mathbf{x'}) \varepsilon_{nmk} b'_k t'_n dx'$$

Once $\boldsymbol{\sigma}^{\mathbf{A}}(\mathbf{x})$ and $\boldsymbol{\sigma}^{\mathbf{B}}(\mathbf{x})$ are computed one can easily deduce the expression of $\boldsymbol{\sigma}(\mathbf{x})$ using (2.4). In the isotropic case $\boldsymbol{\sigma}$ is symmetric. More precisely, both terms $\sigma^A_{ij} + \sigma^A_{ji}$ and $(\sigma^B_{ij} - \delta_{ij}\sigma^B_{pp})$ are symmetric. Hence, we can always compute σ_{ij} for $i \leq j$ and deduce the remaining components by transposition.

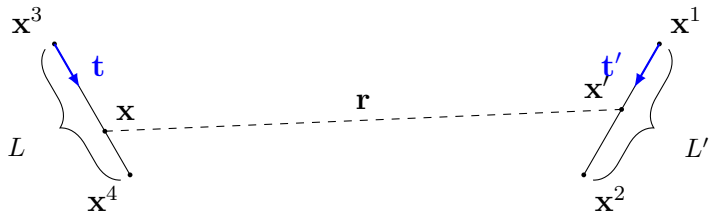

Figure 2.3: Parametrization of 2 interacting dislocation segments, the source $\left[\mathbf{x}^1\mathbf{x}^2\right]$ and the target segment $\left[\mathbf{x}^3\mathbf{x}^4\right]$.

Finite dislocation line Although (2.3) is only defined for a closed dislocation loop, not only is it useful but also necessary to write this expression for a segment, *i.e.*, a finite dislocation line, in order to be able to sum up all contributions of a given discretized dislocation loop. Let us consider a source segment $[\mathbf{x}^1\mathbf{x}^2]$, *i.e.*, extremity 1 (resp. 2) is located at point $\mathbf{x}^1$ (resp. $\mathbf{x}^2$), characterized by a Burger's vector $\mathbf{b'}$, a line tangent $\mathbf{t'}$ and a length L'. Thus, the elastic stress created by the finite dislocation line $[\mathbf{x}^1\mathbf{x}^2]$ reads as

$$\sigma_{ij}(\left[\mathbf{x}^1\mathbf{x}^2\right],\mathbf{x}) = \frac{\mu}{8\pi}(\sigma^A_{ij}+\sigma^A_{ji})(\left[\mathbf{x}^1\mathbf{x}^2\right],\mathbf{x}) + \frac{2}{1-\nu}(\sigma^B_{ij}-\delta_{ij}\sigma^B_{pp})(\left[\mathbf{x}^1\mathbf{x}^2\right],\mathbf{x})$$

where $\boldsymbol{\sigma}^{\mathbf{A}}([\mathbf{x}^1\mathbf{x}^2],\mathbf{x})$ and $\boldsymbol{\sigma}^{\mathbf{B}}([\mathbf{x}^1\mathbf{x}^2],\mathbf{x})$ are defined as follows

$$\sigma^A_{ij}(\left[\mathbf{x}^1\mathbf{x}^2\right],\mathbf{x}) = \int_{\mathbf{x}^1}^{\mathbf{x}^2} r_{,mpp}(\mathbf{x}-\mathbf{x'})\varepsilon_{jmk}b'_k t'_i dx' = \int_0^{L'} r_{,mpp}(\mathbf{x}-\mathbf{x'})\varepsilon_{jmk}b'_k t'_i d\xi'$$
$$\sigma^B_{ij}(\left[\mathbf{x}^1\mathbf{x}^2\right],\mathbf{x}) = \int_{\mathbf{x}^1}^{\mathbf{x}^2} r_{,ijm}(\mathbf{x}-\mathbf{x'})\varepsilon_{nmk}b'_k t'_n dx' = \int_0^{L'} r_{,ijm}(\mathbf{x}-\mathbf{x'})\varepsilon_{nmk}b'_k t'_n d\xi'$$

where $\xi' \in [0, L']$ denotes the curvilinear abscissae of $\mathbf{x'}$, $\mathbf{x'} = \mathbf{x}^1 + \xi'\mathbf{t'}$ and $dx' = d\xi'$.

Non-singular formalism The integrals involved in (2.3) actually exist, however they cannot be evaluated numerically using standard quadratures because of the singularity occuring when r approaches 0, *i.e.*, $\mathbf{x'} \to \mathbf{x}$. Attempts at regularizing the integrals have been introduced in Brown [26], that gave the first definition of the core, then Peierls [98] and Nabarro [93] developed the idea of spreading the Burger's vector around the dislocation line. In this thesis focus is made on the non-singular formulation introduced by Cai et al. [28]. This method roughly consists in removing the singularity by considering a slightly different Green's function, *i.e.*,

$$G^{ns}_{ij}(r_a) = \frac{1}{8\pi\mu}(\delta_{ij}r_{a,pp} - \frac{1}{2(1-\nu)}r_{a,ij}),$$

where the distance is shifted by a small and constant contribution, namely a parameter a called the *core width*, such that

$$r_a = \sqrt{a^2 + r^2}.$$

The newly defined Green's function G^{ns} is non-singular since for any $(\mathbf{x},\mathbf{x'}) \in (\mathbb{R}^3)^2$, $r_a(\mathbf{x},\mathbf{x'}) \neq 0$. Not only does this formulation imply very little modifications in terms of

implementation compared to the singular variant, but it also relates to the atomistic models*. Cai et al. [28] also provides analytical expressions for the isotropic elastic stress field and energy using this formalism. These expressions were initially implemented in OptiDis code for the direct computation of the isotropic elastic stress, forces and energy.

2.2.2 Isotropic elastic nodal forces and energy

Sources of motion In the DD model, dislocation lines move under the effect of elastic forces acting on nodes, that divide in 2 contributions: the internal nodal forces $f = f^{int.}$ and the external nodal forces $f^{ext.}$. While the former is induced by the elastic stress field $\boldsymbol{\sigma}$ created by dislocations themselves, the latter is the result of an external stress field $\boldsymbol{\sigma}^{ext.}$ enforced on the boundary of the grain.

The Peach-Koehler force and nodal forces The local force induced by an elastic stress state $\boldsymbol{\sigma}$ on any point $\mathbf{x}$ of a dislocation loop $(\mathbf{b}, \mathbf{t}, L)$ is called the Peach-Koehler force and it reads as

$$\mathbf{f}^{PK}(\mathbf{x}) = (\boldsymbol{\sigma}(\mathbf{x}) \cdot \mathbf{b}) \times \mathbf{t}$$

On the other hand, the elastic nodal forces $\mathbf{f}^i$ acting at the extremity $i = 3, 4$ of a finite dislocation line $[\mathbf{x}^3\mathbf{x}^4]$ characterized by $(\mathbf{b}, \mathbf{t}, L)$ are obtained by integration of the Peach-Koehler force on $[\mathbf{x}^3\mathbf{x}^4]$, *i.e.*,

$$\mathbf{f}^i = \int_{\mathbf{x}^3}^{\mathbf{x}^4} \mathbf{f}^{PK}(\mathbf{x}) N^i(\mathbf{x}) dx$$

where $N^{i=3,4}$ are piecewise linear shape functions, such that $N^i(\mathbf{x}_j) = \delta_{ij}$. If we denote ξ the curvilinear abscissae of $\mathbf{x}$, then $\mathbf{x} = \mathbf{x}^3 + \xi\mathbf{t}$, $dx = d\xi$ and the linear shape functions $N^{i=3,4}$ read as

$$N^4(\mathbf{x}) = \frac{|\mathbf{x} - \mathbf{x}^3|}{L} = \frac{\xi}{L} \text{ and } N^3(\mathbf{x}) = \frac{|\mathbf{x}^4 - \mathbf{x}|}{L} = \frac{L - \xi}{L} = 1 - \frac{\xi}{L} \tag{2.5}$$

Hence, the force acting on node 4 and node 3 reads as

$$\mathbf{f}^4 = \int_0^L \mathbf{f}^{PK}(\mathbf{x}) \frac{\xi}{L} d\xi$$
$$\mathbf{f}^3 = \int_0^L \mathbf{f}^{PK}(\mathbf{x}) d\xi - \mathbf{f}^4$$

External forces The external stress state is usually enforced at the boundary of the grain by a simple Dirichlet boundary condition. Therefore, it is considered constant within the grain (far from the boundary). The external nodal forces are computed by integration of the Peach-Koehler force associated to the external stress state $\boldsymbol{\sigma}^{ext.}$ over the target segment

*This theory is obtained by spreading the Burger's vector smoothly over the core of the dislocation.

$[\mathbf{x}^3\mathbf{x}^4]$, *i.e.*,

$$\mathbf{f}^{ext.,4} = \int_0^L (\boldsymbol{\sigma}^{ext.} \cdot \mathbf{b}) \times \mathbf{t} \frac{\xi}{L} d\xi = \frac{L}{2} (\boldsymbol{\sigma}^{\mathbf{ext.}} \cdot \mathbf{b}) \times \mathbf{t}$$

Hence, external forces are computed directly at a linear cost in n.

Internal forces Similarly, the internal nodal force is computed by integration of the Peach-Koehler force associated to the internal stress state $\boldsymbol{\sigma}^{int.} = \boldsymbol{\sigma}$ over the target segment $[\mathbf{x}^3\mathbf{x}^4]$, *i.e.*,

$$f_i^4 = \int_0^L \varepsilon_{ijk} \sigma_{jp}(\mathbf{x}) b_p t_k \frac{\xi}{L} d\xi, \ \forall i \in \{1,2,3\} \tag{2.6}$$

Therefore, the non-singular contribution of $\boldsymbol{\sigma}^{\mathbf{A}}$ to the internal forces can be expressed as the following double line integral

$$f_i^{A,4} = \int_0^L \varepsilon_{ijk} b_p t_k \frac{\xi}{L} d\xi \int_0^{L'} r_{a,mrr}(\mathbf{x} - \mathbf{x'}) \varepsilon_{pms} b'_s t'_j d\xi', \ \forall i \in \{1,2,3\} \tag{2.7}$$

and similarly for the contribution of $\boldsymbol{\sigma}^{\mathbf{B}}$. Hence, in a naive implementation, the cost of computing the internal forces over the entire network is quadratic in n.

Elastic energy Let us consider the following expression of the isotropic elastic energy of a dislocation network $(\mathcal{C})$ taken from [28]

$$E((\mathcal{C})) = -\frac{\mu}{8\pi}(E_A + \frac{2}{1-\nu}(E_B - E_C + \nu E_D)) \tag{2.8}$$

where contributions E_A, E_B, E_C and E_D are defined as

$$E_A = \oint_{(\mathcal{C})} b_i dx_i \oint_{(\mathcal{C})} r_{a,kk} b'_j dx'_j \ , \quad E_B = \oint_{(\mathcal{C})} b_j dx_k \oint_{(\mathcal{C})} r_{a,ij} b'_i dx'_k$$
$$E_C = \oint_{(\mathcal{C})} b_i dx_j \oint_{(\mathcal{C})} r_{a,kk} b'_i dx'_j \ , \quad E_D = \oint_{(\mathcal{C})} b_j dx_i \oint_{(\mathcal{C})} r_{a,kk} b'_i dx'_j$$

This expression originates from De Wit et al. [41] along with various other forms. All existing forms are equivalent up to an integral, that vanishes on a closed loop.

2.3 Existing Fast Multipole DD formulations

Since the interactions between dislocations are long-ranged and fast-decreasing (see the expression of the Green's function (2.2)), a common approach is to use the Fast Multipole Method (FMM) in order to reduce the asymptotic cost of computing the elastic fields of a n-segments network from a quadratic to a linear cost in n. While nearfield interactions are evaluated directly using analytical expressions, farfield interactions are approximated in order to be evaluated more efficiently.

Direct evaluation of the nearfield As mentioned by Arsenlis et al. [8] the size of the quadrature required to evaluate (2.6) up to a fixed accuracy grows dramatically as the distance between dislocation lines decreases. Therefore, it is not recommended to evaluate the double line integral involved in the nodal forces numerically. Instead, [8] relies on the non-singular theory [28], *i.e.*, $\sigma = \sigma^{ns}$, in order to derive analytical expressions of the force field created by a source segment and acting on a parallel or a non-parallel segment, as well as on the segment itself. For the computation of the nearfield contributions, *OptiDis* implements its own analytical expressions that are derived from the non-singular theory [28]. For equivalent analytical expressions, please refer to [8] for the stress and force fields (2.7), or [28] for the energy (2.8).

Approximation of the farfield Several formulations have already been considered for the approximation of the farfield contributions to the elastic stress and energy, such as expansions in spherical harmonics [130] or Taylor expansions [120]. The well-known DD code *ParaDis* developed at Lawrence Livermore National Laboratory first implemented Taylor expansions of the isotropic elastic fields [8], and only recently incorporated polynomial interpolation based on the *bbfmm*. The aim of our work was to implement an efficient interpolation scheme into *OptiDis*, the parallel implementation of *NumoDis* code (CEA Saclay), that did not involve farfield approximation yet.

3

Randomized low-rank approximation of matrices

Contents

3.1 The Randomized SVD

3.1.1 Introduction to randomized algorithms

Randomized algorithms for the low-rank approximation of matrices, *a.k.a.*, randomized Low-Rank Approximation (LRA) algorithms, usually provide powerful alternatives to standard matrix factorizations for matrices of relatively low-rank such as the Singular Value Decomposition (SVD), QR Decomposition, Cholesky Decomposition, … Recently, they have gained popularity in the numerical linear algebra community because they are easy to implement, highly parallelizable and most of all they achieve very competitive performance within reasonable accuracy. Moreover, these algorithms often involve very basic matrix computations thus leaving significant room for improvement. They have drawn much attention in scientific fields such as geostatistics [71, 42], machine learning [47, 121], genetics [79]. For further references on their applications to data analysis please refer to Mahoney's survey article [78]. Randomized algorithms usually divide in 2 classes: *random sampling*- and *random projection*-based algorithms. Both approaches rely on building a sketch version of the input matrix in order to extract relevant information. The random sampling (also called *column selection*) approach basically consists in selecting a subset of columns and rows, while the random projection more generally consists in multiplying the input matrix by a sketching matrix, that can be either sparse or dense. During this thesis we mainly focused on a dense random projection based LRA algorithm called the Randomized SVD (rSVD), that we present in details in the following subsection.

3.1.2 The Randomized SVD

Standard Decomposition First of all, we recall the formalism of the standard SVD. Let $\mathbf{C}$ be an arbitrary m-by-n real matrix, then its SVD reads as

$$\mathbf{C} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{3.1}$$

where $\mathbf{U} \in \mathbb{R}^{m\times m}$, resp. $\mathbf{V} \in \mathbb{R}^{n\times n}$, denotes the set of m left singular vectors, resp. n right singular vectors, associated with the singular values of $\mathbf{C}$, that we denote $\sigma(\mathbf{C})$. Finally, the diagonal matrix $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ contains the singular values of $\mathbf{C}$, *i.e.*,

$$\Sigma_{ij} = \delta_{ij}\sigma_i(\mathbf{C}), \forall i, j \in \{1, \dots, n\}\,.$$

We denote $\mathbf{C}_r$ the matrix obtained after truncation of the full SVD representation at rank r, *i.e.*,

$$\mathbf{C} \approx \mathbf{C}_r = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^T \tag{3.2}$$

where $\mathbf{U} \in \mathbb{R}^{m\times r}$, resp. $\mathbf{V} \in \mathbb{R}^{n\times r}$, denotes the first r left singular vectors, resp. right singular vectors, associated with the first r singular values $\sigma(\mathbf{C})$. According to *Eckart-Young theorem*, $\mathbf{C}_r$ provides the best rank-r approximation of $\mathbf{C}$. In particular, if the rank

of $\mathbf{C}$ equals r, then $\tilde{\mathbf{C}} = \tilde{\mathbf{C}}_r$. For self-adjoint positive definite matrices $\mathbf{C} \in \mathbb{R}^{n\times n}$, the SVD boils down to

$$\mathbf{C} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^T, \tag{3.3}$$

a.k.a. the Eigen Value Decomposition (EVD). If $\mathbf{C}$ has negative eigenvalues, then $\boldsymbol{\Sigma}$ contains their absolute value. The full EVD can be obtained at an $\mathcal{O}(n^3)$ cost using optimized LAPack routines (*e.g.*, Intel MKL wrapper), while the rank-r truncated EVD can be obtained iteratively at an $\mathcal{O}(rn^2)$ cost using the Arnoldi's algorithm provided by ARPack.

Principle & Algorithm The rSVD is a very popular *random projection*-based LRA algorithm introduced and further enhanced in a series of papers [84, 77, 124], that provides approximate low-rank representation in SVD form at a quadratic cost. The 2 stage of the rSVD for a *symmetric* input matrix $\mathbf{C} \in \mathbb{R}^{n\times n}$ given a prescribed numerical rank r are summarized in Algorithm 6, they consist in:

- A *randomized range approximation*: Form $\mathbf{Q} \in \mathbb{R}^{n\times r}$, an approximate length-$r$ basis for the range of $\mathbf{C}$.
  - Perform a Gaussian random projection, *i.e.*, application of $\mathbf{C}$ to a n-by-r Gaussian random matrix $\boldsymbol{\Omega}$
    $$\mathbf{Y} = \mathbf{C}\boldsymbol{\Omega} \tag{3.4}$$
  - and a subsequent orthogonalization, *e.g.*, a thin QR Decomposition, such that
    $$\mathbf{QR} = \mathbf{Y}$$
    with an upper triangular matrix $R \in \mathbb{R}^{r\times r}$ and an orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n\times r}$.
  - Hence, $\mathbf{Q}$ approximate the range of $\mathbf{C}$ and we can get a rank-r approximation of $\mathbf{C}$ in the form
    $$\tilde{\mathbf{C}} = \mathbf{Q}\mathbf{Q}^\mathbf{T}\mathbf{C}\mathbf{Q}\mathbf{Q}^\mathbf{T} \tag{3.5}$$
    with tight error bounds that hold with high probability.
- A *standard matrix factorization*: In order to provide $\tilde{\mathbf{C}}$ in SVD form, *i.e.*, $\tilde{\mathbf{C}} = \tilde{\mathbf{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\mathbf{U}}^T$, we perform the following operations:
  - We start by assembling the r-by-r matrix $\mathbf{B}$ such that
    $$\mathbf{B} = \mathbf{Q}^\mathbf{T}\mathbf{C}\mathbf{Q} \tag{3.6}$$
  - then we compute the full SVD of the small matrix $\mathbf{B}$, *i.e.*,
    $$\mathbf{B} = \mathbf{U}_\mathbf{B}\boldsymbol{\Sigma}_\mathbf{B}\mathbf{U}_\mathbf{B}^\mathbf{T}. \tag{3.7}$$

– Finally, we obtain an approximate rank-r SVD of $\mathbf{C}$ by forming the exact rank-r SVD of $\tilde{\mathbf{C}}$, *i.e.*,

$$\tilde{\mathbf{U}} = \mathbf{Q}\mathbf{U}_\mathbf{B} \text{ and } \tilde{\mathbf{\Sigma}} = \mathbf{\Sigma}_\mathbf{B}.$$

A comprehensive review of the method can be found in Halko et al. [64]. In this review, we learn that the algorithm is fairly generic and thus easily extends to other factorizations such as the Cholesky Decomposition or the Interpolative Decomposition. Moreover, it provides various improvements such as iterative and pass-efficient variants. The full algorithm is presented in Algorithm 6, please refer to [64] for the details of the fixed rank and fixed accuracy *approximate range finders*.

Asymptotic complexity Since the overall cost of the rSVD is dominated by the cost of the r matrix multiplications required in each stage, the algorithm has a $\mathcal{O}(n^2 \times r)$ asymptotic complexity in time. While standard LRA algorithms have a cubic complexity, algorithms that produce truncated low-rank presentations (like the Arnoldi SVD or the pivoted QR Decomposition) usually have a quadratic complexity. However, the rSVD is supposed to have a much lower running time than Arnoldi's variant. We verify this assumption on numerical benchmarks in Section 7.4. Moreover, randomized algorithms leave significant room for optimizations and parallelization.

Algorithm 6: The Randomized SVD

Input: symmetric matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$, rank r or accuracy ε, number of power iterations q, oversampling parameter s

Output: $\left[\tilde{\mathbf{U}}, \tilde{\mathbf{\Sigma}}\right]$ approximate SVD of $\mathbf{C}$

`// Stage I: Approximate the range of` $\mathbf{C}$

if *accuracy* ε *is prescribed* **then**

  $[\mathbf{Q}, r] = ARRF(\mathbf{C}, n, q, s, \varepsilon)$ — $\mathcal{O}(r \times n^2)$

if *rank* r *is prescribed* **then**

  $[\mathbf{Q}, \varepsilon] = RSI(\mathbf{C}, n, q, s, r)$ — (idem)

`// Stage II: Decompose` $\tilde{\mathbf{C}} = \mathbf{Q}(\mathbf{Q}^\mathbf{T}\mathbf{C}\mathbf{Q})\mathbf{Q}^\mathbf{T}$ `as` $\tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{U}}^\mathbf{T}$

Build $\mathbf{B} = \mathbf{Q}^\mathbf{T}\mathbf{C}\mathbf{Q} \in \mathbb{R}^{r \times r}$ — $\mathcal{O}(r \times n^2)$

Perform SVD of $\mathbf{B} = \mathbf{U}_\mathbf{B}\mathbf{\Sigma}_\mathbf{B}\mathbf{U}_\mathbf{B}^\mathbf{T}$ — $\mathcal{O}(r^3)$

Form $\tilde{\mathbf{U}} = \mathbf{Q}\mathbf{U}_\mathbf{B}$ and $\tilde{\mathbf{\Sigma}} = \mathbf{\Sigma}_\mathbf{B}$ — $\mathcal{O}(n \times r^2)$

Randomized range finders Algorithm 6 implements the random projection during the *randomized range approximation*. It can be performed either directly using one large matrix-to-matrix multiplication or iteratively by means of multiple small matrix-to-matrix multiplications. More precisely, if the rank of the matrix is known a priori, then the Randomized Subspace Iterations (RSI) algorithm can be used to approximate the range directly given a prescribed rank r, see Algorithm 4.4 in [64]. However, if the rank is not known in advance and accuracy has to be monitored, the Adaptive Randomized Range Finder (ARRF) can be used to return a *near-optimal* range approximation given a prescribed accuracy ε, see Algorithm 4.1 in [64]. Due to their random nature these algorithms have a non-zero chance to

fail, however error bounds such as (3.11) generally hold with high probability, *e.g.*, at least $1 - n \times 10^{-r}$ for the RSI algorithm. For instance, a rank-r approximation of a symmetric matrix of size $n = 10^6$ has a 1 chance to fail out of $10^{10-6} = 10^4$.

Improving accuracy In order to improve the accuracy of the range approximation the projection can be performed on a slightly larger subspace, namely using a n-by-$(r + s)$ random matrix $\mathbf{\Omega}$ where s is called the *oversampling* parameter, and finally keeping the first r columns of $\mathbf{Q}$. Furthermore, the original algorithm does not apply well to matrices with a slow decreasing (or relatively flat) spectrum, as it fails to identify the most important singular values. A common alternative is to stretch this spectrum by using q subspace iterations, *i.e.*, by considering $(\mathbf{CC}^*)^q\mathbf{C}$ instead of $\mathbf{C}$ as the input matrix. The resulting $\tilde{\mathbf{C}}$ approximates $\mathbf{C}$ within well-established and controlled error bounds, that can be expressed in Frobenius or Spectral norm. First of all, the error of the rSVD, namely $\|\mathbf{C} - \tilde{\mathbf{C}}\|$, cannot be lower than the error of the best rank-r approximation of $\mathbf{C}$ provided by the SVD, namely $\|\mathbf{C} - \mathbf{C}_r\|$, which translates to

$$\|\mathbf{C} - \mathbf{C}_r\| \leq \|\mathbf{C} - \tilde{\mathbf{C}}\| \tag{3.8}$$

This deterministic lower bound is also called the *baseline* in [64] and it reads as

$$\|\mathbf{C} - \mathbf{C}_r\|_F = \left(\sum_{i=r+1}^{n} \sigma_i^2(\mathbf{C}) \right)^{1/2} \tag{3.9}$$

in Frobenius norm and

$$\|\mathbf{C} - \mathbf{C}_r\|_S = |\sigma_{r+1}(\mathbf{C})| \tag{3.10}$$

in Spectral norm. An upper bound for the average error in Frobenius norm can be expressed in term of the baseline as

$$\mathbb{E}(\|\mathbf{C} - \tilde{\mathbf{C}}\|_F) \leq f_F(r, s, q)\|\mathbf{C} - \mathbf{C}_r\|_F, \tag{3.11}$$

where f_F is a polynomial function of the rank r that ensures tighter error bounds in average as s and q grow, see [64] for a comprehensive discussion on error bounds and explicit expressions of f_F and f_S, the equivalent in Spectral norm.

3.2 Fast Randomized LRA

In this section we give a brief overview of the existing fast algorithms for performing LRA of matrices using randomized techniques.

3.2.1 Fast random projection

The acceleration of randomized LRA algorithms based on *random projection* usually relies on the ability to apply fast matrix multiplication, hence most research works in this area focus on exploiting the structure and the sparsity of the input matrix C and the random matrix Ω.

Random matrix Random projection algorithms exploiting the nature of the random matrix are presented by Woodruff [123] in a comprehensive survey within the general framework of matrix sketching. While Sarlós [105] first mentioned the idea of using structured random (sketching) matrices such as the Fast Johnson Lindenstrauss Transforms (FJLT), Ailon et al. [5] described this algorithm in full details. The idea is based upon Achlioptas's work [3] that aimed at developing sparse sketching for more efficient random projection. Li et al. [74] in turn developed a very sparse *random projection*-algorithm based on this work. Woolfe et al. [124] subsequently designed an FJLT algorithm based on the Discrete Fourier Transform, namely the Subsampled Randomized Fourier Transform (SRFT). The ability to compute such transform in quasi-linear time allowed for reducing the computational cost of the random projection from $\mathcal{O}(n^2 \times r)$ to $\mathcal{O}(n^2 \times \log r)$. Such method can be used to compute LRAs of matrices, in fact in Liberty et al. [77] the SRFT is used to speedup the randomized Interpolative Decomposition introduced in Martinsson et al. [84]. However, it is not clear yet if there exists a fixed accuracy variant for fast transform based projection schemes. Various efficient algorithms for performing dense random projection are studied in Liberty's thesis [75, 76] such as the general FJLT, the SRFT, the Subsampled Randomized Hadamard Transform (SRHT) (based on the Discrete Walsh-Hadamard Transform [19]) as well as the Mailman algorithm for matrix multiplication. Finally, Tropp [114] provided an improved error analysis for the SRHT.

Input matrix To our knowledge, very few articles address the acceleration of the matrix multiplications in *Gaussian random projection*-based algorithm for dense structured input matrices. In the context of low-rank approximations of matrices, Martinsson [82] describes a way to build HSS matrices using a Fast Multipole Method (FMM)-powered *Gaussian random projection* algorithm. The FMM is used there in order to accelerate the projection stage, *i.e.*, the product between the input matrix and the Gaussian random matrix (3.4). Then, the factorization in HSS form is handled by the Interpolative Decomposition.

3.2.2 Fast random sampling

Block structure Random sampling techniques are particularly well-suited for applications requiring relatively low accuracy LRA. The most widely used random sampling algorithm for *spd* matrices is the Nyström method, see [47, 57, 129]. Applications of the Nyström method to the LRA of covariance kernel matrices can be found in Wang et al. [117]. An improved variant of the original Nyström method introduced in Si et al. [108] exploiting the

block structure of kernel matrices is known as the Memory Efficient Kernel Approximation (MEKA). It relies on a pre-clustering of the data and a block-wise LRA. Very recently, Wang et al. [118] provided an algorithm based on a similar idea and called the structured Block Basis Factorization (BBF), that does not have stability issues and exhibits smaller standard deviation than the MEKA.

Hybrid approaches The complementarity of random sampling and random projection algorithms have often been highlighted in the literature [57]. Therefore, the design of an efficient randomized algorithm often relies on the relevant combination of both approaches. The most obvious example of fast methods for random sampling algorithm is probably the fast approximation of matrix coherence using the randomized SVD proposed by Drineas and Mahoney [46]. On the other hand, fast random projection techniques such as the SRHT or the SRFT are tightly connected to improved random column selection. Finally, [118] makes use of random sampling, random projection and clustering in order to design an efficient $\mathcal{O}(n)$ factorization of a covariance kernel matrix with a low dependence on the ambient dimension. Since it is taylored for the high dimensions involved in machine learning applications, the BBF should perform worse than the FMM in 3D applications.

4

Positioning & Contributions

Contents

Many applications of scientific computing require fast methods for applying large scale dense matricial operators, that can efficiently deal with the dimensionality of the problem. Such methods should be fairly generic and not too intrusive, in order to allow more flexibility and provide a better applicability to a wider range of problems and formulations. In particular, hierarchical methods such as kernel-independent Fast Multipole Method (FMM) allow linear time computations of kernel matrices involved in computational physics, geostatistics,… On the other hand, randomized low-rank approximation techniques provide a powerful generic framework to compute approximate standard decomposition of large low-rank matrices. Here, we describe the context of our research and our motivations for developing faster FMM schemes and randomized Low-Rank Approximation (LRA) techniques.

4.1 Efficient interpolation based FMM

As mentioned in Section 1.2, the *bbfmm* suffers from a suboptimal low-rank representation of the `M2L` operators [54], that is mainly due to the 3D interpolation procedure. Even if `M2L` operators can be re-compressed [87], the algorithms are usually relatively involved and computationally intensive. For instance, they can significantly increase the precomputation time for an arbitrary interaction kernel, as shown by our numerical benchmarks in Section 5.4. Moreover, these optimizations are often kernel-dependent, in particular they depend on certain features of the kernel such as its symmetry or its homogeneity. Finally, these optimizations are based on low-rank approximations, therefore they need to be carefully tuned in order to control the extra error they introduce in the representation of the kernel matrix. Consequently, there is a great need for a generic and exact method that allows for a faster precomputation and application of the `M2L` with lower memory footprint, *i.e.*, a more optimal low-rank representation of the `M2L` operators. There is at least 2 options, that we could consider in order to find an interpolation scheme that produces a more optimal rank. The first one would be to use less interpolation points but selecting them more cleverly in order to maintain accuracy, as described in Casenave [30]. Another option would be to use interpolation grids, that produce structured `M2L` operators. If those operators can be easily express in low-rank form, then they can be applied faster with a lower memory footprint. In Chapter 5 we present a new efficient interpolation based FMM, called the *ufmm*, that dramatically reduces the cost of the `M2L` and the overall running time of the FMM. This method relies on an equispaced grid therefore it becomes instable for large interpolation orders, however many real life applications only require relatively low accuracy and can thus fully benefit from the performance of the *ufmm*. In the following chapters we address 2 applications requiring fast matrix multiplications for either tensorial kernels or scalar kernels with many right-hand-sides.

4.2 Fast Multipole DD simulations

Although most existing Dislocation Dynamics (DD) codes implement a Fast Mutipole evaluation of the elastic fields, the formulations often lack efficiency and genericity. First of all, FMM formulations based on Spherical Harmonics or Taylor expansions usually have slow convergence, therefore they require a large number of modes to provide a relatively low accuracy on the force field evaluation. Second of all, these methods depend heavily on the kernel, therefore switching from one representation of the elastic field to another requires a new implementation and a significant amount of work. In order to provide efficient farfield evaluation of the isotropic elastic fields, we implemented interpolation based FMM in *OptiDis*, a DD parallel code using *ScalFMM* as an FMM engine. However, the tensorial nature of the interactions makes the problem inherently expensive to compute. Therefore, the *bbfmm* may not perform well and we may require more efficient methods such as the *ufmm*. Additional work should also be considered in order to reduce the effect of computing tensorial farfield interactions. Finally, the main purpose of this work is not to compete with existing DD codes but rather to propose a different alternative and extend the capabilities of *OptiDis*. Moreover, performing comparisons between the state-of-the-art codes is in our opinion neither easy nor relevant. In fact, all implementations exhibit major differences in term of parametrization and tuning. Additionally, most existing codes are not open-source, including *OptiDis*, and the few available releases, *e.g.*, *ParaDis*, are far from having the optimal performance of the development branch.

4.3 Fast Randomized LRA

The growing interest for randomized LRA methods and the lack of efficient software solution brought us to develop an optimized code implementing fast numerical methods for random projection based algorithms. In particular, we provided fast algorithms for 2 applications involving the computation of large covariance matrices given as correlation kernels or computed from distance matrices. Let us describe the reasons that motivated us to address such problems in full details.

Gaussian Random Fields First of all, many scientific applications based on Monte Carlo approaches arising from geosciences, cosmology or image processing, rely heavily on intensive covariance matrix computations. A very usual issue encountered in such applications is the generation of large ensembles of multivariate Gaussian Random Variables, *a.k.a.*, Gaussian Random Field (GRF), given an input correlation kernel. In particular, the efficient computation of a square root of the associated covariance matrix cannot be addressed by standard matrix decomposition algorithms. The existing alternatives usually have strong numerical limitations and often do not benefit from the shape of the distribution of points. Randomized LRA techniques provide convenient alternatives, since they benefit from the genericity of the standard matrix decompositions but they simply rely on matrix multipli-

cations. Furthermore, a hierarchical method such as the FMM allows for decreasing the asymptotic complexity of the method from quadratic to linear in time, while avoiding the assembly of the full matrix and benefiting from the spatial distribution of points. Finally, the algorithm can benefit from the genericity and performance of the *ufmm* in order to be applied to any kernel at a low computational cost. Hierarchical methods are becoming more and more popular for performing faster covariance matrix computations [6, 7] but are only rarely combined with random projection techniques, we propose to overcome this shortcoming.

Taxonomy Our last contribution is an attempt at extending the capabilities of our library to arbitrary matrices arising from biological applications. The classification of biological species can be addressed using dimensionality reduction algorithms such as the Multidimensional Scaling (MDS). More precisely, MDS allows for visualizing a point cloud in a low dimensional subspace given a distance matrix, by simply computing a square root of the associated covariance matrix, better known as *similarity* matrix. Since distance matrices are computed from real-life data, namely samples of $\mathcal{O}(10^5)$ reads coming from new generation DNA sequencing, they result in large but low-resolution covariance matrices. However, most fast MDS implementations rely on column selection based LRA, *i.e.*, on the Nyström method, therefore they are often restricted to *spd* matrices and may require a significant amount of columns in order to represent the point cloud with sufficient accuracy. On the other hand, a random projection based LRA such as the randomized Singular Value Decomposition (SVD) is a well-suited and competitive algorithm for the computation of such square root, since it also gives useful extra information on the data and the structure of the *similarity* matrix. In Chapter 7, we further discuss the benefits of using random projection over common algorithms for such applications and its potential complementarity with the state-of-the-art approaches.

Part II

Efficient *kernel* methods for n-body problems

5

An efficient interpolation based FMM

Contents

In this chapter, we introduce a new computationally efficient Fast Multipole Method (FMM) that reduces the cost of the `M2L` operators, namely the *ufmm* (Algo. 7). We also derive a block low-rank algorithm, called the *smooth-ufmm*, that is optimized for globally smooth kernels. Then, we evaluate the theoretical complexity and memory requirements of these algorithms and compare them to the *bbfmm*. Finally, we evaluate the accuracy and running times on artificial numerical benchmarks and we analyze the relative performance of the *ufmm* and *smooth-ufmm* compared to optimized variants of the *bbfmm*.

5.1 A new FFT-accelerated FMM, the Uniform FMM

In the present section, we show that interpolating on equispaced grids allows for dramatically decreasing the memory footprint and the computational cost of the `M2L` operators in the *bbfmm*, namely from $\mathcal{O}(p^6)$ to $\mathcal{O}(p^3 \log p)$ for precomputation and $\mathcal{O}(p^3)$ for application. First of all, we describe the basics of the method and its usual limitations. Then, we discuss the structure of the associated `M2L` operators and improvements based on Fast Fourier Transform (FFT). Finally, we present the algorithm as well as optimizations and implementation details.

5.1.1 A new FMM based on equispaced interpolation grids

Uniform interpolation grid For the sake of clarity, let us first consider the 1D case. The interpolation formula based on an equispaced grid still reads as (1.9) but involves different expressions for the interpolation points $\bar{x}$ and the interpolators $S(x, \bar{x})$. More precisely, the interpolation points are defined as

$$\bar{x}_m = -1 + \frac{2m}{p}, \forall m = 0, \dots, p \tag{5.1}$$

and the polynomial interpolators, *a.k.a.*, the Lagrange polynomials, take the following form

$$S_n(x) \equiv S(p; x, \bar{x}_n) = \prod_{\substack{m=0 \\ m \neq n}}^{p} \frac{x - \bar{x}_m}{\bar{x}_n - \bar{x}_m}, \forall x \in [-1, 1], \forall n = 0, \dots, p. \tag{5.2}$$

Addressing Runge phenomenon Interpolation schemes based on equispaced grids usually become unstable for high orders of interpolation p (Runge phenomenon), however as described in a review by Boyd et al. [23] efficient regularization methods already exist. They are for instance based on optimization techniques such as Tikhonov regularization [21], series expansions such as Gegenbauer regularization [22], subsampling techniques such overdetermined least-squares and Mock-Chebyshev interpolation [25]) or even multi-domain approaches [24]. Moreover, these methods defeat Runge phenomenon while preserving subgeometric convergence. In our approach divergence is not likely to occur since we use interpolation on multiple subdomains and in almost all cases of interest the value of p re-

mains relatively small, *e.g.*, $p < 20$. In particular, machine accuracy was reached with our algorithm for a wide range of non-oscillatory kernels, *e.g.*, $1/r$, $1/r^2$, correlation kernels (Section 5.4) and various isotropic elastostatic Green's functions arising in Dislocations Dynamics (Section 6.4.2).

5.1.2 Optimization of the M2L using the Fast Fourier Transform

Structure of the M2L operators The M2L operators associated to a uniform interpolation grid exhibit a well-known structure that can be used to provide a more efficient fast multipole summation scheme. Indeed, since $k(\cdot,\cdot)$ is evaluated on equispaced 1D source (respectively target) grids denoted $\{\bar{x}_i\}_{i=0\ldots p}$ (respectively $\{\bar{y}_i\}_{i=0\ldots p}$) and only depends on the distance between grid points, *i.e.*,

$$k(x_i, y_j) = k(x_i - y_j) \tag{5.3}$$

for all $i, j = 0, \ldots, p$, then each diagonal of $\bar{\mathbf{K}}$ contains constant values (see Fig. 5.1 left). We say that the M2L operator $\bar{\mathbf{K}}$ is a Toeplitz matrix. In the case of 2D grids the resulting matrix is block Toeplitz, *i.e.*, the matrix is composed of constant blocks over its diagonals while each block is itself a Toeplitz matrix (see Fig. 5.2 left). In 3D, we introduce an extra level of blocking meaning that the constant diagonal blocks are now block Toeplitz.

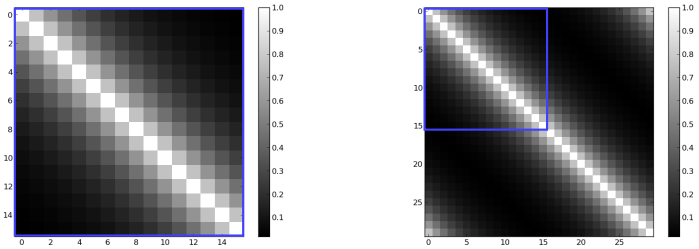

Figure 5.1: Circulant embedding (right) of a symmetric 1D Toeplitz matrix (left), namely a Gaussian kernel matrix evaluated on a 1D grid of 16 points. The Toeplitz matrix (blue frame) is embedded in the upper left corner of the circulant matrix (right).

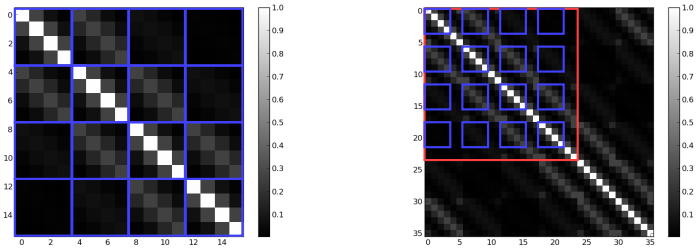

Figure 5.2: Circulant embedding (right) of a symmetric 2D Toeplitz matrix (left), namely a Gaussian kernel matrix evaluated on a 1D grid of 4-by-4 points. The small Toeplitz matrices (blue frames) are embedded in the upper left corner of the small circulant matrices (right), while blocks are also embedded in the upper left corner (red frame) of a circulant block structure (right).

Circulant embedding The embedding of a Toeplitz matrix in a larger circulant matrix is a very common technique that allows for efficiently applying a matrix to vector product

as a convolution in Fourier space. Here, we briefly recall this method in the case of a 1-dimensional M2L operator $\bar{\mathbf{K}}$ of order p. First of all, since $\bar{\mathbf{K}}$ is Toeplitz, it is fully defined by its first row

$$\mathbf{R} = (R_0, \dots, R_p) \in \mathbb{R}^{p+1}, \text{ with } R_i = k(\bar{x}_0, \bar{x}_i) \text{ for } i = 0, \dots, p.$$

and its first column

$$\mathbf{C} = (C_0, \dots, C_p)^t \in \mathbb{R}^{p+1}, \text{ with } C_i = k(\bar{x}_i, \bar{x}_0) \text{ for } i = 0, \dots, p.$$

These row and column can be embedded into a row $\tilde{\mathbf{R}} \in \mathbb{R}^{\tilde{p}}$ defined as

$$\tilde{\mathbf{R}} = (R_0 = C_0, \dots, R_p, C_p, \dots, C_1)$$

with $\tilde{p} = (p+1) + p = 2p + 1$. This new row can be used to generate a circulant matrix $\bar{\mathbf{E}} \in \mathbb{R}^{\tilde{p} \times \tilde{p}}$, that embeds $\bar{\mathbf{K}}$ in its upper left corner (see Fig. 5.1 blue frame on the right). As a result, the application of both matrices to a vector is equivalent, if the last p columns and rows of $\bar{\mathbf{E}}$ are masked. On the other hand, if the Toeplitz matrix is symmetric, then it is uniquely defined by its first row $\mathbf{R}$ and the associated the embedding is slightly smaller, namely $\tilde{p} = p + 1 + (p-1) = 2p$. The circulant embedding of a 1D (resp. 2D) symmetric Toeplitz matrix is illustrated on the right-hand-side of Figure 5.1 (resp. Figure 5.2).

Conversion to Fourier domain The discrete convolution theorem implies that the set of eigenvectors of any circulant matrix $\bar{\mathbf{E}}$ coïncides with the Discrete Fourier Transform (DFT) operator, *i.e.*,

$$\mathbf{F} = \left\{ \tilde{p}^{-1/2} e^{-2i\pi mn/\tilde{p}} \right\}_{m,n=0,\dots,\tilde{p}} ,$$

while the DFT of the first column of $\bar{\mathbf{E}}$ yields the vector $\mathbf{\Lambda}$ containing the eigenvalues of $\bar{\mathbf{E}}$, *i.e.*, $\mathbf{\Lambda} = \mathbf{F}\tilde{\mathbf{R}}$. Let us consider a multipole expansion $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{p+1}$ and a resulting local expansion $\boldsymbol{\mathcal{L}} \in \mathbb{R}^{p+1}$, such that

$$\boldsymbol{\mathcal{L}} = \bar{\mathbf{K}}\boldsymbol{\mathcal{M}} \tag{5.4}$$

If $\tilde{\boldsymbol{\mathcal{M}}} \in \mathbb{R}^{\tilde{p}}$ denotes the vector obtained after padding $\boldsymbol{\mathcal{M}}$ with p zeros, then for $i = 0, \dots, p$ we have

$$(\boldsymbol{\mathcal{L}})_i = \left(\bar{\mathbf{E}}\tilde{\boldsymbol{\mathcal{M}}}\right)_i = \left(\mathbf{F}^*\mathbf{diag}(\mathbf{\Lambda})\mathbf{F}\tilde{\boldsymbol{\mathcal{M}}}\right)_i = \left(\mathbf{F}^*\left[\mathbf{\Lambda} \odot \mathbf{F}\tilde{\boldsymbol{\mathcal{M}}}\right]\right)_i = \left(\mathbf{F}^*\left[\mathbf{F}\tilde{\mathbf{R}} \odot \mathbf{F}\tilde{\boldsymbol{\mathcal{M}}}\right]\right)_i$$

Hence, the matrix to vector product (5.4) can be performed in the discrete Fourier space in the form of an entrywise product $\mathbf{F}\tilde{\mathbf{R}} \odot \mathbf{F}\tilde{\boldsymbol{\mathcal{M}}}$. The same method applies to the 3D case, except that the embedding has to be performed independently in each dimension. In particular, the expansions and the row $\tilde{\mathbf{R}}$ need to stored as 3D arrays and transformed by 3D FFT. Transfers back and forth between Fourier and physical domains are done by forward and backward FFT resulting in an asymptotic overall cost of $\mathcal{O}(p^3 \log p)$ for precomputation

and $\mathcal{O}(p^3)$ for application.

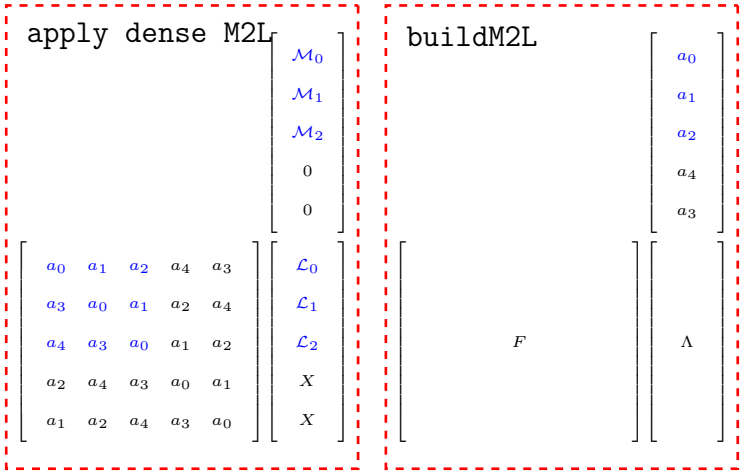

Figure 5.3: Left: Multiplication of a 1D M2L operator for $p = 2$ (in blue) by a vector, after embedding it in a circulant matrix. Right: Precomputation of M2L operator in the Fourier domain.

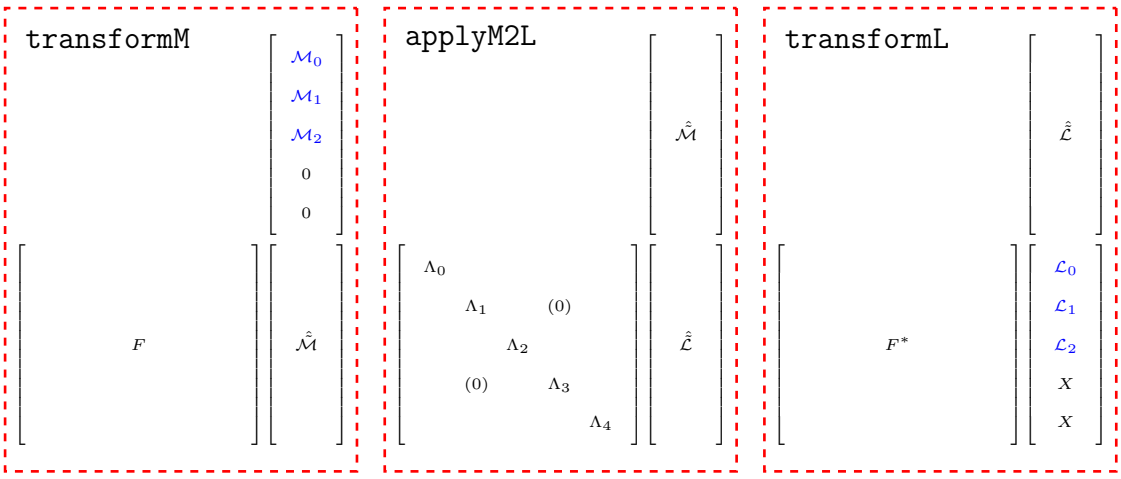

Figure 5.4: Left: Padding and Forward DFT of the multipole expansion. Center: Application of the M2L operator in the Fourier domain. Right: Backward DFT and unpadding of the local expansion.

5.1.3 Algorithm and efficient implementation of the *ufmm*

Let us show how we can efficiently adapt such feature in a Fast Multipole implementation by describing various optimizations and then discuss the associated gains in terms of computational cost and memory footprint. The resulting algorithm, called the *ufmm*, is presented

in Algo 7.

Algorithm 7: *ufmm* functions

Function `precomputeM2L()`

// Assemble M2L operators (in Fourier domain)

for *source cell* $\mathcal{C}_{\mathbf{y}}^{(L)} \in \mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})$ **do**

// Compute first row and column of 3D block Toeplitz M2L operators

$R_{\boldsymbol{\beta}} = k(\bar{\mathbf{x}}_{\mathbf{0}}, \bar{\mathbf{y}}_{\boldsymbol{\beta}}), C_{\boldsymbol{\beta}} = k(\bar{\mathbf{x}}_{\boldsymbol{\beta}}, \bar{\mathbf{y}}_{\mathbf{0}}), \forall \boldsymbol{\beta}/|\boldsymbol{\beta}| \leq p$

// Embed R in the first row $\tilde{R}$ of a 3D block circulant matrix

for $\boldsymbol{\beta}/|\boldsymbol{\beta}| < \tilde{p} = 2p+1$ **do**

if $\beta_d > p$ **then**

$\tilde{R}_{\boldsymbol{\beta}} = R_{\boldsymbol{\beta}'}$ with $\beta'_d = 2p + 1 - \beta_d$ and $\beta'_i = \beta_i$ for $i \neq d$

else

$\tilde{R}_{\boldsymbol{\beta}} = C_{\boldsymbol{\beta}}$

// Apply 3D DFT

$\hat{\tilde{\mathbf{R}}} = \mathbf{F}\tilde{\mathbf{R}}$ with $F_{\boldsymbol{\alpha}\boldsymbol{\beta}} = e^{-\frac{2i\pi}{\tilde{p}}(\boldsymbol{\alpha}\cdot\boldsymbol{\beta})}, \forall(\boldsymbol{\alpha}, \boldsymbol{\beta})/|\boldsymbol{\alpha}|, |\boldsymbol{\beta}| < \tilde{p}$

Function `applyM2L(`$\hat{\tilde{\boldsymbol{\mathcal{L}}}}(\mathcal{C}_{\mathbf{x}}^{(L)}), \mathcal{C}_{\mathbf{x}}^{(L)}$`)`

// Apply M2L operators (in Fourier domain)

for *source cell* $\mathcal{C}_{\mathbf{y}}^{(L)} \in \mathcal{I}(\mathcal{C}_{\mathbf{x}}^{(L)})$ **do**

$\hat{\tilde{\mathcal{L}}}_{\boldsymbol{\alpha}}(\mathcal{C}_{\mathbf{x}}^{(L)}) += \delta_{\boldsymbol{\alpha}\boldsymbol{\beta}} \hat{\tilde{R}}_{\boldsymbol{\alpha}} \hat{\tilde{\mathcal{M}}}_{\boldsymbol{\beta}}(\mathcal{C}_{\mathbf{y}}^{(L)}), \forall \boldsymbol{\alpha}/|\boldsymbol{\alpha}| < \tilde{p}$

Function `transformM(`$\boldsymbol{\mathcal{M}}$`)`

// Pad expansion with zeros and transfer to Fourier domain.

$\tilde{\boldsymbol{\mathcal{M}}} = \mathbf{0}_{\tilde{\mathbf{p}}}$

$\tilde{\mathcal{M}}_{\boldsymbol{\beta}} = \mathcal{M}_{\boldsymbol{\beta}}, \forall \boldsymbol{\beta}/|\boldsymbol{\beta}| \leq p$

$\hat{\tilde{\boldsymbol{\mathcal{M}}}} = \mathbf{F}\tilde{\boldsymbol{\mathcal{M}}}$

Function `transformL(`$\hat{\tilde{\boldsymbol{\mathcal{L}}}}$`)`

// Transfer back to physical domain and unpad.

$\tilde{\boldsymbol{\mathcal{L}}} = \mathbf{F}^{-1}\hat{\tilde{\boldsymbol{\mathcal{L}}}}$

$\mathcal{L}_{\boldsymbol{\alpha}} = \tilde{\mathcal{L}}_{\boldsymbol{\alpha}}, \forall \boldsymbol{\alpha}/|\boldsymbol{\alpha}| \leq p$

Fast Fourier Transforms In order to design an efficient algorithm we need to perfom as few operations as possible during the M2L step, therefore the forward transformations of the multipole expansions (resp. backward transformation of local expansion) are done during the P2M/M2M (resp. L2P/L2L) steps, as in the *compressed-bbfmm* (Algo. 4). Furthermore, we only precompute the first row of the circulant embedding of M2L operators and immediately transform and store them in Fourier domain. Consequently, the only operations that remain in the M2L steps are entrywise products between complex valued vectors of size $\tilde{p}$. However, in order to minimize the cost of storing larger complex valued expansions, it is recommended to apply the backward transformations right after the entrywise product with the M2L operators. Hence, we avoid the storage of the transformed local expansions, which

can be crucial for large tree depths (see Section 5.3.3).

Real valued transforms In the *ufmm*, the cost of storing and applying the M2L operators decreases from $\mathcal{O}(p^6)$ to $\mathcal{O}(p^3)$ with a slightly larger constant that can be divided by 2 using symmetries of the DFT. In 1D, since both the expansions and the M2L operators are real valued arrays, their 1D DFTs return *conjugate-even*[*] vectors. Consequently, only the first entry and the first half of the complex valued entries of the conjugate-even vectors need to be stored, *i.e.*, $p+1$ complex entries for an input real valued array of size $2p+1$. In multiple dimensions, this data packing only affects the last dimensions, *e.g.*, in 2D the output of the 2D DFT of a $(2p+1)$-by-$(2p+1)$ real valued array is a $(2p+1) \times (p+1)$ complex valued array. Hence, the storage of the M2L and the multipole expansions is roughly divided by 2. Moreover, since the application of the M2L is a simple entrywise product, the output as a similar format and the computational time is also divided by 2.

1D transform The block circulant embedding is a convenient approach, since it is relatively easy to visualize (at least in 1D or 2D). However, there exists slightly different embedding, that may be harder to visualize but lead to a fully circulant embedding matrix. Let us illustrate such embedding on a simple 2D example with $p = 2$, where the M2L operator is defined as

$$\bar{\mathbf{K}} = \left[\begin{array}{cc|cc} a & b & d & e \\ c & a & f & d \\ \hline g & h & a & b \\ i & g & c & a \end{array}\right]. \tag{5.5}$$

Then, the block and fully circulant embedding matrices are respectively defined as

$$\bar{\mathbf{E}} = \left[\begin{array}{ccc|ccc|ccc} a & b & c & d & e & f & g & h & i \\ c & a & b & f & d & e & i & g & i \\ b & c & a & e & f & d & h & i & g \\ \hline g & h & i & a & b & c & d & e & f \\ i & g & i & c & a & b & f & d & e \\ h & i & g & b & c & a & e & f & d \\ \hline d & e & f & g & h & i & a & b & c \\ f & d & e & i & g & i & c & a & b \\ e & f & d & h & i & g & b & c & a \end{array}\right] \quad \text{and} \quad \bar{\mathbf{E}}' = \left[\begin{array}{ccc|ccc|ccc} a & b & f & d & e & i & g & h & c \\ c & a & b & f & d & e & i & g & h \\ h & c & a & b & f & d & e & i & g \\ \hline g & h & c & a & b & f & d & e & i \\ i & g & h & c & a & b & f & d & e \\ e & i & g & h & c & a & b & f & d \\ \hline d & e & i & g & h & c & a & b & f \\ f & d & e & i & g & h & c & a & b \\ b & f & d & e & i & g & h & c & a \end{array}\right]. \tag{5.6}$$

Therefore, it is possible to use the 1D FFT to compute the DFT of the M2L operators. As a result, we can benefit from better performance of the FFT and also further reduce the size of the transformed arrays. In fact, instead of packing in the last dimension, we

[*]The first value is real and the rest is complex. Moreover, the second half of the rest is the conjugate of the first half.

pack the entire complex arrays. More precisely, after transformation the M2L operators are stored as complex valued rows of size $((2p+1)^3-1)/2+1 = 4p^3+6p^2+3p+1$ instead of $(2p+1)\times(2p+1)\times(p+1) = 4p^3+8p^2+5p+1$, namely $2p(p+1)$ extra entries. On the other hand, the *bbfmm* stores M2L operators as real valued matrices of size $(p+1)^3\times(p+1)^3$, *i.e.*, $(p+1)^6$ real entries.

5.2 A new block low-rank algorithm for smooth kernels, the *smooth-ufmm*

The FMM introduced by Greengard et al. [59] was originally developed for kernels with a strong singularity at the origin, *e.g.*, $k(\mathbf{x},\mathbf{y}) = 1/|\mathbf{x}-\mathbf{y}|$. Moreover, it relied on spherical harmonics expansions, thus requiring $\mathbf{x}$ and $\mathbf{y}$ to be sufficiently far appart. Therefore, the algorithm was designed to always ensure well-separation ($\gamma = 2$). For globally smooth kernels, *e.g.*, the Gaussian kernel

$$k(r) = k_\infty(r) = e^{-r^2/(2\ell^2)},\ \forall \ell \in \mathbb{R} \tag{5.7}$$

involved in the application of Chapter 7, we show that this condition can be relaxed in order to define a new efficient block low-rank algorithm.

5.2.1 Extension of the interaction list

Provided the kernel is *sufficiently* smooth near the origin we want to consider a summation scheme where the nearfield interactions are approximated as well. An algorithm of this nature can be derived from the *ufmm* algorithm by simply changing the admissibility criterion $\gamma^{(\bar{L})}$ at the leaf level from 2 to 0. While in the *ufmm* well-separation of leaves ($\gamma^{(\bar{L})} = 2$) is imposed, in the new variant coïnciding leaves form admissible pairs as well ($\gamma^{(\bar{L})} = 0$). In fact, since the kernel is *sufficiently* smooth at the origin, the diagonal subblocks of the kernel matrix associated with coïncident and adjacent cell pairs are *relatively* low-rank. This variant is denoted *smooth-ufmm* and can be seen as a block low-rank approximation technique.

Figure 5.5 illustrates the difference between both variants in term of interaction list. In the *smooth* variant the interaction list at the leaf level $\mathcal{I}(\mathcal{C}_x^{\bar{L}})$ includes all leaf cells whose parent is a direct neighbor of the target parent cell $\mathcal{C}_x^{\bar{L}-1}$ and consequently $\mathcal{N}(\mathcal{C}_x^{\bar{L}}) = \varnothing$, whereas in the standard variant the direct neighbors of the target leaf cell form the nearfield. The interaction list of a given leaf now includes the direct neighbors and the leaf itself, which means that all interactions will be transferred by means of M2L operations and thus no interaction needs to be computed at the P2P step.

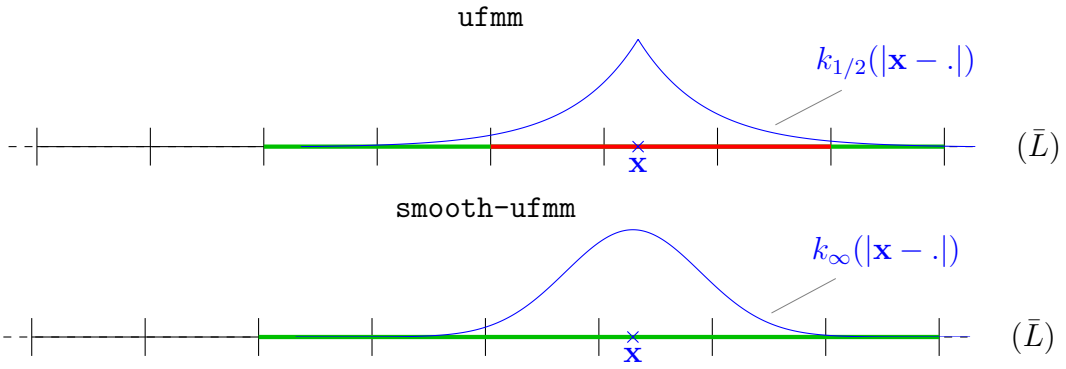

Figure 5.5: Illustration of both *ufmm* variants on a 1D tree. The target particle $\mathbf{x}$ (blue) lies in the leaf cell $\mathcal{C}_{\mathbf{x}}^{\bar{L}}$. The interaction list $\mathcal{I}(\mathcal{C}_{\mathbf{x}}^{L})$ at a given level L is represented in green, while the nearfield $\mathcal{N}(\mathcal{C}_{\mathbf{x}}^{L})$ is represented in red.

5.2.2 Optimal setup

Since the P2P and the M2L do not compete anymore in the *smooth-ufmm*, these steps do not have to be balanced. Consequently, the concept of level is not crucial anymore but it remains convenient, and the actual tuning parameter becomes the width of the leaf cells. However, we still need to minimize the cost of the M2L by using the fewest number of clusters, *i.e.*, the largest leaf cells. Therefore, our algorithm remains a multi-level scheme, where the depth for the octree is used to control the width of the leaves and thus the overall cost of the algorithm. The optimal setup for the algorithm is the lowest tree depth, that leads to a similar accuracy as the original *ufmm* scheme.

5.3 Comparative cost analysis

5.3.1 *ufmm* versus *bbfmm*

The theoretical complexities of the *bbfmm* and *ufmm* algorithms are given in terms of computational cost in Table 5.1 and memory footprint in Table 5.2. In the *ufmm* the precomputation of the M2L operators requires $\mathcal{O}(p^3 \log p)$ operations (*i.e.*, the cost of the FFT) while their application requires only $\mathcal{O}(p^3)$ operations (*i.e.*, the cost of an entrywise product). All algorithms scale linearly in n but they optimize the M2L step differently. In fact, the cost of storing and applying M2L operators scales like $\mathcal{O}(p^3)$ in the *ufmm* and $\mathcal{O}(p^6)$ in the *bbfmm*, therefore we expect significant differences in memory requirements and computational times.

5.3.2 *smooth-ufmm* versus *ufmm*

Due to the extension of the interaction list in the *smooth-ufmm*, the maximum number of M2L operators to store and apply at the leaf level increases from 189 to $189 + 27 = 216$ in 3D. However, as shown in the numerical benchmarks presented in Section 5.4, for a given accuracy the optimal number of level is usually slightly lower than the level required for the standard *ufmm*. Moreover, since low-rank representations replace the direct computations,

for sufficiently large n the overall computational time of the *smooth-ufmm* is expected to be lower than that of the *ufmm*.

	bbfmm	*ufmm*	*smooth-ufmm*
P2P	$n_0 \times n$	$n_0 \times n$	0
P2M/L2P	$p^3 \times n$	$p^3 \times n + p^3 \log(p) \times n/n_0$	$p^3 \times n + p^3 \log(p) \times n/n_0$
M2M/L2L	p^4	$p^4 + p^3 \log(p)$	$p^4 + p^3 \log(p)$
build M2L	$316p^6$	$316p^3 \log(p)$	$343p^3 \log(p)$
apply M2L	$189p^6$	$189p^3$	$216p^3$

Table 5.1: Asymptotic complexities of the *ufmm* variants compared to the *bbfmm*. The complexities of the M2L and M2M/L2L are given per level and per non empty cell, they should be multiplied by the number of non-empty cells (max. 8^L) and then summed over levels. For the P2P and P2M/L2P the complexities are given globally. The constant $n_0 = n/8^{\bar{L}}$ denotes the average number of particles per leaf.

Steps	Operators	*bbfmm*	*ufmm*	*smooth-ufmm*
M2L	$\bar{\mathbf{K}}$	$316 \times (p+1)^6$	$316 \times (4p^3 + 6p^2 + 3p + 1)$	$343 \times (4p^3 + 6p^2 + 3p + 1)$
P2M/L2P	$\boldsymbol{\mathcal{M}}/\boldsymbol{\mathcal{L}}$	$n/n_0 \times (p+1)^3$	$n/n_0 \times (p+1)^3$	$n/n_0 \times (p+1)^3$
P2M/L2P	$\tilde{\boldsymbol{\mathcal{M}}}/\tilde{\boldsymbol{\mathcal{L}}}$	0	$n/n_0 \times (4p^3 + 6p^2 + 3p + 1) \times 2$	$n/n_0 \times (4p^3 + 6p^2 + 3p + 1) \times 2$
P2P*	$\mathbf{K}$	$n_0 \times n$	$n_0 \times n$	0

Table 5.2: Memory footprint at various steps of the *ufmm*, *smooth-ufmm* and *bbfmm* expressed in terms of the number of entries. We obtain the actual number of Bytes by multiplying the number of entries by the size of the value type, *i.e.*, 8Bytes for real valued entries in double precision arithmetics. The constant $n_0 = n/8^{\bar{L}}$ denotes the average number of particles per leaf in a uniform distribution of particles.

5.3.3 Theoretical memory requirements

Although the M2L step is usually the most computationally intensive step of the FMM, because it involves a total of $2^{d\bar{L}} \times (6^d - 3^d)$ multipole to local transfers of expansions, where d is the ambiant dimension, only $\bar{L} \times (7^d - 3^d)$ M2L operators have to be stored. On the other hand, the storage of the $2^{d\bar{L}}$ expansions is usually the limitant factor in term of memory consumption. Figure 5.6 representS the memory footprint of the *ufmm* and the *bbfmm* *w.r.t.* the interpolation order p for growing values of $\bar{L} = 3, 5, 7$. First of all, for large tree depths, *e.g.*, $\bar{L} \geq 7$, the amount of memory required for the storage of the expansions in both algorithms dominates the overall cost. Most importantly, it even exceeds the ressource available on standard computers for intermediate interpolation orders, namely approximately 50GB for the *ufmm* and 100GB for the *bbfmm* with $p = 7$. In such regime, usually associated with distributions of several million particles, a distributed memory version of the algorithm is usually prefered. For lower tree depths, *e.g.*, $\bar{L} \leq 5$, the M2L of the *bbfmm* dominates the overall cost due to its $\mathcal{O}(p^6)$ complexity. On the other hand, in the *ufmm* the storage of the expansions rapidely dominates the overall memory footprint, namely for $\bar{L} \geq 5$.

*The P2P operators are precomputed only if matrix to matrix products are considered (Section 7.2). In that case, the *smooth-ufmm* requires a significantly lower amount of memory than the *ufmm*.

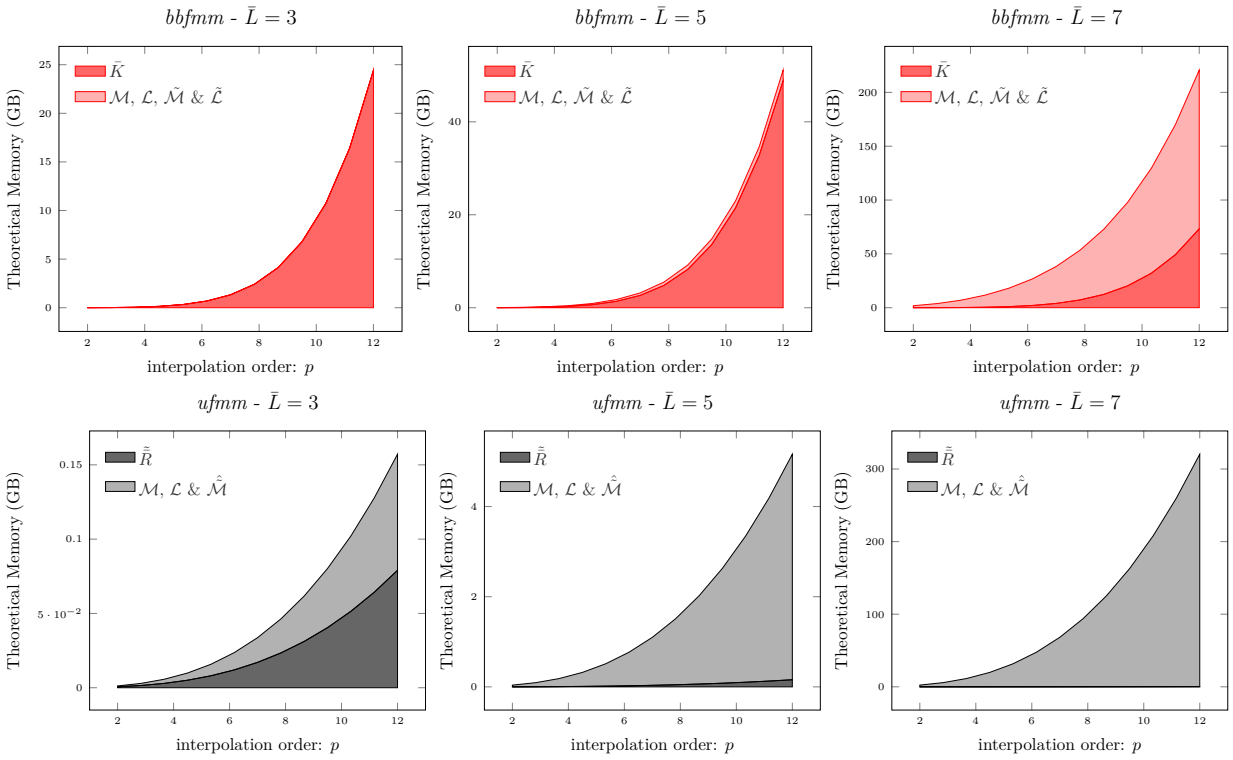

Figure 5.6: Theoretical memory requirements of the interpolation based FMM in the worst case scenario, namely no empty cell. These values should be qualified by the fact that for heterogeneous geometries the number of non-empty cells can be very low. For instance, as shown on Figure B.2 for $n = 10^6$ particles, the memory required for storing the expansions at level $\bar{L} = 7$ could be divided by 100 for a unit sphere and 1000 for a prolate sphere.

5.4 Numerical benchmarks

In order to compare the efficiency of our new algorithms to existing variants of the *bbfmm*, we will consider artificial numerical benchmarks. The distributions of particles considered are presented in Section 5.4.1, they involve different filling densities in the octree. Our comparative analysis is organized as follows:

- We first discuss statistics on the filling of the octree for various geometries.
- Then, we compare the relative accuracy and numerical performance of the *bbfmm* and the *ufmm* using the Laplacian kernel.
- Finally, we compare the relative accuracy and numerical performance of the *ufmm* and the *smooth-ufmm* using a smooth kernel, namely the Gaussian kernel.

5.4.1 Distributions of points and kernels of interaction.

Distributions Let us consider 3 different geometries for the distribution of particles:

- The *unitCube* is a volumic distribution, in which the particles are distributed uniformly inside a $2 \times 2 \times 2$ cube.
- The *unitSphere* is a surfacic distribution, in which the particles are distributed uniformly on a unit sphere.

- The *prolateSphere* is a surfacic distribution, in which the particles are distributed uniformly on a $0.2 \times 0.2 \times 2$ ellipsoid, *a.k.a.* prolate sphere with ratio 10.

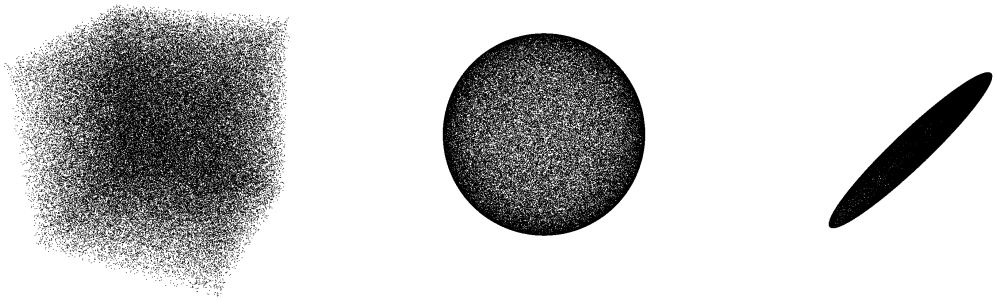

Figure 5.7: Uniform distribution of particles in the unit cube (left), on the unit sphere (center) and on a prolate sphere (right) with proportions $(0.1, 0.1, 1.0)$.

Figure 5.7 shows examples of such particle distributions for $n = 10^5$ particles. All theses geometries share the same bounding box, *i.e.*, the $2 \times 2 \times 2$ box, however they fill the octree differently. More precisely, the *unitCube* has a fully populated octree, while the *unitSphere* and the *prolateSphere* have respectively an intermediate and a low number of non-empty cells. For advanced statistics on the filling of the octree and on the number of nearfield and farfield operators please refer to Appendix B.

Kernels In our numerical benchmarks, we will consider various kinds of interactions characterized by the different kernels represented on Figure 5.8. First of all, we will validate the accuracy and analyze the performance of our algorithms on a standard homogeneous kernel, namely the Laplacian kernel

$$k(r) = 1/r, \tag{5.8}$$

and then on a smooth and non-homogeneous kernel, namely the Gaussian kernel

$$k(r) = e^{-\frac{r^2}{2\ell^2}}, \text{ with } \ell \in \mathbb{R}. \tag{5.9}$$

Tensorial variants of the Laplacian will be used in Chapter 6, while variants of the Gaussian will be used in Chapter 7. Before starting any performance analysis, the accuracy of the algorithm should always be validated by plotting the error ε_{fmm} on the potential *w.r.t.* interpolation order p.

5.4.2 *bbfmm* vs. *ufmm* on the Laplacian kernel

In this subsection we discuss the relative accuracy and sequential performance of the *ufmm* and optimized variants of the *bbfmm* using the standard Laplacian kernel (5.8).

Accuracy The convergence of the *ufmm* and *smooth-ufmm* is represented on Figure 5.9 (left) in term of the relative 2-norm of the error on the potential. As expected all variants of

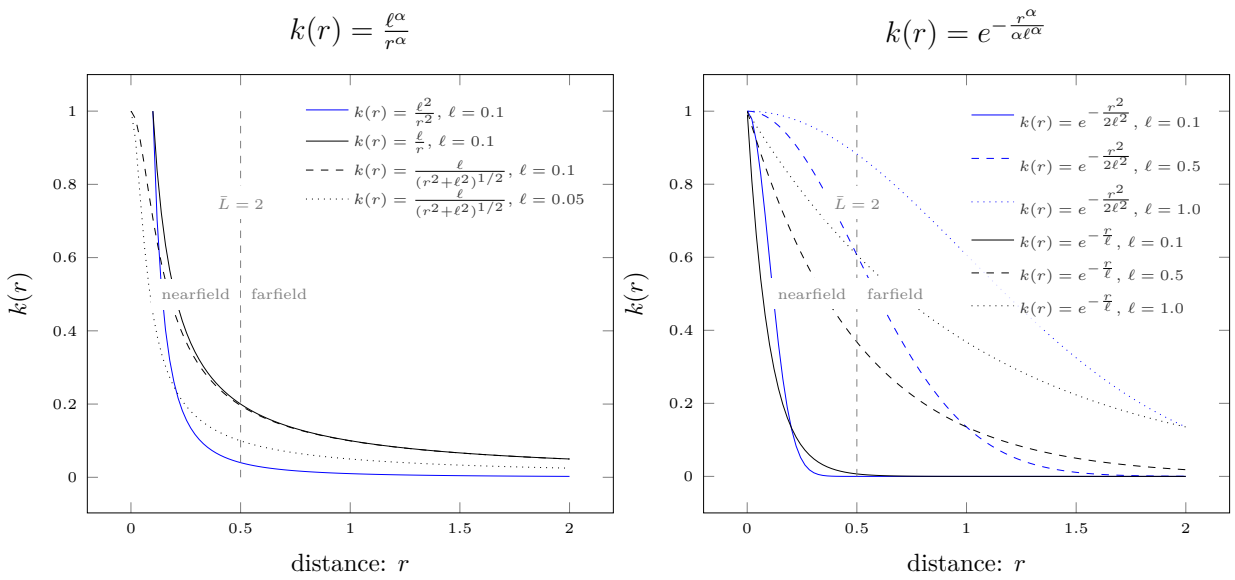

Figure 5.8: Shape of the kernel *w.r.t.* the distance r if the width of the root bounding box equals 2. The limit between nearfield and farfield is represented for $\bar{L} = 2$. Left: Negative powers of r and some regularized variants. Right: Exponential and Gaussian kernels represent the extreme cases of smoothness in the family of Matérn functions.

the *bbfmm* have the same accuracy and convergence rate. Moreover, the *ufmm* is slightly less accurate than the *bbfmm* due to the near minimax property of the Chebyshev interpolation. Nevertheless, we will show later on that it still performs better for a fixed accuracy.

Sequential performance The relative computational costs of the *bbfmm*, with or without compression of the `M2L` operators (see [87]), and the *ufmm* are represented on Fig. 5.10 using a fixed number of particles $n = 20.000$ distributed in the unit cube with $\bar{L} = 3$ (*i.e.*, $n_0 \approx 30$). The *ufmm* outperforms the unoptimized *bbfmm* in terms of both computational time and memory requirements. Let us recall that the *symmetric-bbfmm* is taylored for symmetric kernels allowing for a massive reduction of the interaction list and that it involves individual compression of the `M2L` operators, while the *compressed-bbfmm* only involves a global compression of the `M2L` operators. As shown on the graphs, the *ufmm* and the *symmetric-bbfmm* have similar performance though the *ufmm* applies to any kernel. More precisely, the *ufmm* is faster in term of computational time but requires a little more memory than the *symmetric-bbfmm*.

Detailled timings Figure 5.11 shows the detailled running times of each step of the *ufmm* and the *symmetric-bbfmm* for fixed tree depth $\bar{L} = 3$ *w.r.t.* the interpolation order. Consequently, we observe how the cost of the farfield as the accuracy grows, while the cost of the nearfield is fixed. The graphs also illustrate how reducing the cost of the `M2L` affects the overall application time in the *ufmm*. Finally, it shows that the other steps (`P2M`/`L2P` and `M2M`/`L2L`) are more expensive in the *ufmm* than in the *bbfmm* if the interpolation order is large, namely $p > 7$, but cheaper in the opposite case.

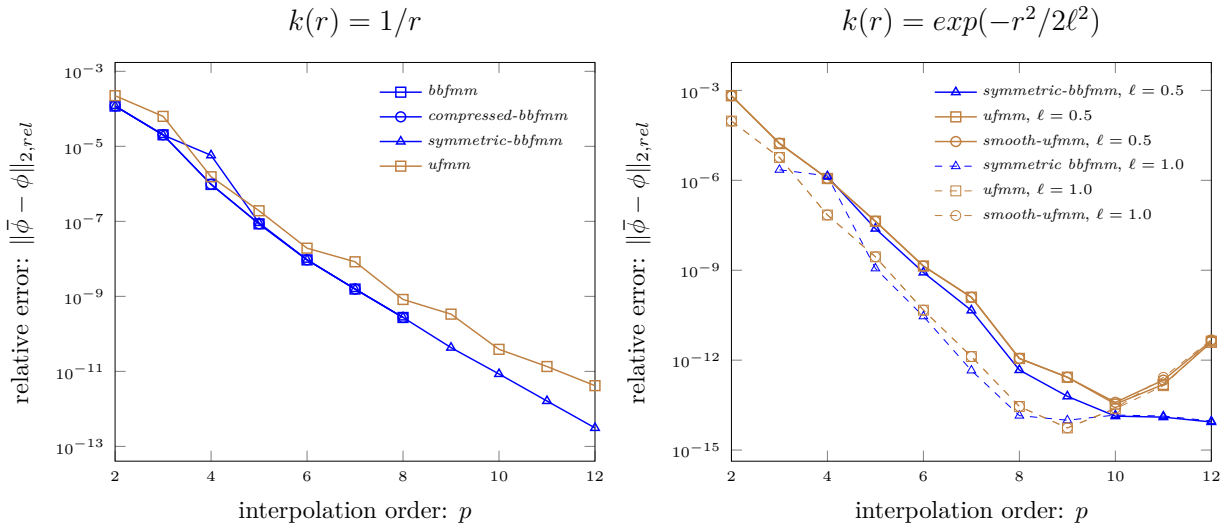

Figure 5.9: Experimental accuracy of the *ufmm* and variants of the *bbfmm* *w.r.t.* the interpolation order p. We considered a uniform distribution of 20.000 particles inside the unit cube with $\bar{L} = 3$. Left: Homogeneous Laplacian kernel $k(r) = 1/r$. Right: Non-homogeneous Gaussian kernel $k_\infty(r, \ell)$ with $\ell \in \{0.5, 1.0\}$. The computation is done on simple desktop computer.

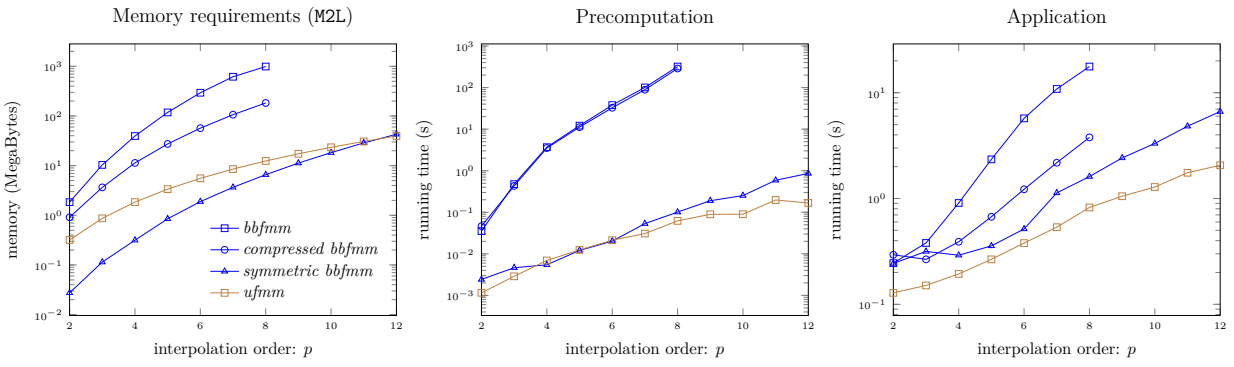

Figure 5.10: Experimental memory requirements and computational time of the *ufmm* and variants of the *bbfmm* *w.r.t.* the interpolation order p. We use the homogeneous Laplacian kernel $k(r) = 1/r$ on a uniform distribution of 20.000 particles inside the unit cube with $\bar{L} = 3$. The computation is done on simple desktop computer.

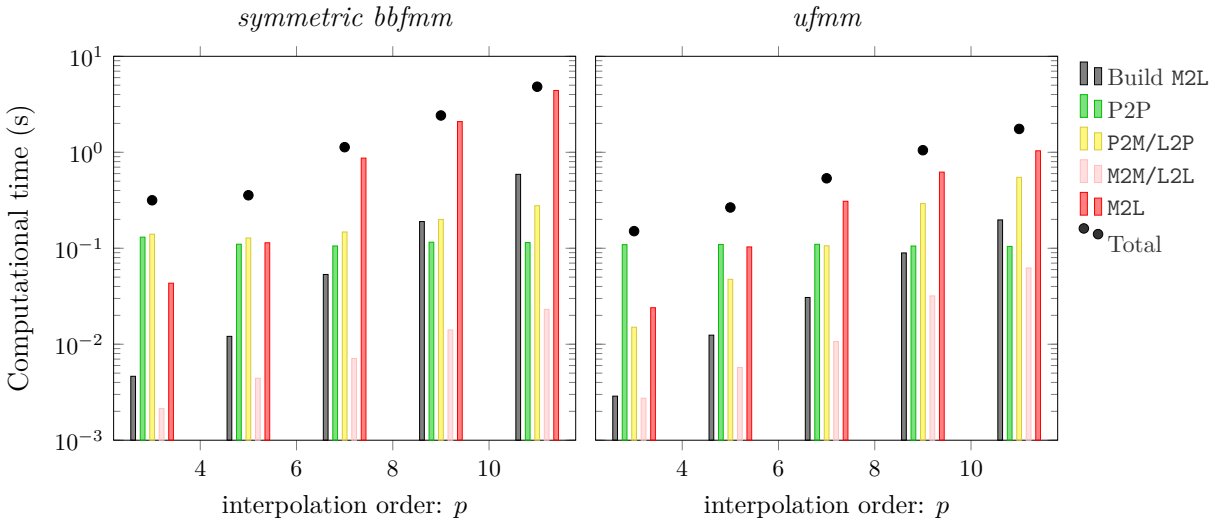

Figure 5.11: Detailled computational time for the *symmetric-bbfmm* (left) or the *ufmm* (right) *w.r.t.* the interpolation order p. We used $n = 20.000$ particles distributed uniformly in the unit cube with $\bar{L} = 3$.

Performance at fixed accuracy For kernels such as the Laplacian, we may need to increase the interpolation order of the *ufmm* by 1 in order to get the same accuracy as the *bbfmm* (Figure 5.9), in particular for accuracy below 10^{-8}, *i.e.*, $p \geq 7$. In any cases the *ufmm* beats all *bbfmm* variants by a factor of at least 2 (for $p = 7$), that grows with the interpolation order.

5.4.3 *ufmm* vs. *smooth-ufmm* on Gaussian kernels

In this subsection we discuss the relative accuracy and sequential performance of the *ufmm* and the *smooth-ufmm* using Gaussian kernels (5.9) with length scales $\ell \in \{0.5, 1.0\}$.

Accuracy The convergence of the *ufmm* and *smooth-ufmm* is represented on Figure 5.9 (right) in term of the relative 2-norm of the error on the potential. In particular, we observe that the larger ℓ the faster the convergence and the lower the error, which directly results from the smoothness of the function when ℓ grows. For large values of p, this fast convergence allows us to witness another phenomenon. In fact, for any ℓ once the machine accuracy is reached the interpolation scheme may become instable. However, most application require precisions that are far lower than the machine accuracy.

Performance at fixed size The relative computational costs of the *ufmm* and the *smooth-ufmm* are represented on Figure 5.12 using a fixed number of particles $n = 20.000$ and $\bar{L} = 3$, *i.e.*, $n_0 \approx 30$. First of all, the extra memory required for storing the extra `M2L` operators of the *smooth-ufmm* is negligible compared to the memory required by the storage of the `M2L` operators of the *ufmm*. Second of all, the application time grows faster for the *ufmm* than for the *smooth-ufmm* as the interpolation order grows. As a result, the application time is 10 times lower for the *smooth-ufmm* at $p = 2$, while it is equal to the application time of the

ufmm for $p = 9$. Finally, in this configuration, the *ufmm* and *smooth-ufmm* exhibit rather similar performance, which may not be the case in practice. In fact, in practice the optimal tree depth may be lower for the *smooth-ufmm*, therefore its computational cost may be far lower.

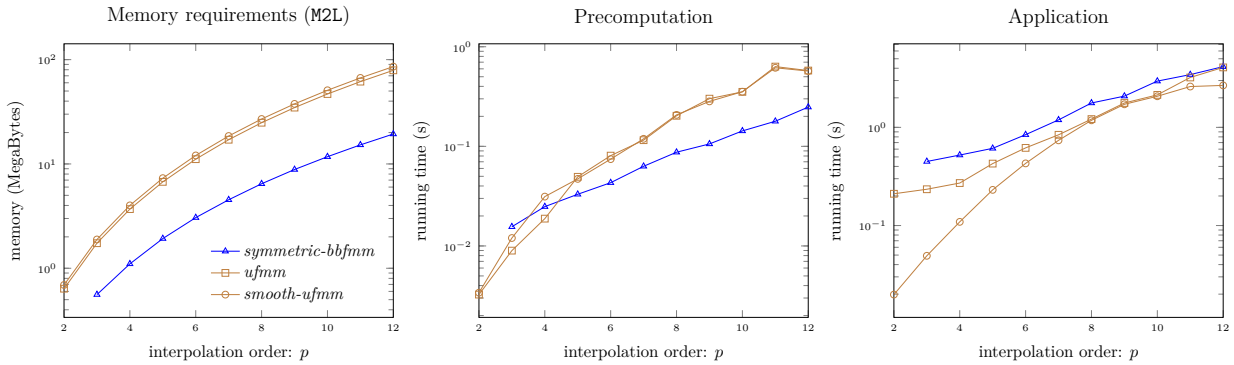

Figure 5.12: Experimental memory requirements and computational time of the *ufmm* and *smooth-ufmm* *w.r.t.* the interpolation order p. We use a smooth and non-homogeneous Gaussian kernel $k_\infty(r, \ell)$ with $\ell = 0.5$ and a uniform distribution of 20.000 particles inside the unit cube with $\bar{L} = 3$.

Performance at fixed accuracy In order to improve our comparison, we analyze the relative performance of the *ufmm* and the *smooth-ufmm* for a given accuracy of about 10^{-5}. Figure 7.10 shows the application times for 3 different geometries for n up to 10^6 particles. The *smooth-ufmm* unsurprisingly has better performance than the *ufmm*, since the cost of approximating the nearfield becomes cheaper as n increases. Furthermore, it beats the *ufmm* by a factor of 10 for sufficiently large n, while the *ufmm* only beats the *bbfmm* by a factor of 2.

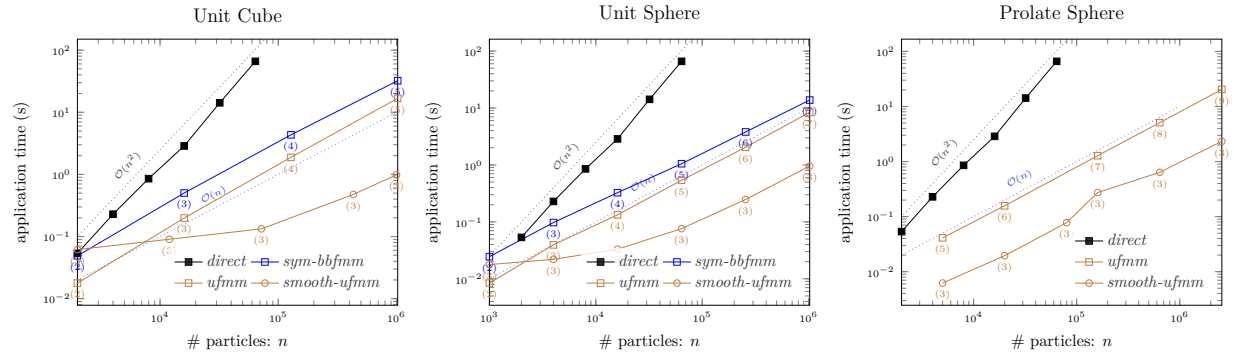

Figure 5.13: Running time (s) *w.r.t.* the number of particles n for a target accuracy of about 10^{-5}. In order to compare all algorithms we use a smooth kernel, namely the Gaussian kernel with $\ell = 0.5$. Particles are distributed uniformly in the unit cube (left), on the unit sphere (center) and on the prolate sphere (right).

5.5 Conclusion

We introduced a new interpolation based FMM with efficient formulation of the M2L step in Fourier domain. Please refer to our research report [15] for a shorter introduction. The

performance and accuracy of the algorithm were tested on artificial benchmarks illustrating the significant benefits of using the *ufmm* over the *bbfmm* and its optimized variants. Moreover, a new algorithm called the *smooth-ufmm* was designed for handling smooth kernels more efficiently. This variant differs substantially from the original FMM in the sense that it implements a different admissibility criterion but it does not depend on the interpolation procedure, therefore it could as well be derived from the *bbfmm*. In the next two chapters, the applications of the *ufmm* and the *smooth-ufmm* to highly expensive simulations will be considered, namely tensorial interactions (Chapter 6) and matrix-to-matrix multiplications (Chapter 7).

6

Fast Multipole DD simulations

Contents

In this chapter, we present the interpolation formula of tensorial interaction kernels involved in elasticity problems such as DD simulations. Then, we introduce various Fast Multipole Method (FMM) summation schemes for the fast evaluation the isotropic elastic stress, forces and energy. We start by simple and naïve schemes and then we move on to more evolved `M2L`-optimized schemes. Finally, we evaluate the performance of these schemes on numerical benchmarks.

6.1 Fast Multipole computation of the isotropic stress and forces

6.1.1 Introduction

Consistency First of all, we would like to recall that a dislocation loop may cross the edges of the octree, thus part of its segments can contribute to the nearfield interactions while the other segments contribute to the farfield interactions. Since all existing expressions of the elastic stress and energy are only equivalent on a *closed* dislocation loop, we need to use the same formula for the evaluation of the farfield and for the nearfield, namely (2.4) for the stress and (2.8) for the energy. Therefore, the choice of well-suited expressions is crucial in an efficient implementation and will be discussed here in particular for the evaluation of the energy.

Balance For various computational reasons related to the velocity of the dislocations, the width of the leaf cells cannot be smaller than a few dislocation lengths, namely $\omega_{\bar{L}} = 3L_{max}$. Consequently, the DD codes implementing the FMM usually impose a maximum depth for the octree. For instance, if $L_{max} = 100$ with a root bounding box of width $\omega_0 = 20000$ then $\bar{L} \leq 6$. However, the number of nearfield interactions can grow dramatically if the number of segment increases while the depth remains constant. As a result, the balance between nearfield and farfield becomes impossible to maintain. Therefore, the efficient implementation of the nearfield requires a lot of care.

6.1.2 Interpolation of the elastic stress

Let us first define the tensorial interaction kernels $\mathbf{k}(\mathbf{x}, \mathbf{y})$ as

$$k_i(\mathbf{x}, \mathbf{y}) = r_{a,i}, \quad k_{ij}(\mathbf{x}, \mathbf{y}) = r_{a,ij} \quad \text{and} \quad k_{ijk}(\mathbf{x}, \mathbf{y}) = r_{a,ijk}, \tag{6.1}$$

where $r_a = \sqrt{a^2 + r^2}$, $r = |\mathbf{r}| = \sqrt{r_i r_i}$ and $\mathbf{r} = \mathbf{x} - \mathbf{y}$. The notation $r_{a,i}$ denotes the partial derivatives of r_a *w.r.t.* x_i, *i.e.*,

$$r_{a,i} = \frac{\partial r_a}{\partial x_i}.$$

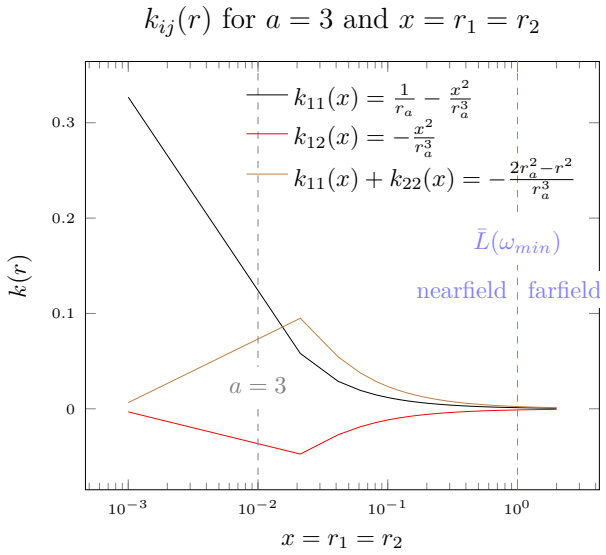

Figure 6.1: Representation of the components k_{11}, k_{12} and $k_{11} + k_{22}$ in 2D, *i.e.*, $r = \sqrt{r_1^2 + r_2^2}$, for $r_1 = r_2 \in [0, 2]$. Values are plotted *w.r.t.* the scaled distance $x = r/\omega_{min}$ for $a = 3$ and a typical segment length $L_{max} = 90$, *i.e.*, $a = L_{max}/30$. We consider the critical case, where the leaf cell width equals $\omega_{min} = 3L_{max}$.

Hence, the analytical expressions associated with the 3 variants of $\mathbf{k}(\mathbf{x}, \mathbf{y})$ presented above read as

$$k_i = \frac{r_i}{r_a}, \quad k_{ij} = \frac{\delta_{ij}}{r_a} - \frac{r_i r_j}{r_a^3} \quad \text{and} \quad k_{ijk} = -\frac{\delta_{jk} r_i + \delta_{ik} r_j + \delta_{ij} r_k}{r_a^3} + 3\frac{r_i r_j r_k}{r_a^5}. \tag{6.2}$$

The graphs on Figure 6.1 represent certain components of the 2^{nd}-order interaction tensor in 2D for $a = 3$ on a representative area, namely a few minimum leaf cell widths, *w.r.t.* the scaled distance. It shows the maximum variations, that can exhibit the kernel in the nearfield. In particular, we observe that the kernels decrease fast within the first few core widths a and become rather smooth in the farfield, which makes interpolation based FMM well-suited. Although the non-singular kernel is smooth by definition, it still exhibits brutal variations within the nearfield, just like the Gaussian kernel for $\ell = 0.1$. Therefore, it cannot be approximated in the nearfield using the *smooth-ufmm*. Using the compact notations defined in (1.15), we denote $\tilde{\mathbf{k}}$ the interpolant of $\mathbf{k}$ of order p such that

$$\tilde{\mathbf{k}}(\mathbf{x}, \mathbf{y}) = \sum_{|\boldsymbol{\alpha}| \leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) \sum_{|\boldsymbol{\beta}| \leq p} \bar{\mathbf{K}}_{\boldsymbol{\alpha}\boldsymbol{\beta}} S_{\boldsymbol{\beta}}(\mathbf{y}), \ \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^3. \tag{6.3}$$

If we replace the expression (6.3) of the interpolant $\tilde{\mathbf{k}}$ in (2.4), we get an approximation of the elastic stress field $\boldsymbol{\sigma}$, that reads as

$$\sigma_{ij}(x) \approx \frac{\mu}{8\pi}(\tilde{\sigma}_{ij}^{A'} + \frac{2}{1-\nu}\tilde{\sigma}_{ij}^{B'}), \ \forall (i,j) \in \{1,2,3\}^2, \tag{6.4}$$

where for all $i, j \in \{1, 2, 3\}$

$$\tilde{\sigma}_{ij}^{A'} = \tilde{\sigma}_{ij}^{A} + \tilde{\sigma}_{ji}^{A}$$
$$\tilde{\sigma}_{ij}^{B'} = \tilde{\sigma}_{ij}^{B} - \delta_{ij}\tilde{\sigma}_{\ell\ell}^{B}$$

and

$$\tilde{\sigma}_{ij}^{A}(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) \sum_{|\boldsymbol{\beta}|\leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{m\ell\ell} \varepsilon_{jmk} \oint_{(\mathcal{C}')} S_{\boldsymbol{\beta}}(\mathbf{x}') b'_k t'_i dx' \tag{6.5}$$

$$\tilde{\sigma}_{ij}^{B}(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) \sum_{|\boldsymbol{\beta}|\leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{mij} \varepsilon_{nmk} \oint_{(\mathcal{C}')} S_{\boldsymbol{\beta}}(\mathbf{x}') b'_k t'_n dx' \tag{6.6}$$

Since $(\mathcal{C}')$ is a dislocation network discretized by the reunion of segments $\mathcal{S}_s = [\mathbf{x}_s^1 \mathbf{x}_s^2]$, *i.e.*, $(\mathcal{C}') = \cup_{s=1}^{n} \mathcal{S}_s$, the contributions $\tilde{\boldsymbol{\sigma}}^{\mathbf{A}}$ and $\tilde{\boldsymbol{\sigma}}^{\mathbf{B}}$ can also be expressed as the sum of the contributions of each segment, namely

$$\tilde{\sigma}_{ij}^{A}(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) \sum_{|\boldsymbol{\beta}|\leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{m\ell\ell} \varepsilon_{jmk} \sum_{[\mathbf{x}^1\mathbf{x}^2]\in(\mathcal{C}')} b'_k t'_i \int_{\mathbf{x}^1}^{\mathbf{x}^2} S_{\boldsymbol{\beta}}(\mathbf{x}') dx' \tag{6.7}$$

$$\tilde{\sigma}_{ij}^{B}(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) \sum_{|\boldsymbol{\beta}|\leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{mij} \varepsilon_{nmk} \sum_{[\mathbf{x}^1\mathbf{x}^2]\in(\mathcal{C}')} b'_k t'_n \int_{\mathbf{x}^1}^{\mathbf{x}^2} S_{\boldsymbol{\beta}}(\mathbf{x}') dx' \tag{6.8}$$

6.1.3 A naïve Fast Multipole summation scheme

Equations (6.7) and (6.8) can be computed using a fast multipole summation scheme, that consists in the following steps:

- For $\tilde{\boldsymbol{\sigma}}^{\mathbf{A}}$, perform successively

$$(\mathcal{M}_{\boldsymbol{\beta}}^{A})_{mij} = \varepsilon_{jmk} \sum_{[\mathbf{x}^1\mathbf{x}^2]\in(\mathcal{C}')} b'_k t'_i \int_{\mathbf{x}^1}^{\mathbf{x}^2} S_{\boldsymbol{\beta}}(\mathbf{x}') dx' \tag{P2M-SA}$$

$$(\mathcal{L}_{\boldsymbol{\alpha}}^{A})_{ij} = \sum_{|\boldsymbol{\beta}|\leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{m\ell\ell} (\mathcal{M}_{\boldsymbol{\beta}}^{A})_{mij} \tag{M2L-SA}$$

$$\tilde{\sigma}_{ij}^{A}(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) (\mathcal{L}_{\boldsymbol{\alpha}}^{A})_{ij} \tag{L2P-SA}$$

- For $\tilde{\boldsymbol{\sigma}}^{\mathbf{B}}$, perform successively

$$(\mathcal{M}_{\boldsymbol{\beta}}^{B})_{m} = (\mathcal{M}_{\boldsymbol{\beta}}^{A})_{mnn} \tag{P2M-SB}$$

$$(\mathcal{L}_{\boldsymbol{\alpha}}^{B})_{ij} = \sum_{|\boldsymbol{\beta}|\leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{mij} (\mathcal{M}_{\boldsymbol{\beta}}^{B})_{m} \tag{M2L-SB}$$

$$\tilde{\sigma}_{ij}^{B}(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) (\mathcal{L}_{\boldsymbol{\alpha}}^{B})_{ij} \tag{L2P-SB}$$

- Finally, compute $\tilde{\boldsymbol{\sigma}}$ using (6.4).

First of all, we notice that the multipole expansions $(\boldsymbol{\mathcal{M}}^B)_m$ for $m \in \{1,2,3\}$ are the traces of $\left\{(\boldsymbol{\mathcal{M}}^A)_{mij}\right\}_{i,j\leq 3}$, therefore they can be computed at almost no extra cost. Furthermore, the M2L operators used for $\tilde{\boldsymbol{\sigma}}^{\mathbf{A}}$ are the traces of those used for $\tilde{\boldsymbol{\sigma}}^{\mathbf{B}}$, more precisely

$$\left(\bar{K}_{\boldsymbol{\alpha\beta}}\right)_{m\ell\ell} = tr\left(\left\{\left(\bar{K}_{\boldsymbol{\alpha\beta}}\right)_{mij}\right\}_{ij}\right)$$

Finally, both terms $\tilde{\boldsymbol{\sigma}}^X$ for $X \in \{A,B\}$ share the same L2P step, *i.e.*,

$$\tilde{\boldsymbol{\sigma}}^X = \sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}(\mathbf{x})\boldsymbol{\mathcal{L}}^X_{\boldsymbol{\alpha}}. \tag{6.9}$$

Nodal forces The computation of the farfield contribution to the nodal forces presented in (2.6) can be computed using a different summation at the L2P step, namely

$$f_i^4 = \int_0^L \sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}^3 + \xi t)\varepsilon_{ijk}(\boldsymbol{\mathcal{L}}_{\boldsymbol{\alpha}})_{j\ell} b_\ell t_k \frac{\xi}{L} d\xi,\ \forall i \in \{1,2,3\} \tag{L2P-F4}$$

Cost estimation In order to perform this summation, we need to store 30 M2L components, namely 3 for $\tilde{\boldsymbol{\sigma}}^{\mathbf{A}}$ and 27 for $\tilde{\boldsymbol{\sigma}}^{\mathbf{B}}$. We need to store 27 components of $\boldsymbol{\mathcal{M}}^A_{\boldsymbol{\beta}}$ and 3 components of $\boldsymbol{\mathcal{M}}^B_{\boldsymbol{\beta}}$. Then, we need to perform 54 M2L operations in order to compute all 9 components of $\tilde{\boldsymbol{\sigma}}^{\mathbf{A}}$ and $\tilde{\boldsymbol{\sigma}}^{\mathbf{B}}$. Indeed, computing (M2L-SA) involves $3 \times 9 = 27$ M2L operations and computing (M2L-SB) involves $9 \times 3 = 27$ M2L operations.
The summation schemes proposed here is relatively naïve and expensive, therefore it will only be used as a reference for more efficient schemes.

6.2 Improved farfield computation of the stress

Here we introduce optimized variants of the previous summation scheme based on symmetries in Section 6.2.1, refactoring of the interactions in Section 6.2.2 and shifted derivatives in Section 6.2.3. Finally, we discuss the relative costs of each variant in Section 6.2.4.

6.2.1 Considering symmetries

Symmetries First of all, in the hypothesis of isotropic elasticity the stress field is symmetric, only 6 components out of 9 need to be computed. Furthermore, since in the expression of the kernel of interactions $\mathbf{k}$ the partial derivatives can be applied in an arbitrary order, then $\left(\bar{K}_{\boldsymbol{\alpha\beta}}\right)_{ijk}$ is symmetric *w.r.t.* i,j and k. Consequently, $\tilde{\boldsymbol{\sigma}}^{\mathbf{B}}$ is symmetric and only its upper triangular part needs to be assembled during the M2L step. On the other hand, it could be more convenient to assemble $\tilde{\boldsymbol{\sigma}}^{\mathbf{A'}}$ at the P2M step instead of the L2P, in order to store only symmetric quantities.

Fast Multipole summation scheme Hence, for all $(i,j) \in \{1,2,3\}^2$ with $i \leq j$ the optimized summation scheme can summarized as

- For $\tilde{\boldsymbol{\sigma}}^{\mathbf{A}'}$, perform successively

$$(\mathcal{M}_{\boldsymbol{\beta}}^{A'})_{mij} = (\mathcal{M}_{\boldsymbol{\beta}}^{A})_{mij} + (\mathcal{M}_{\boldsymbol{\beta}}^{A})_{mji} \qquad \texttt{(Symm-P2M-SA)}$$

$$(\mathcal{L}_{\boldsymbol{\alpha}}^{A'})_{ij} = \sum_{|\boldsymbol{\beta}|\leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{m\ell\ell} (\mathcal{M}_{\boldsymbol{\beta}}^{A'})_{mij} \qquad \texttt{(Symm-M2L-SA)}$$

$$\tilde{\sigma}_{ij}^{A'}(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) (\mathcal{L}_{\boldsymbol{\alpha}}^{A'})_{ij} \qquad \texttt{(Symm-L2P-SA)}$$

- For $\tilde{\boldsymbol{\sigma}}^{\mathbf{B}'}$, perform successively

$$(\mathcal{M}_{\boldsymbol{\beta}}^{B})_{m} = (\mathcal{M}_{\boldsymbol{\beta}}^{A})_{mnn} \qquad \texttt{(Symm-P2M-SB)}$$

$$(\mathcal{L}_{\boldsymbol{\alpha}}^{B})_{ij} = \sum_{|\boldsymbol{\beta}|\leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{mij} (\mathcal{M}_{\boldsymbol{\beta}}^{B})_{m} \qquad \texttt{(Symm-M2L-SB)}$$

$$\tilde{\sigma}_{ij}^{B'}(\mathbf{x}) = (\tilde{\sigma}^{B})_{ij} - \delta_{ij} (\tilde{\sigma}^{B})_{\ell\ell} \qquad \texttt{(Symm-L2P-SB)}$$

- Finally, compute $\tilde{\boldsymbol{\sigma}}$ using (6.4).

Cost estimation In order to perform this summation, we need to store 13 M2L components, namely 3 for $\tilde{\boldsymbol{\sigma}}^{\mathbf{A}}$ and 10 for $\tilde{\boldsymbol{\sigma}}^{\mathbf{B}}$. We need to store $3 \times 6 = 18$ components for $\boldsymbol{\mathcal{M}}^{A'}$ and only 3 for $\boldsymbol{\mathcal{M}}^{B}$. Then, we need to perform 36 M2L operations in order to compute 6 upper triangular components of $\tilde{\boldsymbol{\sigma}}^{\mathbf{A}'}$ and $\tilde{\boldsymbol{\sigma}}^{\mathbf{B}'}$. Indeed, (Symm-M2L-SA) involves $3 \times 6 = 18$ M2L operations and (Symm-M2L-SB) involves $6 \times 3 = 18$ M2L operations.

6.2.2 Refactoring interactions

The number of multipole and local expansions required by the two first schemes is relatively high, namely 39 for the `Direct` and 27 for the `Symmetric` scheme. As mentioned in Section 5.3.3, the amount of memory required to store 1 multipole expansion can quickly become prohibitive in interpolation based FMM. Therefore, we propose a new scheme based on expanding and refactoring the tensorial interactions and called the `Factorized` scheme, that simultaneously minimizes the number of M2L operations and the number of multipole expansions. For the sake of clarity, details of this scheme are presented in Appendix D.1.1. If the energy needs to be computed as well, this variant is more convenient than using symmetries of the stress, since it implements the same multipole expansions, namely

$$(\mathcal{M}_{\boldsymbol{\beta}})_{ij} = \sum_{[\mathbf{x}^1\mathbf{x}^2]\in(\mathcal{C}')} b'_i t'_j \int_{\mathbf{x}^1}^{\mathbf{x}^2} S_{\boldsymbol{\beta}}(\mathbf{x}') dx'. \qquad \texttt{(Facto-P2M)}$$

6.2.3 Shifting derivatives

Principle As mentioned in Section 5.1, the derivative of a function can be approximated by deriving the interpolated function. Thus, k_{ijk} can be approximated by deriving $\tilde{k}_{ij}$, the interpolant of k_{ij}. The associated interpolation formula reads as

$$k_{ijk}(\mathbf{x},\mathbf{y}) \approx \nabla_{x_k}\tilde{k}_{ij}(\mathbf{x},\mathbf{y}) = \sum_{|\boldsymbol{\alpha}|\leq p} \nabla_k S_{\boldsymbol{\alpha}}(\mathbf{x}) \sum_{|\boldsymbol{\beta}|\leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{ij} S_{\boldsymbol{\beta}}(\mathbf{y}) \tag{6.10}$$

Since S is a higher order polynomials than $\nabla_k S$ for $k \in \{1,2,3\}$, then equation (6.3) should provide a more accurate approximation of k_{ijk} than (6.10). On the other hand, (6.10) reduces the number of components to approximate from 10 for all k_{ijk} to 6 for all k_{ij}, while increasing the dimension of the interpolator by 3. Finally, if the derivation is done with respect to y, then it applies to the right interpolator. Moreover, since $k(\mathbf{x},\mathbf{y}) = k(\mathbf{x}-\mathbf{y})$,

$$\nabla_{y_k}\tilde{k}_{ij} = -\nabla_{x_k}\tilde{k}_{ij}$$

and thus

$$k_{ijk}(\mathbf{x},\mathbf{y}) \approx -\nabla_{y_k}\tilde{k}_{ij}(\mathbf{x},\mathbf{y}) = -\sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) \sum_{|\boldsymbol{\beta}|\leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{ij} \nabla_k S_{\boldsymbol{\beta}}(\mathbf{y})$$

In other terms, for such interpolation formula, shifting derivatives from right to left implies changing the sign in front of the formula.

Fast Multipole summation scheme Shifting the derivatives *w.r.t.* m and applying the permutation symbol to the right interpolator in (6.7) and (6.8) results in a much cheaper summation scheme in terms of both memory and computational time. Let us show this by writing the new interpolation formula as follows

$$\tilde{\sigma}^A_{ij}(\mathbf{x}) = -\sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) \sum_{|\boldsymbol{\beta}|\leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{\ell\ell} \varepsilon_{jmk} \sum_{[\mathbf{x}^1\mathbf{x}^2]\in(\mathcal{C}')} b'_k t'_i \int_{\mathbf{x}^1}^{\mathbf{x}^2} \nabla_m S_{\boldsymbol{\beta}}(\mathbf{x}')dx' \tag{6.11}$$

$$\tilde{\sigma}^B_{ij}(\mathbf{x}) = -\sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) \sum_{|\boldsymbol{\beta}|\leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{ij} \varepsilon_{nmk} \sum_{[\mathbf{x}^1\mathbf{x}^2]\in(\mathcal{C}')} b'_k t'_n \int_{\mathbf{x}^1}^{\mathbf{x}^2} \nabla_m S_{\boldsymbol{\beta}}(\mathbf{x}')dx' \tag{6.12}$$

Let us define the new multipole expansions associated with $\tilde{\boldsymbol{\sigma}}^{\mathbf{A}}$ as

$$(\mathcal{M}^A_{\boldsymbol{\beta}})_{ji} = \varepsilon_{jmk} \sum_{[\mathbf{x}^1\mathbf{x}^2]\in(\mathcal{C}')} b'_k t'_i \int_{\mathbf{x}^1}^{\mathbf{x}^2} \nabla_m S_{\boldsymbol{\beta}}(\mathbf{x}')dx'$$

Since summation over m can now be performed at the `P2M` step, the dimension of this multipole expansion is divided by 3. Hence, for all $(i,j) \in \{1,2,3\}^2$ with $i \leq j$, the associated fast multipole summation scheme consists in the following steps:

- For $\tilde{\boldsymbol{\sigma}}^{\mathbf{A}'}$, perform successively

$$(\mathcal{M}_{\boldsymbol{\beta}}^{A'})_{ji} = (\mathcal{M}_{\boldsymbol{\beta}}^{A})_{ij} + (\mathcal{M}_{\boldsymbol{\beta}}^{A})_{ji} \qquad \texttt{(Shift-P2M-SA)}$$

$$(\mathcal{L}_{\boldsymbol{\alpha}}^{A'})_{ij} = \sum_{|\boldsymbol{\beta}|\leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{\ell\ell} (\mathcal{M}_{\boldsymbol{\beta}}^{A'})_{ij} \qquad \texttt{(Shift-M2L-SA)}$$

$$\tilde{\sigma}_{ij}^{A'}(\mathbf{x}) = -\sum_{|\boldsymbol{\alpha}|\leq p} S_{\boldsymbol{\alpha}}(\mathbf{x})(\mathcal{L}_{\boldsymbol{\alpha}}^{A'})_{ij} \qquad \texttt{(Shift-L2P-SA)}$$

- For $\tilde{\boldsymbol{\sigma}}^{\mathbf{B}'}$, perform successively

$$\mathcal{M}_{\boldsymbol{\beta}}^{B} = (\mathcal{M}_{\boldsymbol{\beta}}^{A})_{nn} = tr(\boldsymbol{\mathcal{M}}_{\boldsymbol{\beta}}^{A}) \qquad \texttt{(Shift-P2M-SB)}$$

$$(\mathcal{L}_{\boldsymbol{\alpha}}^{B})_{ij} = \sum_{|\boldsymbol{\beta}|\leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{ij} \mathcal{M}_{\boldsymbol{\beta}}^{B} \qquad \texttt{(Shift-M2L-SB)}$$

$$\tilde{\sigma}_{ij}^{B'}(\mathbf{x}) = (\tilde{\sigma}^{B})_{ij} - \delta_{ij}(\tilde{\sigma}^{B})_{\ell\ell} \qquad \texttt{(Shift-L2P-SB)}$$

- Finally, compute $\tilde{\boldsymbol{\sigma}}$ using (6.4).

Cost estimation In order to perform this summation, we need to store 7 M2L components, namely 1 for $\tilde{\boldsymbol{\sigma}}^{\mathbf{A}}$ and 6 for $\tilde{\boldsymbol{\sigma}}^{\mathbf{B}}$. Moreover, we need to store 6 components of $\boldsymbol{\mathcal{M}}_{\boldsymbol{\beta}}^{A'}$ and 1 components of $\boldsymbol{\mathcal{M}}_{\boldsymbol{\beta}}^{B}$. On the other hand, one need to apply 12 M2L operations in order to compute all 6 components of $\tilde{\boldsymbol{\sigma}}^{\mathbf{A}'}$ and $\tilde{\boldsymbol{\sigma}}^{\mathbf{B}'}$. Indeed, (Symm-M2L-SA) involves 6 M2L operations and (Symm-M2L-SB) involves 6 M2L operations. This cost reduction should compensate the need for a larger interpolation order induced by the loss of precision.

6.2.4 Comparative analysis of the theoretical cost

Here, we summarize the cost of each variant in two tables, please refer to Table 6.1 for the memory requirements and Table 6.2 for the computational costs. The tables show the constant that needs to be applied to the memory requirements or the computational costs of a given interpolation based FMM (*bbfmm* or *ufmm*), for each step and each operator.

Storage Table 6.1 shows that the variant using shifted derivatives requires the less overall memory. Concerning the two other optimized variants, using the symmetries requires approximately half of the memory required when refactoring the interactions for the storage of the expansions. On the other hand, the memory footprint of the M2L does not vary significantly for these 2 approaches. For the shifted variant, one should mention that the computation of the energy does nor require extra memory for the storage of the M2L operators since they share the same kernel of interaction. However different multipole expansions have to be stored for the stress and the energy, respectively 7 and 6.

Application In order to compare the computational cost of each tensorial approach, one should consider the constant in front of the cost of a scalar M2L written in Table 6.2. Conse-

Quantity	Approach	$\#\mathcal{L}$	$\#\bar{K}$	$\#\mathcal{M}$	$(\#\mathcal{M}+\#\mathcal{L})/\#\bar{K}$
Stress	Direct	9	$3+27=30$	$27+3=30$	1.3
	Symm	6	$3+10=13$	$18+3=21$	2.1
	Facto[1]	6	$10+1=11$	9	1.4
	Facto[2]	6	$11+3=14$	12	1.4
	Shift	6	$6+1=7$	$6+1=7$	1.9
Energy	Direct	6	$6+1=7$	6	1.7

Table 6.1: Number of FMM operators required for the computation of the stress and the energy, that represent the constants in front of the theoretical memory requirements for each approach.

quently, the variant using shifted derivatives should be the fastest one in theory, since it has the lowest constant. The cost of the P2M and L2P steps is similar for all optimized variants, except that the P2M of the symmetric variant is far more expensive than that of the other optimized variants.

Quantity	Approach	L2P	M2L	P2M
Stress	Direct	9	$27+27=54$	27
	Symm	6	$18+18=36$	27
	Facto[1]	6	46	9
	Facto[2]	6	$18+33=51$	9
	Shift	6	$6+6=12$	$6+1=7$
Energy	Direct	19	47	9

Table 6.2: Constants in front of the theoretical complexity for each approach *w.r.t.* to the FMM steps.

Discussion For a given variant, the *ufmm* is always faster than the *bbfmm* in theory. This assumption is verified experimentally on the numerical benchmarks presented in Section 6.4. Moreover, the *ufmm* require less memory for low tree depths. However, as the depth of the tree grows, the expansions require more memory than the M2L and the *ufmm* becomes more demanding. This may occur for lower depths than in the scalar case, since the ratio between the number of components of the expansions and the M2L exceeds 1 for all approaches. Figure 6.2 shows the total theoretical memory requirement of each variant for either the *bbfmm* or the *ufmm*. We observe that for $p = 4$ (a typical value used in *OptiDis*, that provides good accuracy) the profile does not change a lot due to the small variations of the ratio (between 1 and 2). The profiles only differ for the low tree depths where the cost of the M2L dominates in the *bbfmm*. For large tree depth, the memory footprints of the *ufmm* is always larger since the storage of the expansions dominates the overall memory footprint and it is substantially larger than that of the *bbfmm*. Based on previous observations, the

optimal choice between the symmetric and the refactored variant may vary *w.r.t.* to the chosen interpolation grid. For instance, the refactored variant seems better suited for a FMM with fast M2L application such as the *ufmm*. Meanwhile, the symmetric variant can benefit from low memory requirements at the P2M step such as in the *bbfmm*. Finally, for a fixed interpolation grid, the variant using shifted derivatives has the lowest overall memory footprint.

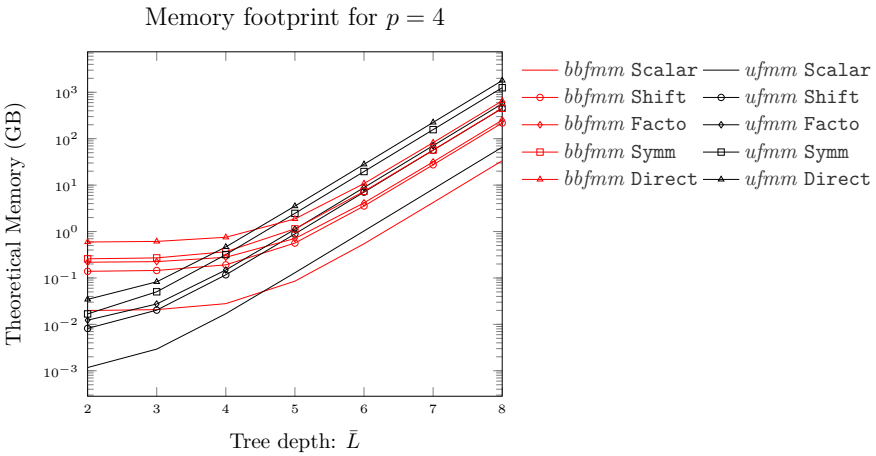

Figure 6.2: Theoretical memory requirements at the leaf level *w.r.t.* the tree depth in the worst case scenario (no empty leaf) and for a fixed interpolation order $p = 4$. We consider the maximum amount of memory required for storing the 316 tensorial M2L operators, the multipole and the local expansions as well as the transformed versions. The results are given in simple precision arithmetic. Observations: The memory required by the M2L is almost not noticeable for the *ufmm*, in fact only the cost of storing the expansions is observed. For the *bbfmm*, the memory required for the storage of the M2L dominates the cost and can thus be observed for the first values of $\bar{L}$.

6.3 Implementation details

6.3.1 The extended bounding box concept

Extending leaves Inserting the elements of a spatial discretization in an octree structure can sometimes be tedious, in particular when an element crosses an edge of the octree (see Figure 6.3). For instance, in most Fast Multipole BEM implementations [31, 86], a boundary element is considered to be in a cluster, if the center of the element lies inside of it. Moreover, since interpolation is involved in our algorithm, it is crucial that the entire element lies inside the interpolation domain. Therefore, as suggested by [86] we define a slightly larger interpolation domain, called the extended bounding box, that contains all segments lying inside a cluster. Since extending the bounding box reduces the minimum distance between points of admissible clusters, it obviously affects the accuracy of the method. Associating segments to a cluster with respect to their center leads to the minimal width extension, *i.e.*, the maximum segment length (see 6.3). Note that the center of the cell remains unchanged after extension of the width.

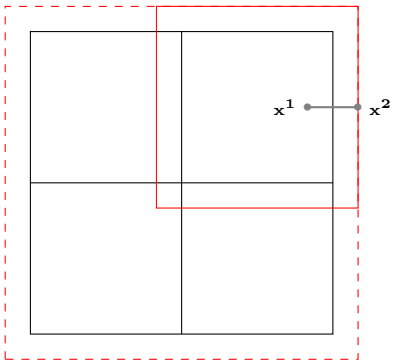

Figure 6.3: Illustration of the bounding box extension in the worst case scenario, where the center of a segment $\left[\mathbf{x}^1\mathbf{x}^2\right]$ lies on the edge of a leaf cell.

Extending parent cells Once the leaf cells are extended all parent cells need to be extended as well, since interpolation is performed from child to parent and anterpolation from parent to child. The minimal parent's width extension, such that the extended child is contained inside the extended parent, is equal to the child's width extension. Therefore, we use the same width extension for all cells of all levels. Note that since the center remains unchanged, the extensions of adjacent cells intersect.

6.3.2 Numerical integration over segments

Farfield As described extensively in appendix D.3, during the P2M step (resp. L2P step), the integrals over the segments $[\mathbf{x}^1\mathbf{x}^2]$ (resp. $[\mathbf{x}^3\mathbf{x}^4]$) can be computed exactly by mean of a simple Gaussian quadrature rule. For the integral involved in (P2M-SA) or (P2M-SB) the resulting quadrature formula reads as

$$\int_{\mathbf{x}^1}^{\mathbf{x}^2} S_{\boldsymbol{\beta}}(\mathbf{x}')dx' = \sum_{q=1}^{Q} \omega'_q S(\mathbf{x'}_q),$$

where Q denotes the quadrature size satisfying $2Q+1=p$ and ω'_q (resp. $\mathbf{x'}_q$) denotes the q-th Gauss weights (resp. points) mapped to the source segment $[\mathbf{x}^1\mathbf{x}^2]$. More precisely, if $\omega_q \in [0,1]$ and $\theta_q \in [-1,1]$ denote standard Gauss weights and points, then mapped weights reads as

$$\omega'_q = \omega_q \frac{L}{2}$$

and mapped points read as

$$\mathbf{x'}_q = \mathbf{x}^1 + \xi(\theta_q)\mathbf{t} = \mathbf{x}^1 + (\theta_q + 1)\frac{L}{2}\mathbf{t}.$$

Nearfield As mentioned in the introduction 6.1.1, because of the condition imposed on the tree depth, the efficiency of the FMM is lost after a certain problem size. In fact, once the tree depth reaches its maximum value, the cost of computing the direct interactions starts to grow quadratically n. Therefore, it is crucial to have an efficient implementation of the P2P step. First of all, *OptiDis* already optimizes the evaluation of the nearfield using

AVX instructions. Second of all, we implemented a new feature, that allows the direct interactions to be evaluated using numerical quadratures if segments are sufficiently far appart. Indeed, as mentioned in [8], using a quadrature can becomes cheaper than using analytical expressions for very distant segments. For instance, if the target accuracy is 10^{-4}, then the required quadrature size equals 2 for $d > 5L_{max}$ and drops to 1 for $d > 500L_{max}$. Since the maximum distance at the maximum level is $d_{max} = 6L_{max}$, this represents a small gain at the maximum level. However, this can significantly accelerate the nearfield computation for the low FMM levels if the root bounding box is very large compared to the maximum segment length. For instance, if $L_{max} = 100$ with a root bounding box of width $\omega_0 = 100.000$ then $d_{max}/L_{max} = \omega_0/(2L_{max}) = 500$ at level $\bar{L} = 2$.

6.3.3 Enforcing homogeneity

Unlike the singular interaction kernel, the non-singular kernel is not homogeneous (see Section 1.2.4) because of the core width, that creates a shift in the distance. However, since a is very small compared to the minimum cell width ω_{min}, *i.e.*, $\omega_{min} = 3L_{max}$, then both kernels almost coïncides in the farfield, as shown on Figure 6.4. Consequently, using the singular kernel in the farfield computation instead of the non-singular variant should only result in a minor extra error compared to the interpolation error. This assumption is validated in Section 6.4.2. On the other hand, as described in Section 1.1.3, given the homogeneity of the kernel, we can store the `M2L` operators at only one reference level and thus divide the memory footprint of the `M2L` step by the number of level $\bar{L} - 1$. We would like to highlight, that special care must be taken, when combining extended bounding boxes with the scaling of a homogeneous kernel.

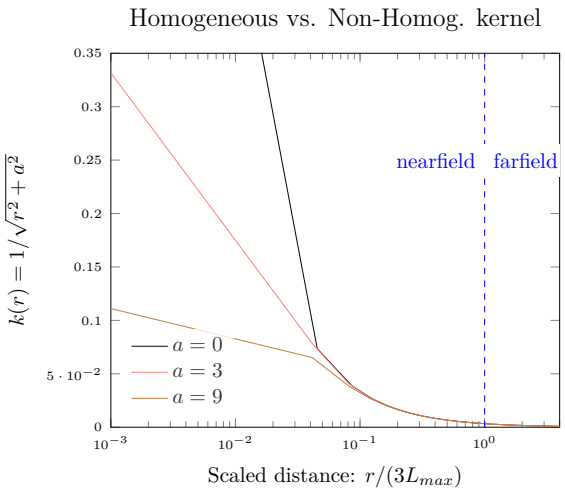

Figure 6.4: Comparison of the homogeneous kernel $k(r) = 1/r$ with the non-homogeneous kernel $k_a(r) = 1/\sqrt{r^2 + a^2}$ for 2 representative values of a. The kernel values are plotted *w.r.t.* the scaled distance r/ω_{min} with $\omega_{min} = 3L_{max}$ using a typical maximum segment length of $L_{max} = 90$. Non-homogeneous kernels are represented for $a = L_{max}/30 = 3$ (average core width) and $a = L_{max}/10 = 9$ (large core width). Observations: The non-singular kernels coïncide with the singular kernel in the farfield, even for large values of the core width.

6.4 Numerical benchmarks

In this section, we present 2 artificial benchmarks in order to validate the accuracy our approach. We first verify the accuracy of the various Fast Multipole summation schemes presented in the previous Section 6.2 and then determine an appropriate interpolation order p, that leads to an accuracy of 10^{-4} on the force field. Then, we discuss their respective sequential running times *w.r.t.* the number of segments using either the *ufmm* or the *bbfmm*.

6.4.1 Distributions of loops and visualization of the fields

In order to validate our implementation we do not need a very complex topology. However, it is crucial to respect the order of magnitude of the various parameters, that are involved in the dynamical simulation. In order to ensure the network to be closed and allow easy monitoring of the number of segments, we consider 2 very basic distributions of dislocation loops:

- the first one enforces 1 BCC loop per leaf (Fig. 6.5). As the tree grows, the diameter D and the minimum and maximum segment length decrease. More precisely,

$$D = 1000/2^{\bar{L}-2} \quad \text{and} \quad L = L_{min} + 1 = L_{max} - 1 = 200/2^{\bar{L}-2}. \tag{6.13}$$

- the second uses a prescribed density d of loops in m/m^3 (Fig. 6.6). As the tree grows, the loop diameter D and segment lengths remain constant but the density grows.

Hence, in both cases the number of segment per leaf remains constant and the tree is always fully populated (worst case scenario) except for lowest tree depth $\bar{L} = 2$.

(a) $4 \times 4 \times 4$ loops, $n = 1.920$, $\bar{L} = 2$ **(b)** $8 \times 8 \times 8$ loops, $n = 15.360$, $\bar{L} = 3$

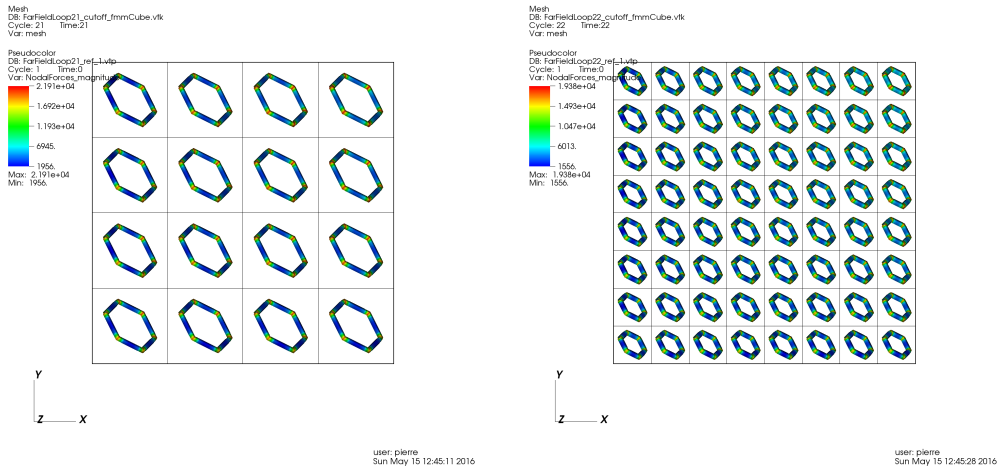

Figure 6.5: 3 configurations of BCC loops with $\mathbf{b} = [111]$. The number of segments is driven by the number of loops, *i.e.*, the number of leafs, while the number of segments per loop remains constant. The exact nodal force modules are represented on the dislocation lines for each configuration.

(a) $d = 5.10^{19} m^{-2}$, $n = 11.250$, $\bar{L} = 4$ **(b)** $d = 10^{20} m^{-2}$, $n = 90.090$, $\bar{L} = 5$

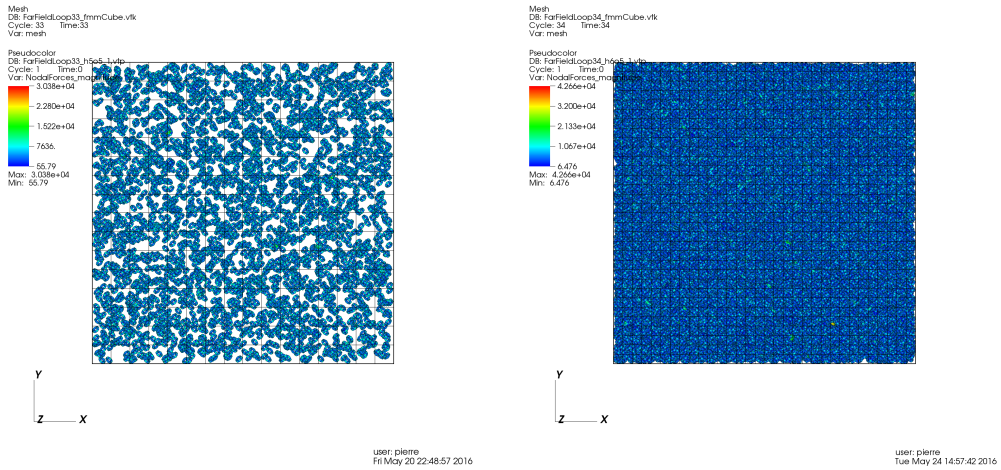

Figure 6.6: 3 configurations of BCC loops with $\mathbf{b} = [111]$. The number of segments is driven by the density d, while the segment length $L = 20$ and loop diameter $D = 100$ remain constant. The exact nodal force modules are represented on the dislocation lines for each configuration.

6.4.2 Accuracy

Sources of error In order to verify the accuracy of the various schemes implemented during this thesis, we consider the convergence in term of the relative 2-norm error on the force and energy *w.r.t.* interpolation order p in the worst case scenario, where all possible sources of errors are taken into account. Therefore, we consider the distribution of loops driven by the specified density for the following reasons, since it is the closest to real life simulations, the network is closed and dislocation can cross the edges of the octree. Finally, the last source of errors is the use of the homogeneous kernel instead of the regularized kernel. Figure 6.7 represents the experimental accuracy of the *ufmm* and the *bbfmm* for a direct approach and the variant using shifted derivatives.

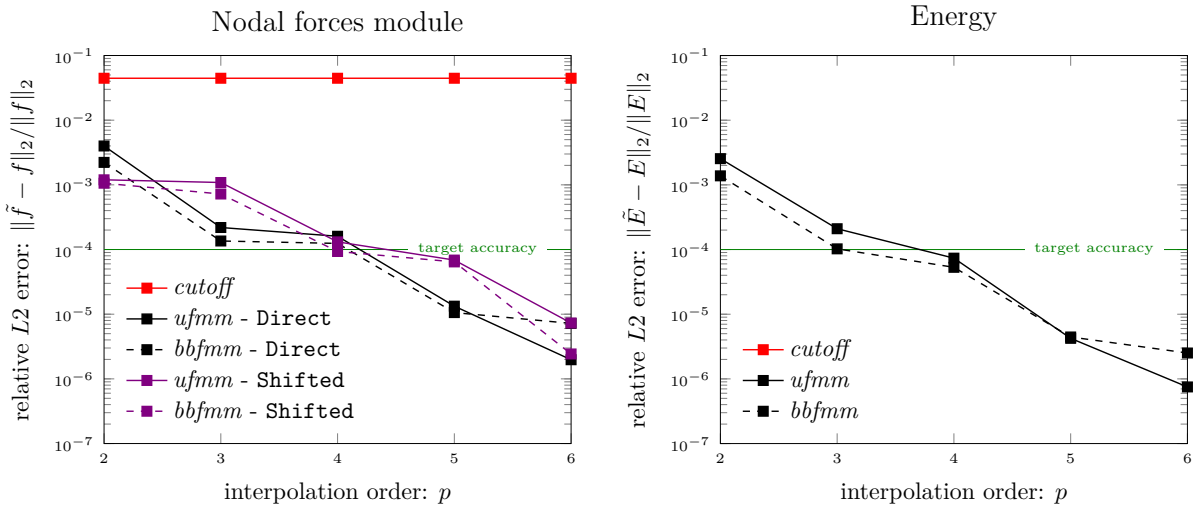

Figure 6.7: Accuracy of the nodal forces (left) and energy (right) evaluation for different interpolation schemes (*ufmm* and *bbfmm*, with and without shifted derivative) *w.r.t.* the interpolation order p. The farfield core width is set to 0. Configuration (b) of Fig. 6.5.

Discussion The accuracy of the *cutoff* variant is represented on the graphs in order to show the benefit of having a farfield implementation, in this case the error on the force evaluation equals 4.10^{-2}. The rate of convergence is similar to the scalar case. As expected, we observe a convergence delay of 1 interpolation order for certain values of p between the direct and the shifted cases. However, for $p = 4$ all schemes have about the same accuracy of 10^{-4}. Therefore, in the range of accuracy targeted by the application, all schemes use the same p.

6.4.3 Sequential performance

Validation procedure First of all, in order to properly evaluate the relative performance of each variant we consider both configurations at comparable tree depths for $p = 4$. Then, in order to better show the asymptotic complexity of the method, we enforce a constant n_0 and a fully populated octree. Therefore, for a given distribution of loops, we first determine the optimal n for $\bar{L} = 2$ and then multiply n by 8 each time we increment $\bar{L}$. The sequential running times of the `Factorized` and the `Shifted` variants are represented on Figure 6.8.

Discussion As expected, all variants have a linear complexity with the number of segments, however they do not evaluate the farfield with the same efficiency. The simulations driven by the specified density reach their permanent regime a little slower, since for the first densities the octree is not fully populated. It is important to mention, that by fixing the tree depth we cannot ensure that all variants balance nearfield and farfield computations optimally. In particular, in our numerical benchmarks, only the `Symmetric` variant with *bbfmm* is balanced. Nevertheless, the cheapest approach is the `Shifted` variant with *ufmm*, where the cost of the farfield computation is 3 times lower than for the `Factorized` variant.

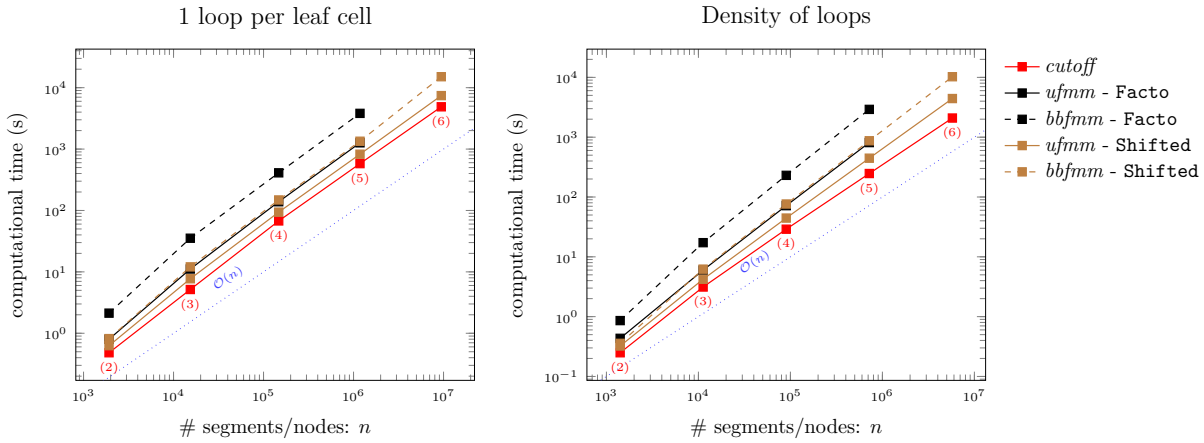

Figure 6.8: Computational time required by the nodal forces evaluation for different interpolation schemes *w.r.t.* the number of segments n. The tree depth $\bar{L}$ is displayed under the marks of the *cutoff* variant. Left: Configurations of Fig. 6.5, ensures a constant number of segments per leaf and a fully populated octree. Right: Configurations of Fig. 6.6 is more realistic, however the asymptotic complexity is harder to emphasizes, since there exists empty leaf cells. Observations: Only the *ufmm*-`Facto` and the *bbfmm*-`Symm` variants have well-balanced nearfield and farfield. For a given n, the *bbfmm*-`Facto` is 3 times slower and the *ufmm*-`Shift` is about 2 times faster. Machine: *plafrim2/miriel* 2 Dodeca-core Haswell Intel Xeon E5-2680 @ 2,5 GHz RAM 128Go (DDR4 2133MHz).

Part III

Randomized low-rank approximations of *covariance* matrices

7

Sampling from Gaussian Random Fields

Contents

7.1 Fundamentals and computational aspects

Generating realizations of Gaussian Random Fields (GRFs) is a key issue in numerous scientific research fields such as cosmology [100, 14, 29], geostatistics [69, 73], hydrogeology [104, 12] or even Brownian dynamics [10]. A GRF is a multivariate Gaussian Random Variable (GRV), *i.e.*, a vector $\mathbf{Y} \in \mathbb{R}^n$ of correlated Gaussian random variables each associated with the points of a grid denoted $\{\mathbf{x}_i\}_{i=1,\dots,n}$. In the present work we will only consider spatial grids, *i.e.*, $\mathbf{x}_i \in \mathbb{R}^3$.

7.1.1 Basics

The concept of GRF is extensively presented in [1], in this section we will recall its fundamental concepts and discuss several properties of spatial correlation functions and covariance matrices.

Multivariate Random Variables In order to accurately represent the first moments of a statistical distribution, namely the mean and the variance, GRFs usually need to be sampled in large ensembles (*laws of large numbers*). For instance, let X be a univariate GRV with mean 0 and variance 1, as shown on Table 7.1, one need to perform about 10^3 realizations X_i in order to reach 1% accuracy in term of sample mean and variance, namely

$$\mathbb{E}(X) = \frac{1}{n_{real}} \sum_{i=1}^{n_{real}} X_i \quad \text{and} \quad \mathbb{E}(X^2) = \frac{1}{n_{real}} \sum_{i=1}^{n_{real}} X_i^2$$

In the multivariate case, let say a length-n GRF, *i.e.*, n GRVs, if $n = 10^3$ one needs $n_{real} = \mathcal{O}(10^5)$ realizations to reach errors between 1 and 10% in terms of sample mean $\mathbb{E}(\mathbf{Y})$ and covariance $\mathbb{E}(\mathbf{Y}\mathbf{Y}^\mathbf{T})$. Therefore, efficiently generating many realizations of a GRF given a large spatial grid, *i.e.*, $n = \mathcal{O}(10^6)$, can rapidly become challenging even on modern computers.

n_{real}	$\mathbb{E}(X) - 0$	$roc_{\mathbb{E}}$	$\mathbb{E}(X^2) - 1$	$roc_{\mathbb{E}^2-1}$	t_X (s)
$1 \cdot 10^2$	$7.68 \cdot 10^{-2}$		0.26		$1.62 \cdot 10^{-5}$
$1 \cdot 10^3$	$1.8 \cdot 10^{-2}$	0.63	$4.64 \cdot 10^{-2}$	0.74	$1.26 \cdot 10^{-4}$
$1 \cdot 10^4$	$8.89 \cdot 10^{-3}$	0.31	$1.84 \cdot 10^{-2}$	0.4	$5.07 \cdot 10^{-4}$
$1 \cdot 10^5$	$1.76 \cdot 10^{-3}$	0.7	$5.18 \cdot 10^{-3}$	0.55	$5.79 \cdot 10^{-3}$

Figure 7.1: Convergence of a univariate GRV in terms of sample mean error $\mathbb{E}(X) - 0$ and sample variance error $\mathbb{E}(X^2) - 1$ *w.r.t.* the number of realizations n_{real}. We used a Mersenne Twister random number generator with mean 0 and variance 1. The rates of convergence (roc) are defined as the ratio of the logarithm of two consecutive values of an error. Finally, t_X denotes the time required to generate the random variable and compute the errors. We observe the theoretical $1/2$ convergence rate for the mean and variance.

Correlation kernels We use the terminology $\mathbf{Y} \sim \mu(\mathbf{0}, \mathbf{C})$ to define a GRF $\mathbf{Y}$ with mean $\mathbf{0}$ and covariance $\mathbf{C} \in \mathbb{R}^{n \times n}$. The covariance matrix can be prescribed as a kernel matrix,

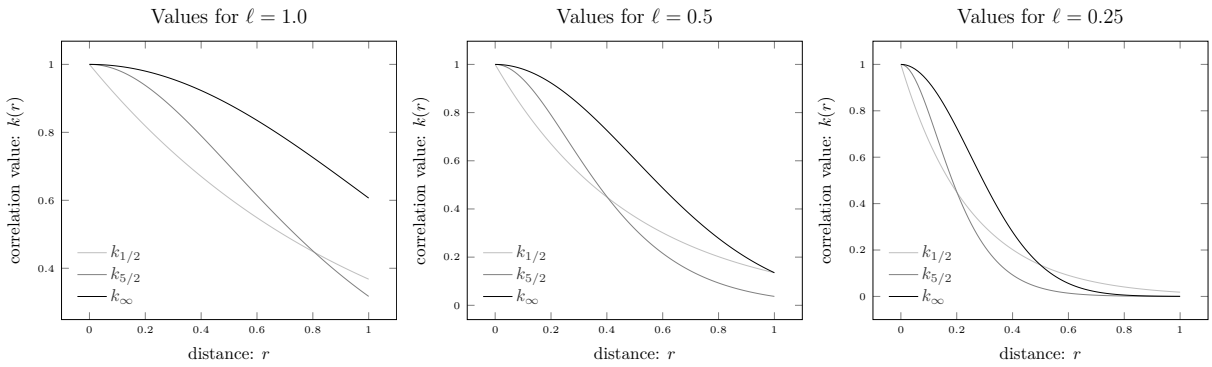

Figure 7.2: Representation of the correlation functions k_∞,$k_{5/2}$ and $k_{1/2}$ *w.r.t.* the distance $r \in [0, 1]$ for $\ell = 1.0$ (left), $\ell = 0.5$ (center) and $\ell = 0.25$ (right).

i.e.,

$$\mathbf{C} = \{k(r_{ij})\}_{i,j=1,\dots,n},$$

where $r_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||_2$. The kernel k denotes a correlation function such as the Gaussian kernel

$$k_\infty(r) = e^{-r^2/(2\ell^2)} \tag{7.1}$$

or the exponential kernel

$$k_{1/2}(r) = e^{-r/\ell} \tag{7.2}$$

where the length scale ℓ characterizes the decreasing speed of the correlation function. These functions are the extreme cases of Matérn functions [85] defined as

$$k_\nu(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}(\frac{\sqrt{2\nu}r}{\ell})^\nu B(\frac{\sqrt{2\nu}r}{\ell}), \tag{7.3}$$

where B denotes a modified Bessel function and Γ the gamma function [2]. In particular, as shown on Figure 7.2, k_∞ is globally smooth and $k_{1/2}$ has discontinuous first order derivative at the origin. In this chapter, we also consider $k_{5/2}$, which presents an intermediate degree of smoothness and is often used in machine learning applications. Covariance matrices are symmetric positive definite (*spd*) by definition of correlation kernels [1]. Hence, $\mathbf{C}$ admits the following representation

$$\mathbf{C} = \mathbf{A}\mathbf{A}^\mathbf{T}, \tag{7.4}$$

where the matrix factor $\mathbf{A} \in \mathbb{R}^{n\times n}$ will be called a *square root* of the covariance matrix $\mathbf{C}$, even though others may call it a Cholesky factor and would expect the square root of an *spd* matrix to be *spd* itself.

Sampling from GRF Realizations of a GRF $\mathbf{Y} \sim \mu(\mathbf{0}, \mathbf{C})$ can be obtained in $\mathcal{O}(n^2)$ by applying $\mathbf{A}$ to a length-n white noise $\mathbf{X}$, *i.e.*, by computing

$$\mathbf{Y} = \mathbf{A}\mathbf{X}, \tag{7.5}$$

where $\mathbf{X} \sim \mu(\mathbf{0}, \mathbf{I_n})$ is a Gaussian Random Field verifying $\mathbb{E}(\mathbf{X}) = \mathbf{0}$ and $\mathbb{E}(\mathbf{X}\mathbf{X}^T) = \mathbf{I_n}$.

7.1.2 Limits of standard and alternative techniques

There exist numerous approaches for generating GRFs, that usually differ by the way $\mathbf{A}$ is precomputed. Approaches based on standard matrix decompositions such as the Singular Value Decomposition (SVD) or the Cholesky Decomposition [37] are the most popular ones, since they provide an exact factorization and rely on well understood and robust algorithms that apply to any *spd* matrix $\mathbf{C}$. If the decomposition can be truncated at a prescribed numerical rank r, then the resulting low-rank factor $\mathbf{A_r} \in \mathbb{R}^{n\times r}$ can be applied to a length-r white noise in $\mathcal{O}(n)$ operations in order to generate a GRF Y with mean $\mathbf{0}_N$ and covariance close to $\mathbf{C}$, *i.e.*, $\mathbb{E}(\mathbf{Y}\mathbf{Y}^\mathbf{T}) = \mathbf{A_r}\mathbf{A_r^T} \approx \mathbf{C}$.

Alternatives Nevertheless, standard factorization algorithms involve $\mathcal{O}(n^3)$ operations and thus become computationally prohibitive for large n, *i.e.*, n over a few thousands. Alternative methods are often considered such as the *sequential simulation* method [67], the *moving average* methods [95], the *turning bands* method [69, 81], continuous [107] or discrete [62, 104] spectral methods, combinations of spectral methods with the *turning bands* [80] or with *moving average* [73], improved matrix decomposition [36]. They usually provide approximate square roots but most importantly they do not always extend to 3D grids.

Circulant Embedding A popular alternative based on matrix decomposition, known as *circulant embedding* technique, was developed in the early 90's [43, 122]. It provides an exact method for computing the product $\mathbf{Y} = \mathbf{A}\mathbf{X}$, that has a $\mathcal{O}(n)$ cost in both running time and storage, but most importantly $\mathbf{A}$ is assembled in Fourier domain in $\mathcal{O}(n \log n)$ operations using the Fast Fourier Transform (FFT). However, it presents significant numerical limitations, especially in 3D, not to mention that it only applies to equispaced grids.

7.2 Fast randomized algorithms for generating Gaussian Random Fields

Based on the observation that many covariance matrices of practical interest actually have low-rank, Dehdari and Deutsch [42] used the Randomized SVD (rSVD) in order to accelerate the precomputation of A in low-rank form and thus efficiently generate realizations of GRFs. In this section, we present the details of this randomized algorithm and its numerical limitations. Then, we show that it can be significantly improved by using a $\mathcal{H}^2$-representation of the covariance matrix such as the Fast Multipole Method (FMM).

7.2.1 Randomized approach

Correlation functions used in practical applications usually lead to globally low-rank covariance matrices, therefore the rSVD (Algorithm 6) can be considered in order to efficiently approximate $\mathbf{C}$ by its approximate low-rank representation

$$\tilde{\mathbf{C}} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{U}}^T. \tag{7.6}$$

Since covariance matrices are by definition *symmetric positive semi-definite* (*spsd*), a low-rank matrix factor can then be computed as

$$\tilde{\mathbf{A}} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}^{1/2} \in \mathbb{R}^{n\times r}. \tag{7.7}$$

In fact, since $\tilde{\mathbf{A}}$ provides an exact square root of $\tilde{\mathbf{C}}$, *i.e.*, $\tilde{\mathbf{C}} = \tilde{\mathbf{A}}\tilde{\mathbf{A}}^T$, then $\tilde{\mathbf{A}}$ provides an approximate square root of $\mathbf{C}$, *i.e.*, $\mathbf{C} \approx \tilde{\mathbf{A}}\tilde{\mathbf{A}}^T$. The matrix $\tilde{\mathbf{A}}$ can be applied to a length-r white noise at a $\mathcal{O}(n\times r)$ cost, which allows for generating realizations of n-variate Gaussian random variables at a linear cost in time. However, this approach still requires the input covariance matrix $\mathbf{C}$ to be fully assembled and the cost of precomputing an approximate square root scales quadratically with n.

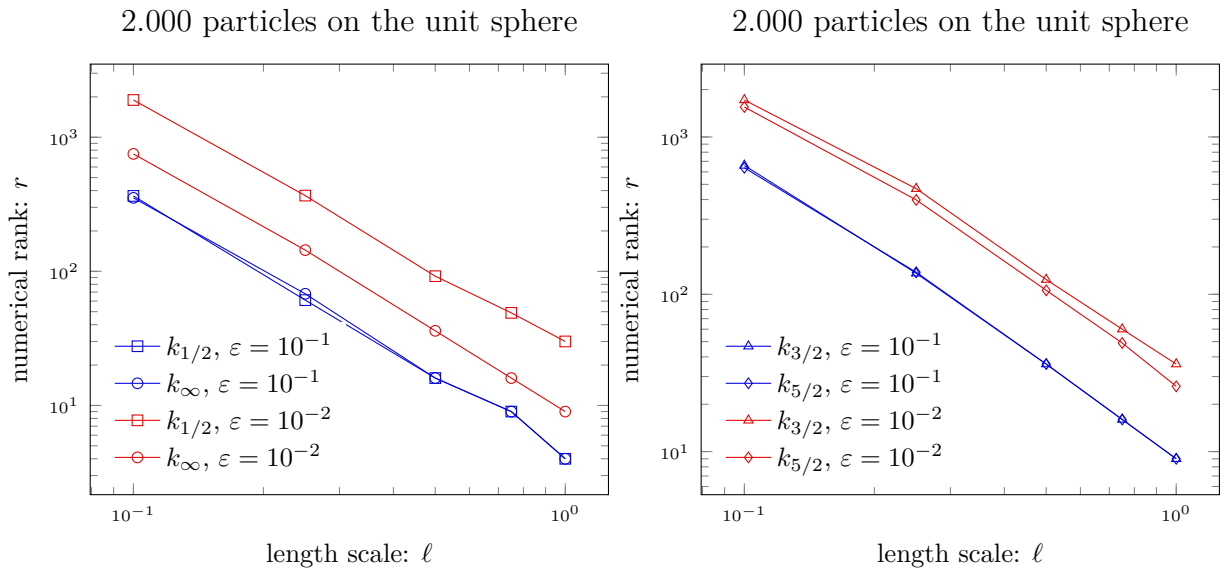

Figure 7.3: Numerical rank r of some covariance kernel matrices *w.r.t.* the correlation length scale ℓ for 1% and 10% target accuracy in Frobenius norm. We used $n = 2.000$ particles distributed on the unit sphere. Observations: In this range of accuracy the rank grows quadratically with the inverse of ℓ, while being inferior to 10 for $\ell = 1$. Finally, the rank never exceeds 40% of the full size except for the exponential kernel $k_{1/2}$ with $\ell = 1.0$.

Numerical rank The complexity of the method grows linearly with the rank of the matrix, therefore its applicability is restricted to matrices with a relatively low rank. Before describing our new algorithm we would like to evaluate if the kernel matrices we want to approximate fulfill this condition. For a given accuracy and a given norm, the rank r can be determined numerically by considering the best rank-r error computed from the spectrum of

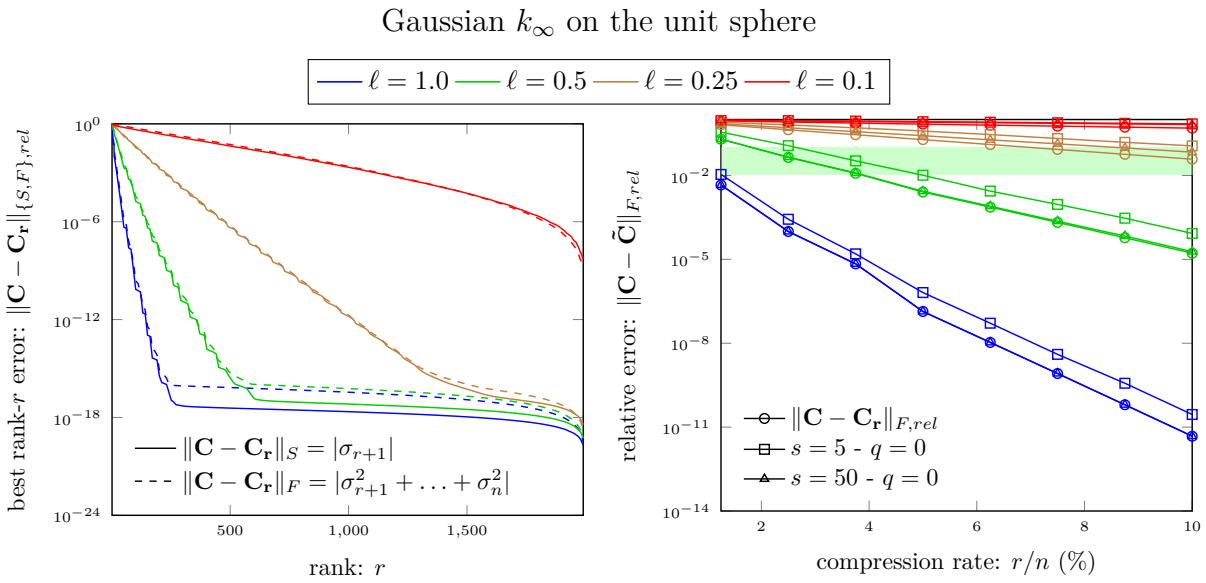

Figure 7.4: Accuracy of the fixed rank rSVD *w.r.t.* the compression rate for a Gaussian correlation and $n = 2.000$ particles distributed on the unit sphere. Left: Optimal relative error in Spectral and Frobenius norms *w.r.t.* to $r = 1, \ldots, n$. Right: Accuracy of the rSVD *w.r.t.* prescribed rank $r = 20, \ldots, 200$. Effect of the oversampling s alone ($q = 0$). Observations: The Gaussian kernel is smooth, therefore its spectrum decreases fast and $\mathbf{C}$ is relatively low-rank. Consequently, the rSVD performs well even without power iterations and with low oversampling ($s = 5$). An oversampling of $s = 50$ leads to a near optimal error. Note that $\mathbf{C}$ becomes high-rank for small ℓ, *e.g.*, $\ell = 0.1$.

the input matrix, we call r the *numerical* rank. The target accuracy required for generating GRFs is usually low, namely between 10^{-1} and 10^{-2} in either Frobenius or Spectral norm. In this thesis we chose to use the Frobenius norm to determine the numerical rank, since it is more restrictive than the Spectral norm and it better accounts for the shape of the spectrum. In order to allow the computation of the full spectrum within a reasonable time by mean of a full SVD, we chose a relatively low value of n, namely $n = 2.000$. In fact, as shown by our numerical benchmarks, for sufficiently dense distributions, *i.e.*, $n > 1.000$ for the unit sphere, the rank of the kernel matrix does not depend on n but rather on the kernel function and the geometry.

Oversampling Figure 7.4 illustrates the benefit of using oversampling with the Gaussian kernel k_∞ (7.1) for $n = 2.000$ particles distributed on the unit sphere. No power iteration is required, since the associated covariance matrix has a fast decreasing spectrum. Hence, we set $q = 0$ and analyze the effect of the oversampling. First of all, the results show that the fixed rank approach works well for matrices of relatively low rank, *e.g.* for $\ell \geq 0.50$. For instance, if $\ell \geq 0.50$, $r/n \approx 5\%$ and $s = 5$, then the relative error in Frobenius norm is already below 2%. On the other hand, for $\ell = 0.1$ the matrix is not low-rank, since for a compression rate of 10% the optimal relative error exceeds 10%. Moreover, as shown on Figure 7.4, increasing the oversampling significantly improves the accuracy and even provide *near-optimal* accuracy on this particular test case. In fact, for $\ell \geq 0.5$, the error for $s = 50$ almost coïncides with the theoretical lower bound $\|\mathbf{C} - \mathbf{C}_r\|_F$.

Subspace iterations On the other hand, Figure 7.6 illustrates the benefit of using subspace iterations with the exponential kernel $k_{1/2}$ (7.2), that exhibits a relatively flat spectrum. In particular, it shows that with an oversampling of $s = 50$ the accuracy is still far from the optimal. However, a *near-optimal* accuracy can be reached with a few subspace iterations, namely $q = 1$. Since in this case the matrix is not low-rank, it only produces a raw approximation of $\mathbf{C}$ in the desired range of compression, *i.e.*, compression rates below 10%. Hence, we will not consider this kernel in the numerical benchmarks presented in Section 7.4.

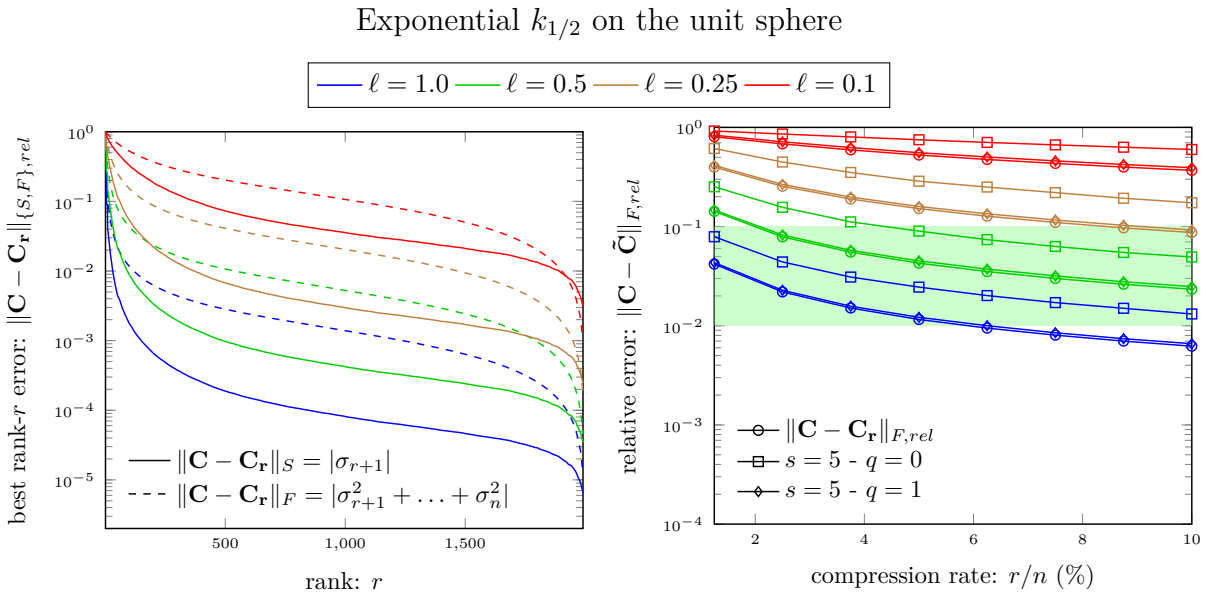

Figure 7.5: Accuracy of the fixed rank rSVD *w.r.t.* the compression rate for an exponential correlation and $n = 2.000$ particles distributed on the unit sphere. Left: Optimal relative error in Spectral and Frobenius norms *w.r.t.* to $r = 1, \ldots, n$. Right: Accuracy of the rSVD *w.r.t.* prescribed rank $r = 20, \ldots, 200$. Effect of q power iterations using a low oversampling, namely $s = 5$. Observations: The exponential correlation $k_{1/2}$ is not smooth, therefore its spectrum is relatively flat. One power iteration leads to a *near-optimal* error but the optimal error remains high. In fact, a 1% relative error is not reachable for $\ell < 0.5$ for a maximum compression rate of 10%.

7.2.2 FMM-powered randomized algorithms

Algorithm & Theoretical complexity In the rSVD, the matrix-to-matrix multiplications involved at each stage, namely $\mathbf{C\Omega}$ for the random projection (3.4) and $\mathbf{CQ}$ for the assembly of $\mathbf{B} = \mathbf{Q^T CQ}$ (3.6), dominate the asymptotic cost of the algorithm. Therefore, the overall cost of the original randomized SVD is quadratic in n in both running time and memory footprint. Since $\mathbf{C}$ is given as a kernel matrix it can be applied to another matrix at a linear cost in n without ever assembling the full matrix using a $\mathcal{H}^2$-method like the *ufmm* or the *bbfmm*. The resulting approach remains a dense random projection as $\mathbf{\Omega}$ stays dense, but it benefits from a data sparse representation of $\mathbf{C}$. Algorithm 8 presents our $\mathcal{H}^2$-powered randomized SVD and indicates the asymptotic complexity of each stage. The resulting algorithm has an overall $\mathcal{O}(r^2 \times n)$ computational cost, which is one order less than the $\mathcal{O}(r \times n^2)$ complexity of the original method. However, we will show that for the

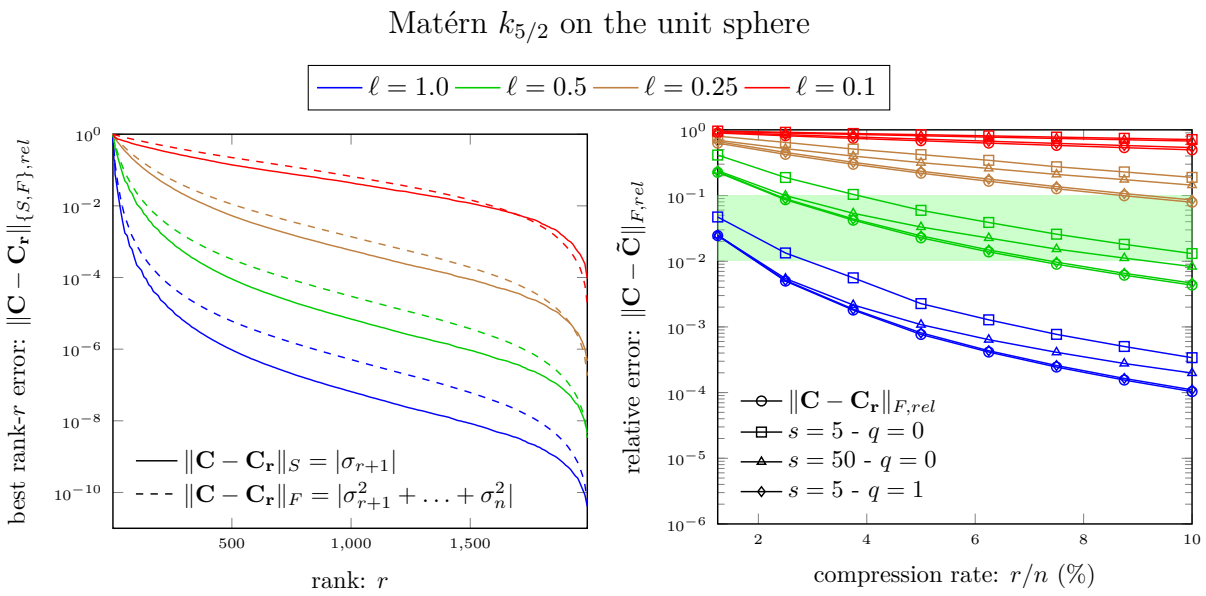

Figure 7.6: Accuracy of the fixed rank rSVD *w.r.t.* the compression rate for a Matérn $\nu = 2.5$ correlation and $n = 2.000$ particles distributed on the unit sphere. Left: Optimal relative error in Spectral and Frobenius norms *w.r.t.* to $r = 1, \ldots, n$. Right: Accuracy of the rSVD *w.r.t.* prescribed rank $r = 20, \ldots, 200$. Observations: The correlation is smoother than the exponential, therefore its spectrum decreases faster. On one hand, it seems insufficient to increase the oversampling by a constant value, namely $s = 5$ or $s = 50$. On the other hand, one power iteration leads to a *near-optimal* error but the optimal error remains relatively high. In fact, a 1% relative error is not reachable for $\ell < 0.5$ for a maximum compression rate of 10%.

grid size and accuracy used in practice, the matrix multiplications still dominate the cost although they have a $\mathcal{O}(r \times n)$ complexity.

Algorithm 8: $\mathcal{H}^2$-powered Randomized SVD

Input: Correlation kernel $k(.,.)$, grid size n, positions $\mathbf{x} \in (\mathbb{R}^3)^n$, rank r or accuracy ε, number of power iterations q, oversampling parameter s, interpolation order p, octree depth h

Output: $\left[\tilde{\mathbf{A}}\right]$ approximate square root of $\mathbf{C} = \{k(x_i, x_j)\}_{i,j=1,\ldots,n}$

// Stage 0: Build $\mathcal{H}^2$ representation of $\mathbf{C}$

Build $\boldsymbol{\mathcal{H}^2\mathbf{C}} = \mathcal{H}^2(k, x, N, p, h)$

// Stage I: Approximate the range of $\mathbf{C}$

if *accuracy ε is prescribed* **then**

  $[\mathbf{Q}, r] = ARRF(\boldsymbol{\mathcal{H}^2\mathbf{C}}, q, s, \varepsilon)$ $\quad \mathcal{O}(r^2 \times n)$

if *rank r is prescribed* **then**

  $[\mathbf{Q}, \varepsilon] = RSI(\boldsymbol{\mathcal{H}^2\mathbf{C}}, q, s, r)$ $\quad$ (idem)

// Stage II: Decompose $\mathbf{C_r} = \mathbf{Q}(\mathbf{Q^T}\boldsymbol{\mathcal{H}^2\mathbf{C}}\mathbf{Q})\mathbf{Q^T} \approx \boldsymbol{\mathcal{H}^2\mathbf{C}}$ as $\mathbf{U\Sigma U^T}$

Build $\mathbf{B} = \mathbf{Q^T}(\boldsymbol{\mathcal{H}^2\mathbf{C}}\mathbf{Q}) \in \mathbb{R}^{r \times r}$ $\quad \mathcal{O}(r^2 \times n)$

Perform SVD of $\mathbf{B} = \mathbf{U_B}\mathbf{\Sigma_B}\mathbf{U_B^T}$ $\quad \mathcal{O}(r^3)$

// Compute square-root $\tilde{\mathbf{A}} = \tilde{\mathbf{C}}^{1/2} \approx \mathbf{C}^{1/2}$ as $\tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}^{1/2}$

$\mathbf{B^{1/2}} = \mathbf{U_B}\mathbf{\Sigma_B^{1/2}}$ $\quad \mathcal{O}(n \times r^2)$

$\tilde{\mathbf{A}} = \mathbf{Q}\mathbf{B^{1/2}}$ $\quad \mathcal{O}(n \times r^2)$

Accuracy The FMM provide approximate matrix multiplications, therefore it introduces extra errors in each stage of the algorithm. The accuracy of the FMM needs to be carefully

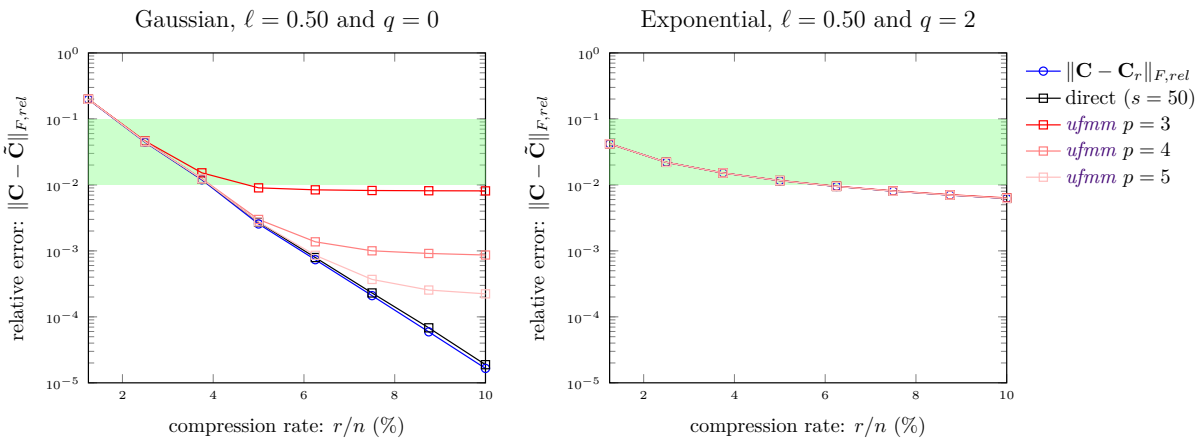

Figure 7.7: Accuracy of the fixed rank $\mathcal{H}^2$-powered rSVD *w.r.t.* the compression rate for $n = 2.000$ particles distributed on the unit sphere. Left: For a Gaussian kernel with $\ell = 0.5$ the accuracy of the rSVD can be very high even below 10% compression. However, the *ufmm* only allows for representing the covariance matrix up to a fixed accuracy. For a given interpolation order p, the $\mathcal{H}^2$ powered variant has the same accuracy as the dense variant then slightly reaches a plateau. Right: For the exponential kernel, even with $\ell = 1.0$ the accuracy of the rSVD is relatively low, therefore the *ufmm* never disturbs the method in this range of compression and p can be set to the lowest value.

monitored in order to avoid perturbing the approximation of the range and the final accuracy of the rSVD. In particular, the optimal order of interpolation should not affect the output rank of the Adaptive Randomized Range Finder (ARRF) or the accuracy of the Randomized Subspace Iterations (RSI). Figure 7.7 shows the effect of *ufmm* on the accuracy of the rSVD for We found that enforcing one order of magnitude between the error on a fast matrix multiplication $\varepsilon_{fmm} = \|\bar{\phi} - Kw\|_{2,rel}$ and the prescribed accuracy ε of the randomized algorithm, *i.e.*,

$$\varepsilon = 10\varepsilon_{fmm}, \tag{7.8}$$

preserves the accuracy of the original method, while still providing significantly better performance. Hence, in the context of GRF, the interpolation order of the *ufmm* never exceeds $p = 4$.

7.3 Efficient Fast Multipole matrix-to-matrix multiplication

In this section, we present an optimized method for performing efficient Fast Multipole matrix-to-matrix multiplication, *i.e.*, FMM with multiple right-hand-sides. In particular, we describe the efficient implementation of the interpolation operators (`P2M`, `M2L` & `L2P`) and the `P2P` operators in order to minimize the cost of the algorithm in terms of both storage and running times as the number of right-hand-sides grows.

7.3.1 Blocking farfield computations

Generic notations Handling multiple right-hand-sides in a FMM computation consists in computing multivariate potentials using a scalar interactions kernel and multivariate

densities. Let r denote the number of right-hand-sides, then the i-th component of the potential generated at the target point $\mathbf{x}$ by the i-th component of the density $\mathbf{w}$ located at the source point $\mathbf{y}$ reads as

$$\phi_i(\mathbf{x}) = k(\mathbf{x}, \mathbf{y}) w_i(\mathbf{y}), \, \forall i = 1, \ldots, r. \tag{7.9}$$

The interpolation formula used for the farfield computations reads as

$$\bar{\phi}_i^{far}(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}| \leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) \sum_{|\boldsymbol{\beta}| \leq p} \bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}} S_{\boldsymbol{\beta}}(\mathbf{y}) w_i(\mathbf{y}) \tag{7.10}$$

Summation scheme The fast multipole summation consists in performing the following steps:

- `P2M`: Transfer the contributions of the r components of the density $\mathbf{w}$ located at the source points $\mathbf{y}$ into the r multipole expansions $(\mathcal{M}_{\boldsymbol{\beta}})_i$ defined on the source interpolation nodes $\bar{\mathbf{y}}_{\boldsymbol{\beta}}$
$$(\mathcal{M}_{\boldsymbol{\beta}})_i = \sum_{\mathbf{y}} S_{\boldsymbol{\beta}}(\mathbf{y}) w_i(\mathbf{y}), \, \forall i = 1, \ldots, r \tag{7.11}$$

- `M2L`: Transfer the r multipole expansions $(\mathcal{M}_{\boldsymbol{\beta}})_i$ into the r local expansions $(\mathcal{L}_{\boldsymbol{\alpha}})_i$ defined on the target interpolation nodes $\bar{\mathbf{x}}_{\boldsymbol{\alpha}}$
$$(\mathcal{L}_{\boldsymbol{\alpha}})_i = \sum_{|\boldsymbol{\beta}| \leq p} \bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}} \, (\mathcal{M}_{\boldsymbol{\beta}})_i \, , \, \forall i = 1, \ldots, r \tag{7.12}$$

- `L2P`: Transfer the contributions of the r local expansions $(\mathcal{L}_{\boldsymbol{\alpha}})_i$ defined on the target interpolation nodes $\bar{\mathbf{x}}_{\boldsymbol{\alpha}}$ into the r components of the potential $\bar{\boldsymbol{\phi}}$ located at the target points $\mathbf{x}$
$$\bar{\phi}_i(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}| \leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) \, (\mathcal{L}_{\boldsymbol{\alpha}})_i \, , \, \forall i = 1, \ldots, r \tag{7.13}$$

Memory footprint Although it does not require extra work or memory during precomputation, this summation scheme presents strong limitations in term of storage, in particular when it comes to storing the expansions at run time. Indeed, the r multipole expansions, r local expansions, and the associated transformed expansions have to be stored independently. For instance, as shown on Figure 5.6, for $p = 4$ and a densely populated octree of maximum level $\bar{L} = 6$, the memory required to store the expansions associated to one right-hand-side equals 5 GigaBytes. Therefore, the number r of right-hand-sides should not exceed $r_{max} = 10$, because this would result in a total memory footprint of 50GB for the expansions only. As a comparison, storing the entire matrix would cost $8 \times (10^5)^2 = 80$GB. Consequently, if the input r exceeds r_{max}, *e.g.*, r is a fraction of the problem size, then we need to perform several sequential but independent multi-*rhs* FMM.

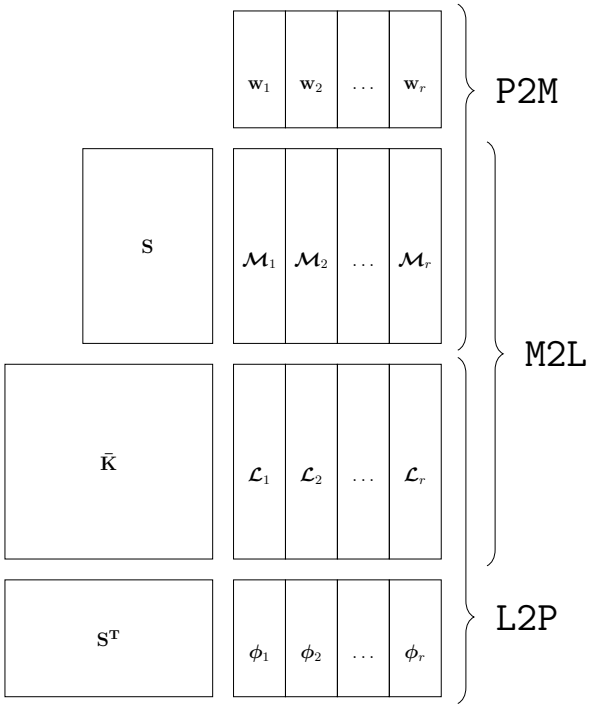

Figure 7.8: Schematic view of the FMM with multiple (r) right-hand-sides.

Computational cost In a naïve implementation, the computational cost of this scheme would roughly equal r times the cost of a standard FMM. First of all, our implementation of the P2M and L2P steps decouples the computation of the polynomial interpolators and the multivariate expansions, in order to factorize as much operations as possible. Second of all, the M2L step is implemented in a similar fashion as the tensorial approach presented in Appendix A. In the *bbfmm*, if r were large enough, the M2L step could be performed in a block way instead of component by component, *e.g.*, through a matrix-to-matrix product based on level-3 Blas routines. However, due to the excessive memory required for large r this approach is not relevant. In the *ufmm*, there is no point in blocking the operations, since we perform an entry-wise product. Consequently, at the M2L step in both variants, we transfer one *rhs* at a time using a simple loop over r_{max}. Hence, transferring all *rhs* costs r times more than transferring one *rhs*.

7.3.2 Precomputing the P2P operators

As we just demonstrated, there is not much room for improvement in the farfield computation when performing FMM with multiple *rhs*, since we quickly reach a memory limit. On the other hand, as explained in this subsection, many operations can be factorized during the nearfield computation.

Implementation The nearfield computations of a single *rhs* FMM involve the computation of the kernel for all pairs of particle lying in neighbor cells and the direct evaluation of the sum (7.9). However, there is no need to compute a new set of interactions for each right-hand-side, the P2P operators can instead be precomputed and stored. The P2P opera-

tors are eventually applied to a set of r *rhs* at once using `Blas::gemm()` routines provided by the Intel MKL library. As a result, the application of the P2P is a lot faster than in the original algorithm and the balance is not necessarily obtained for the same tree depth $\bar{L}$.

Storage The memory required for storing the P2P operators

$$\mathbf{K} = \left\{k(\mathbf{x}_i, \mathbf{y}_j)\right\}_{\mathbf{x}_i \in tree, \mathbf{y}_j \in \mathcal{N}(\mathcal{C}_{\mathbf{x_i}}^{(\bar{L})})} \tag{7.14}$$

equals $n \times 27 n_0 \times 8$ bytes in double precision arithmetic if the tree is fully populated. Since the memory footprint scales like n, *i.e.*, like the number of (non-empty) cells, then it can quickly become prohibitive. Figure 7.9 shows the theoretical amount of memory needed for storing $\mathbf{K}$ for a uniform distribution of particles in a cube for various values of n and n_0. If we consider typical simulations using $n_0 = n/8^{\bar{L}} = 64$ particles per leaf, then for $n = 256.000$ particles with $\bar{L} = 4$, we would already need 3.5GB and for $n = 2 \times 10^6$ particles with $\bar{L} = 5$, we would need about 30GB.

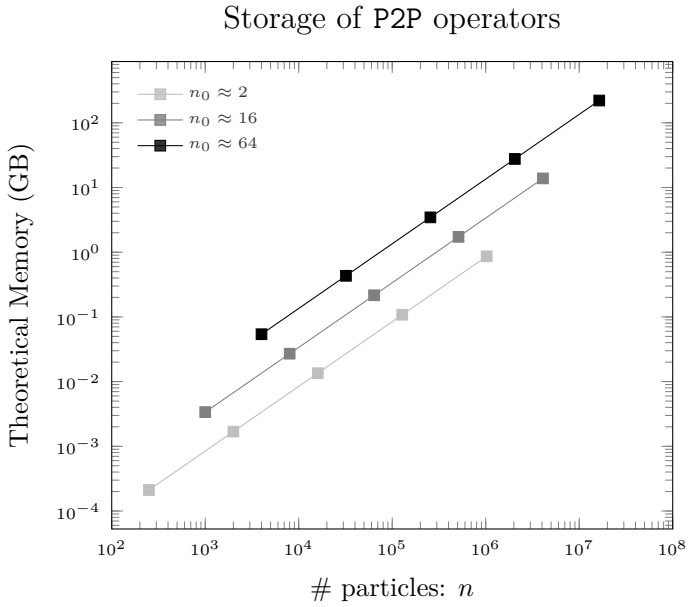

Figure 7.9: Theoretical memory requirements for the nearfield computation in the worst case scenario (no empty cell).

7.4 Numerical Benchmarks

In this section, we present the performance of our algorithm within the range of accuracy required by the generation of Gaussian Random Fields. We first present the computational running time of our multi-*rhs* FMM for a smooth kernel. Then, we compare the performance of the $\mathcal{H}^2$-powered rSVD with the rSVD and the truncated SVD. Finally, we validate our method by generating many Gaussian Random Fields and verifying the convergence of the sample mean and covariance *w.r.t.* the number of realizations. We also present realizations on the unit sphere with the Gaussian kernel for multiple length scales.

7.4.1 Fast Matrix multiplication

multi-rhs In order to calibrate our multi-*rhs* FMM, we measure the performance of the *ufmm* and *smooth-ufmm* for $p = 3$, which ensures an accuracy of $\varepsilon_{fmm} = 10^{-3}$ on the matrix multiplications. Figure 7.10 shows the running times per matrix-to-vector product for $r = 10$ *rhs* and particles distributed in the unit cube or on the unit sphere. Since the experiments are very similar to those presented in Section 5.4, we draw similar conclusions concerning the relative performance of the *ufmm* and the *smooth-ufmm*, as well as concerning the good performance on heterogeneous grids such as the unit sphere. The computational time are significantly lower since the interpolation order is lower and the `P2P` operators are precomputed. In order to show the speed-up of the multi-*rhs* FMM compared to the direct computation, we assemble a full covariance matrix and apply it to $r = 10$ *rhs* using MKL Blas routines.

global fft The *smooth-ufmm* with $\bar{L} = 0$ boils down to an FFT-accelerated Lagrange interpolation on the root bounding box. We call this method the *global fft* as it generalizes the idea of the FFT on an arbitrary grid. The *global fft* has a theoretical linear complexity in n, that can be verified on Fig. 7.10. Furthermore, for a given root bounding box the performance of the *global fft* are independent of the shape of the distribution. Although this method requires larger interpolation orders to reach similar accuracies as the *ufmm* and the *smooth-ufmm*, that both rely on several subdomains, only one expansion has to be transfered. As shown on Fig. 7.10, even in this range of accuracy, the global approach is always slower than the hierarchical variants even with a fully populated octree (Fig. 7.10 left). Moreover, since hierarchical variants benefit from the heterogeneity of the distribution, they beat the *global fft* by a significant factor on geometries such as the unit sphere (Fig. 7.10 right). The figures presented in Appendix C confirm these observations for a fixed n and a varying accuracy. However, they show that the *global fft* exhibits performance that are similar to the hierarchical variants for high values of ℓ, *i.e.*, very smooth kernels.

7.4.2 Fast Randomized SVD

The accuracy and the $\mathcal{O}(n)$ complexity of the $\mathcal{H}^2$-powered rSVD are now illustrated for a Gaussian correlation with $\ell = 0.5$ and particles distributed on the unit sphere.

Optimal setup As shown by Figure 7.4 the covariance matrix can be represented by a matrix of rank $r \approx 70$ with a precision of about $\varepsilon = 10^{-2}$ in Frobenius norm using a conventional rSVD. In this case, our experiments showed that the smallest interpolation order that does not affect the randomized algorithm is $p = 3$ for the *ufmm*, *i.e.*, $\varepsilon_{fmm} < 10^{-3}$. If the desired accuracy equals $\varepsilon = 10^{-3}$ then $r \approx 100$ and $p = 4$ was found optimal, *i.e.*, $\varepsilon_{fmm} < 10^{-4}$. This confirms the optimal setup rule proposed in (7.8).

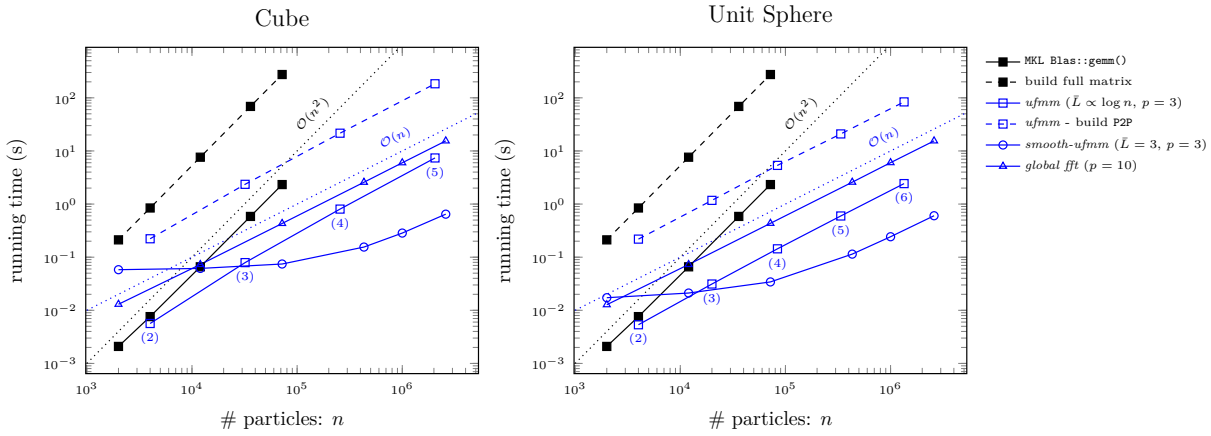

Figure 7.10: Running time (s) per matrix-to-vector product *w.r.t.* the number of particles n for a total of $r = 10$ vectors and target accuracy of about 10^{-3}. In order to compare all algorithms we use a smooth kernel, namely the Gaussian kernel with $\ell = 0.5$. Particles are distributed uniformly in the unit cube (left) and on the unit sphere (right). The tree depth $\bar{L}$ of the *ufmm* (written below blue squares) ensures a relatively constant average number of particles per leaf: $n_0 = 74$ (left) and $n_0 = 62$ (right). Machine: *plafrim/riri* - Deca-core Intel Xeon E7-4870 @ 2.40GHz with 1TB ram and 30MB L3 Cache.

Complexity & Speed-up Figure 7.11 shows the time required to build an approximate square root of $\mathbf{C}$ within an accuracy of 10^{-2} in Frobenius norm. It confirms the theoretical asymptotic complexities of all approaches, namely cubic for the full SVD (in red), quadratic for the truncated SVD based on Arnoldi's algorithm (in green) and randomized SVD (in black) and finally linear for the $\mathcal{H}^2$-accelerated randomized SVD. As mentioned in [64], we observe that the running times of the fixed rank and fixed accuracy variants are very close to each other and that they outperform the truncated SVD by a factor of 6.5. Moreover, the $\mathcal{H}^2$ variants (*ufmm* and *smooth-ufmm*) are faster than the original rSVD for $n \geq 10^4$. Not only do these variants allow computation to be performed for n up to a few millions but they also provide significant speedup, *e.g.*, they would be about 10 times faster for $n = 10^5$ if only a dense matrix multiplication was affordable. This acceleration is even more pronounced for smooth kernels and heterogeneous grids as explained in Section 5.4 and recalled by the results displayed on Figure 7.10. Finally, we notice that even if the cost of computing matrix multiplication has been dramatically reduced, it still governs the overall cost of the algorithm. For instance, for $n = 2.592.000$ particles and $r = 70$ the random projection is done in about 360s, while the QR Decomposition only takes 50s. This can be seen directly on Figure 7.10 since the curves corresponding to FMM algorithms have the same shape than those of Figure 7.11.

7.4.3 Realizations of Gaussian Random Fields

Gaussian Random Fields were simulated on particles distributed on the unit sphere using Gaussian correlations with varying length scales ℓ. The approximation is done by mean of a fixed precision ($\varepsilon = 10^{-2}$) rSVD with *ufmm*- or *smooth-ufmm* accelerated matrix multiplications. Realizations are displayed on Figure 7.12 for length scales $\ell \in \{0.25, 0.5\}$ using the *smooth-ufmm*. The figure shows that the actual correlation length between grid points roughly corresponds to the input ℓ. Smaller length scales computed by mean of a standard

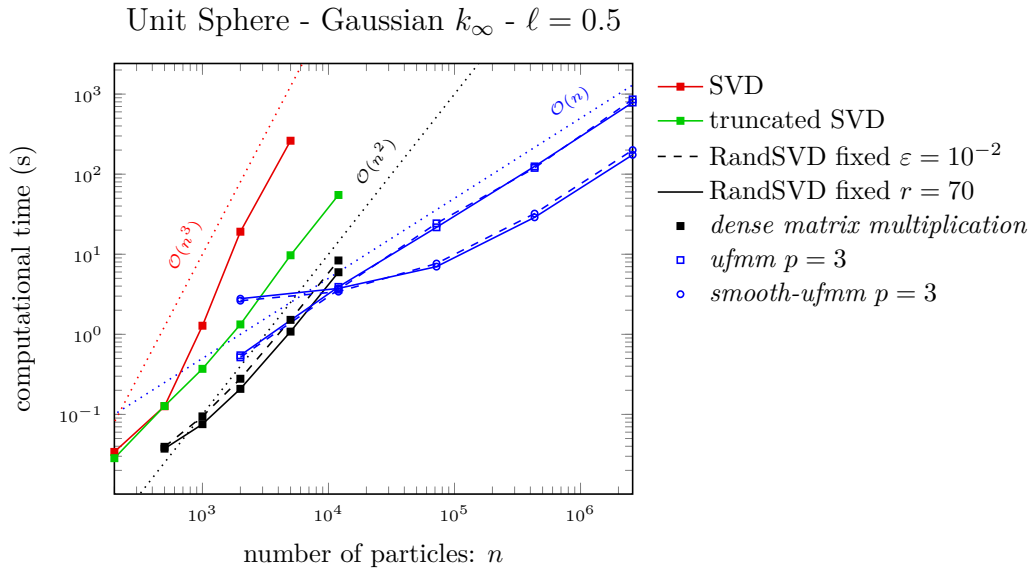

Figure 7.11: Time for computing a randomized SVD using either the fixed rank or fixed accuracy algorithm with $q = 0$ and $s = 10$. Matrix multiplications are computed either in a dense fashion (filled squares) or by means of the *ufmm* (empty squares) or *smooth-ufmm* (empty circles) with $p = 3$, particles are randomly distributed on the unit sphere and the correlation is Gaussian with $\ell = 0.5$. Dense matrix multiplications, SVD and QR Decomposition are performed using sequential MKL Blas routines. Observations: As expected, both fixed rank and fixed accuracy variants exhibit similar performance. On the other hand, the graphs confirm the theoretical asymptotic costs (linear in blue, quadratic in black and cubic in red). Machine: plafrim/riri - Deca-core Intel Xeon E7-4870 @ 2.40GHz with 1TB ram and 30MB L3 Cache.

ufmm are displayed on Figure 7.13. In particular, it shows that decreasing the accuracy by one order of magnitude is hardly noticeable on such visualizations. Furthermore, since the length scale is quite low, namely $\ell = 0.1$, the rank is relatively large but the square root is still precomputed in a reasonable amount of time, namely 90 seconds for 10^{-1} accuracy and 175 seconds for 10^{-2}.

Accuracy The sample covariance matrix $\mathbf{C}_{real}$, computed from the realizations as

$$\mathbf{C}_{real} = \frac{1}{n_{real}} \sum_{i=1}^{n_{real}} (\mathbf{Y}_i - \mathbb{E}(\mathbf{Y}_i))(\mathbf{Y}_i - \mathbb{E}(\mathbf{Y}_i))^t \tag{7.15}$$

provides a good approximation of the experimental covariance of a finite sample. Therefore, we analyze the accuracy of the method using the error between $\mathbf{C}_{real}$ and the input covariance matrix $\mathbf{C}$. In order to limit the computational cost required by verifications, the error can be computed on a random subset of the matrices. The accuracy of the method *w.r.t.* the number of realizations n_{real} is presented in Table 7.1. These results show that the *ufmm* and the *smooth-ufmm* can be used to generate correlated random fields within similar accuracies for $\ell \geq 0.25$, while maintaining a convergence rate that is close to the theoretical $1/2$. For lower length scales we recommend using the *ufmm* in order to better capture the fast variation of the kernel, *i.e.*, the potentially high rank of the matrix.

Running times As shown on Figure 7.11, for a Gaussian correlation with $\ell = 0.50$, building an approximate square root with $n = 72.000$, $r = 70$ takes about 20 seconds with the

ufmm, 8 seconds with the *smooth-ufmm* while it would take about 5 minutes with dense matrix multiplications. For $\ell = 0.25$, the rank r and the computational times are about 3 times larger. The *ufmm* also provides significant speed-up for matrices of relatively high rank associated with low length scales ℓ or non-smooth correlations. For a given length scale, the performance of the *ufmm* depends only very weakly on the nature of the correlation function, therefore the computational cost associated with non Gaussian correlation can be easily estimated once the rank is known.

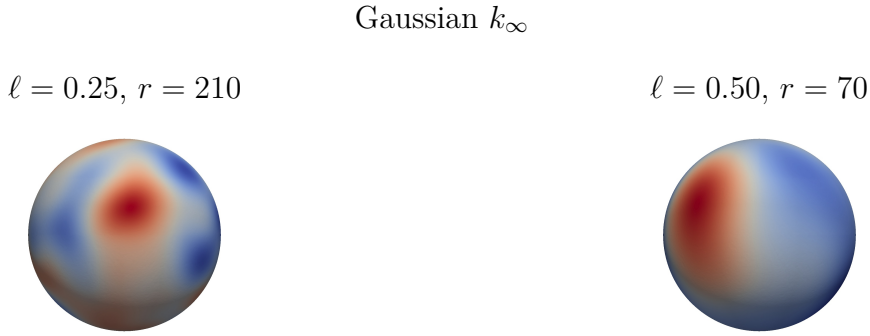

Figure 7.12: Realization of a Gaussian random field on $n = 72.000$ points distributed on the unit sphere using a Gaussian correlation with $\ell = 0.25$ (left) and $\ell = 0.50$ (right). The approximate square root was obtained by mean of a *smooth-ufmm*-accelerated rSVD with a fixed accuracy of $\varepsilon = 10^{-2}$.

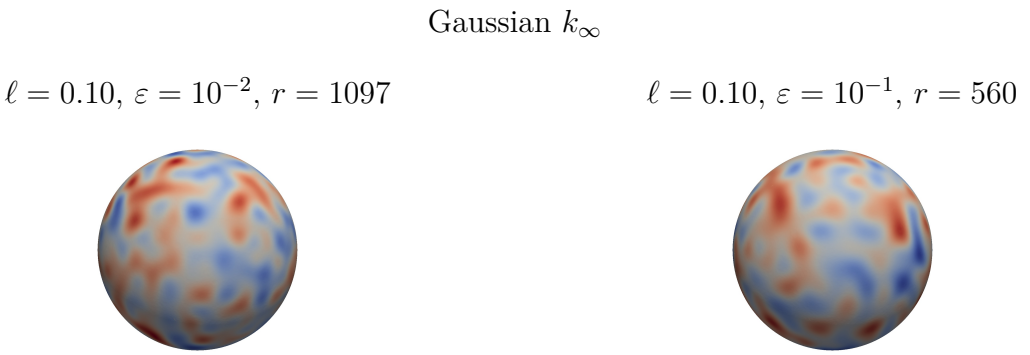

Figure 7.13: Realization of a Gaussian random field on $n = 72.000$ points distributed on the unit sphere using a Gaussian correlation. The approximate square root was obtained by mean of a *ufmm*-accelerated rSVD with a fixed accuracy of $\varepsilon = 10^{-2}$ (left) and 10^{-1} (right). Observations: The differences in terms of covariance quality and smoothness of the field are hardly noticeable on a 3D visualization. However, since the rank is roughly divided by 2, the method is twice as fast for the lower target accuracy.

7.5 Conclusion

Square root algorithm We designed a matrix-free algorithm for computing approximate square roots of kernel covariance matrices in SVD form in linear time. This algorithm relies on a new highly efficient kernel independent FMM that can further benefit from the heterogeneity of the grid and the smoothness of the correlation kernel. The presentation of the algorithm and the experimental results are summarized in a research report [15]. Many geostatistical applications and in particular Kalman Filtering, a very popular data assimilation technique, involve intensive matrix computations on arbitrary grids with up to several

$\ell = 0.25$

ufmm

n_{real}	$\|\mathbf{C}_{real} - \mathbf{C}\|_{2,rel}$	roc_2	$\|\mathbf{C}_{real} - \mathbf{C}\|_{\infty,rel}$	roc_∞
$1 \cdot 10^3$	$2.48 \cdot 10^{-1}$		$1.47 \cdot 10^{-1}$	
$1 \cdot 10^4$	$7.88 \cdot 10^{-2}$	0.5	$4.53 \cdot 10^{-2}$	0.51
$1 \cdot 10^5$	$2.63 \cdot 10^{-2}$	0.48	$1.57 \cdot 10^{-2}$	0.46
$1 \cdot 10^6$	$1.02 \cdot 10^{-2}$	0.41	$8.53 \cdot 10^{-3}$	0.27

smooth-ufmm

n_{real}	$\|\mathbf{C}_{real} - \mathbf{C}\|_{2,rel}$	roc_2	$\|\mathbf{C}_{real} - \mathbf{C}\|_{\infty,rel}$	roc_∞
$1 \cdot 10^3$	$2.48 \cdot 10^{-1}$		$1.57 \cdot 10^{-1}$	
$1 \cdot 10^4$	$7.88 \cdot 10^{-2}$	0.5	$5.68 \cdot 10^{-2}$	0.44
$1 \cdot 10^5$	$2.67 \cdot 10^{-2}$	0.47	$1.84 \cdot 10^{-2}$	0.49
$1 \cdot 10^6$	$1.09 \cdot 10^{-2}$	0.39	$1.09 \cdot 10^{-2}$	0.23

$\ell = 0.50$

ufmm

n_{real}	$\|\mathbf{C}_{real} - \mathbf{C}\|_{2,rel}$	roc_2	$\|\mathbf{C}_{real} - \mathbf{C}\|_{\infty,rel}$	roc_∞
$1 \cdot 10^3$	$1.31 \cdot 10^{-1}$		$1.12 \cdot 10^{-1}$	
$1 \cdot 10^4$	$3.75 \cdot 10^{-2}$	0.54	$4.56 \cdot 10^{-2}$	0.39
$1 \cdot 10^5$	$1.41 \cdot 10^{-2}$	0.43	$1.76 \cdot 10^{-2}$	0.41
$1 \cdot 10^6$	$4.72 \cdot 10^{-3}$	0.47	$5.95 \cdot 10^{-3}$	0.47

smooth-ufmm

n_{real}	$\|\mathbf{C}_{real} - \mathbf{C}\|_{2,rel}$	roc_2	$\|\mathbf{C}_{real} - \mathbf{C}\|_{\infty,rel}$	roc_∞
$1 \cdot 10^3$	$1.32 \cdot 10^{-1}$		$1.21 \cdot 10^{-1}$	
$1 \cdot 10^4$	$3.72 \cdot 10^{-2}$	0.55	$3.95 \cdot 10^{-2}$	0.48
$1 \cdot 10^5$	$1.39 \cdot 10^{-2}$	0.43	$1.35 \cdot 10^{-2}$	0.47
$1 \cdot 10^6$	$4.45 \cdot 10^{-3}$	0.49	$5.69 \cdot 10^{-3}$	0.37

Table 7.1: Error (and rate of convergence, roc) on the covariance matrix $\mathbf{C}$ ($n = 72.000$) *w.r.t.* the number of realizations n_{real} for a Gaussian correlation with $\ell = 0.25$ (top) and $\ell = 0.50$ (bottom). The approximate square root used to generate realizations is computed using a fixed accuracy randomized SVD ($\varepsilon = 10^{-2}$) powered by *ufmm* (left) or *smooth-ufmm* (right) with $p = 3$. The sample covariance $\mathbf{C}_{real}$ is computed from the n_{real} realizations as (7.15). Observations: The convergence rate is close to the theoretical rate, *i.e.*, $1/2$. The rate slightly decreases as n grows, since $\tilde{\mathbf{C}}$ fails to represent $\mathbf{C}$ with sufficient accuracy, though the sample covariance accuracy is already very satisfying, namely below 10%.

million points. Fast low-rank variant of those algorithms often require the efficient computation of matrix square root, namely Ensemble Kalman Filters. Another useful application of our algorithm called *denoising* consists in removing correlated noise from a given random field using the inverse square root of the covariance matrix.

$\mathcal{H}^2$-powered Randomized SVD Our approximate SVD algorithm showed good performance even for matrices of relatively high rank and can partially overcome issues related to matrices with relatively flat spectrum. This contribution is a rare attempt at accelerating random projection based Low-Rank Approximation (LRA) algorithms using the block low-rank structure of the input matrix, which proved its efficiency on artificial test cases and should now be applied to real-life applications. Finally, since many covariance matrices arising in geostatistical applications are computed purely algebraically, *e.g.*, from the experimental field, one should consider using the general $\mathcal{H}^2$ format in order to derive linear time factorization algorithm.

8

A geometric view on biodiversity

Contents

Covariance matrix computations are involved in numerous fields of scientific computing such as geostatistics, computational economics, computational biology, machine learning,… In geostatistical applications such as the one addressed in Chapter 7 the covariance matrix is often given as a smooth and fast decreasing kernel with nice low-rank properties, that allows for Fast Multipole Method (FMM)-powered matrix computations, *e.g.*, multiplication, matrix decomposition, square root, inversion,… In the general case where the covariance matrix is only known algebraically, *e.g.*, assembled from the data, it often exhibits a certain structure, that can be exploited in order to improve the performance of the matrix computations. In this chapter, we focus on a dimensionality reduction algorithm known as Multidimensional Scaling (MDS), that allows for visualizing a point cloud in a low dimensional subspace by the only knowledge of the set of pairwise distances. In particular, we want to make this method tractable to large biological datasets, where distances are computed as dissimilarities between DNA sequences. This application was addressed for the following reasons:

- Extend the capabilities of our randomized LRA library to arbitrary covariance matrices.
- Design an operational tool for performing classification on large real life datasets provided by New Generation Sequencing (NGS).
- Provide new materials and design new metrics for investigating new ways of characterizing the biodiversity.

We first describe the standard MDS approach as well as some specificities of biological datasets. Then, present a new approach based on random projection and discuss its complementarity with the state-of-the-art method. Finally, we provide some visuals in order to better understand what useful outputs our method can provide to perform classification on large datasets.

8.1 Multidimensional Scaling on biological datasets

In this section we present the fundamentals of MDS and discuss some key issues related to error estimation. Then, we put the emphasis on the computational aspects and we highlight the limits of the standard fast MDS algorithm. Finally, we discuss some specific issues related to the operational chain addressed in this thesis.

8.1.1 Multidimensional Scaling

The method implemented here is the *metric* MDS, please refer to [66] for a recent survey and [35] for a seminal monograph.

Metric MDS Let us have a set of n items i with $i \in E = \{1, \dots, n\}$, and a distance D_{ij} between items i and j, then $(E, \mathbf{D})$ is a finite metric space. Let $r \in \mathbb{N}$ and $\mathbb{R}^r$ have a bilinear form B with an associated quadratic form q. Then, $(E, \mathbf{D})$ is embedded into $(\mathbb{R}^r, B)$ if there exists a map $E : i \longrightarrow \mathbf{X}_i \in \mathbb{R}^r$ such that

$$D_{ij}^2 = q(\mathbf{X}_i - \mathbf{X}_j). \tag{8.1}$$

If $(\mathbb{R}^r, B)$ is an Euclidean space, this is the classical

$$D_{ij}^2 = \|\mathbf{X}_i - \mathbf{X}_j\|_2^2. \tag{8.2}$$

For a given dimension $r \in \mathbb{N}$, the *metric* MDS consists in finding a map such that $\|\mathbf{X}_i - \mathbf{X}_j\|_2$ is as close as possible to D_{ij}. The solution is well-known when B is definite positive [35] and it is implemented in three steps:

1. Pre-treatment: We denote $\mathbf{S}$, the similarity matrix associated with B,*i.e.*,

$$S_{ij} = B(\mathbf{X}_i, \mathbf{X}_j) \tag{8.3}$$

   whose entries can be computed from the distances only as

$$B(\mathbf{X}_i, \mathbf{X}_j) = -\frac{1}{2}\left(D_{ij}^2 - \frac{1}{n}\sum_k D_{kj}^2 - \frac{1}{n}\sum_k D_{ik}^2 + \frac{1}{n^2}\sum_{k,\ell} D_{k\ell}^2\right) \tag{8.4}$$

2. Factorization: Compute the eigenvalue decomposition of S

$$\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\mathbf{T} \tag{8.5}$$

3. Components: Represent the point cloud by the components of

$$\mathbf{X} = \mathbf{U}\mathbf{\Lambda}^{\mathbf{1/2}} \tag{8.6}$$

It is straightforward that $\mathbf{X}$ verifies $\mathbf{S} = \mathbf{X}\mathbf{X}^\mathbf{T}$, in fact

$$\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\mathbf{T} = (\mathbf{U}\mathbf{\Lambda}^{\mathbf{1/2}})(\mathbf{U}\mathbf{\Lambda}^{\mathbf{1/2}})^\mathbf{T} = \mathbf{X}\mathbf{X}^\mathbf{T} \tag{8.7}$$

If all eigenvalues are strictly positive, then $(E, \mathbf{D})$ can be embedded isometrically in a Euclidean space, and a best low-dimensional approximation can be computed. The accuracy of this map is usually quantified by the error committed on the distance reconstruction, which reads as

$$\delta(\mathbf{X}) = \frac{\|\mathbf{\Delta}^2(\mathbf{X})\|_F}{\|\mathbf{D}^2\|_F}, \tag{8.8}$$

where the entries of the distortion matrix $\Delta^2(\mathbf{X})$ read as

$$\Delta^2_{ij}(X) = D^2_{ij} - \|\mathbf{X}_i - \mathbf{X}_j\|^2_2, \forall i, j = 1, \dots, n. \tag{8.9}$$

Negative eigenvalues However, in general, not all eigenvalues of $\mathbf{S}$ are strictly positive, since the input distance is not Euclidean. This means that $\sqrt{\mathbf{\Lambda}}$ may have imaginary parts, which may induce extra distortion in the point cloud and affect the quality of the reconstruction. In fact, it is possible to show that there exists an isometry into a pseudo-euclidean space, *i.e.*, a vector space with a bilinear symmetric form, with positive and negative eigenvalues (see [58] and [99] with references therein). Then, there exists two Euclidean spaces E^+ and E^- of dimensions r^+ and r^- respectively if the bilinear form is of signature (r^+, r^-), such that $D^2_{ij} = \|\mathbf{X^+}_i - \mathbf{X^+}_j\|^2_2 - \|\mathbf{X^-}_i - \mathbf{X^-}_j\|^2_2$. Let $\mathbf{\Lambda}^+$ be the diagonal matrix of strictly positive eigenvalues, and $-\mathbf{\Lambda}^-$ of strictly negative ones. Let $\mathbf{X}^+$ be the $n \times r^+$ matrix of images in E^+, and $\mathbf{X}^-$ of images in E^-. Then, $\mathbf{X^+ = U^+(\Lambda^+)^{1/2}}$ and $\mathbf{X^- = U^-(\Lambda^-)^{1/2}}$. Hence, the distortion matrix can be expressed on both E^+ and E^- as

$$\Delta^2_{ij}(X^+, X^-) = D^2_{ij} - \|\mathbf{X^+}_i - \mathbf{X^+}_j\|^2_2 + \|\mathbf{X^-}_i - \mathbf{X^-}_j\|^2_2, \forall i, j = 1, \dots, n \tag{8.10}$$

or on E^+ only as

$$\Delta^2_{ij}(\mathbf{X}^+) = D^2_{ij} - \|\mathbf{X^+}_i - \mathbf{X^+}_j\|^2_2, \forall i, j = 1, \dots, n. \tag{8.11}$$

Finally, we can show that for an exact representation of the similarity matrix, *i.e.*, $\mathbf{X}_i \in \mathbb{R}^n$, the distortion associated to the representation in E^+ defined as

$$\delta(\mathbf{X}^+) = \frac{\|\mathbf{\Delta}^2(\mathbf{X}^+)\|_F}{\|\mathbf{D}^2\|_F} \tag{8.12}$$

is related to the energy of the matrix associated to negative eigenvalues ε^- by the following expression

$$\delta^2(\mathbf{X}^+) = \frac{\varepsilon^-}{\varepsilon^+ - \varepsilon^-}, \text{ with } \varepsilon^- = \frac{\sum_{i=1}^{r^-}(\lambda_i^-)^2}{\sum_{i=1}^{n}(\lambda_i)^2}, \tag{8.13}$$

In the case of a rank-r approximation this relation is more complicated and will be analyzed experimentally in this chapter.

Other factorizations Many matrix decomposition techniques (Cholesky Decomposition, Singular Value Decomposition (SVD), ...) can provide a valid factor $\mathbf{X}$, *i.e.*, real valued components, as long as the matrix is *spsd*. Since $\mathbf{S}$ is in general not strictly *positive definite*, the Cholesky Decomposition does not apply. However, $\mathbf{S}$ is symmetric, thus a factorization equivalent to (8.5) is achievable using an SVD, which takes the following form

$$\mathbf{S = U\Sigma V^T} \tag{8.14}$$

where $\mathbf{V}$ is also an orthonormal matrix and the singular values σ_i need be positive and verify

$$\sigma_i = |\lambda_i|, \forall i = 1, \dots, n \tag{8.15}$$

If $\lambda_i < 0$ then the i-th column of $\mathbf{V}$ integrates this negative sign. Hence, if the factorization is obtained in SVD form then the conversion to Eigen Value Decomposition (EVD) form is straightforward.

8.1.2 Limits of standard techniques and alternatives

As far as calculation is concerned, the most demanding step is step 2, *i.e.*, computing the eigenvectors and eigenvalues of $\mathbf{S}$. As mentioned numerous times in this thesis, the cubic cost of direct approaches such as full SVD is intractable to large sample sizes. Although there exist iterative algorithms for computing EVD, *e.g.*, Arnoldi, randomized low-rank approximation techniques usually perform better and offer more flexibility, while reaching near optimal accuracy for most matrices involved in practical applications.

Column selection The usual work around in *metric* MDS is to consider subsets of the rows and columns of the input matrix, then perform the MDS on the resulting subsample and finally use a matrix factorization technique known as the Nyström method [121] to reconstruct the entire point cloud. The standard Nyström approach consists in first sampling c columns from $\mathbf{S}$ and building an approximation of $\mathbf{S}$ as

$$\mathbf{S} \approx \mathbf{C}\mathbf{W}^{\dagger}\mathbf{C}^{\mathbf{T}} \tag{8.16}$$

where $\mathbf{C} \in \mathbb{R}^{n \times c}$ gathers the sampled columns and $\mathbf{W}^{\dagger}$ is the Moore-Penrose pseudo-inverse of $\mathbf{W}$ the matrix of sampled rows and sampled columns. Hence, Nyström based MDS provide an approximate map, that reads as

$$\mathbf{X} \approx \mathbf{C}\mathbf{W}^{\dagger/\mathbf{2}} \tag{8.17}$$

Many fast MDS variants derive from that approach, *e.g.*, Landmark MDS [39, 40], *FastMap* [53] or *MetricMap* [116]. More precisely, each approach differs from the other by the variant of the Nyström method it implements, namely the way it selects the columns and/or builds the factorization. Nyström based MDS is usually very convenient for biological applications since it offers interpretability, *i.e.*, easy connection between components and individuals, and very competitive performance. However, it presents some significant drawbacks, that we discuss in the following.

Limitations First of all, the Nyström factorization only applies to *spsd* matrices. As stated in [56], although the extension of Nyström approach to non-*spd* similarity matrices was addressed in [13], the alternatives usually lack stability. Moreover, the design of efficient non-negative embeddings is not the purpose of this work, but the reader will find a

fair amount of articles on that topic under the name *non-metric* MDS. Second of all, as shown in [101], certain variant like Landmark MDS and *FastMap* may produce negative eigenvalues. Finally, it is usually very involved to predict or even just monitor the number of selected rows/columns to achieve satisfying accuracy on the rank r approximation of a given matrix, not to mention the variability of the approximation error. An improved Nyström factorization was proposed in [119] and designed to provide better accuracy, that reads as

$$\mathbf{S} \approx (\mathbf{C}\mathbf{C}^{\dagger})\mathbf{S}(\mathbf{C}\mathbf{C}^{\dagger})^{\mathbf{T}} \tag{8.18}$$

However, as mentioned in [57], this variant is just another instantiation of the random projection based Low-Rank Approximations (LRAs) described in [64]. In particular, it has a similar computational cost without the many benefits of random projection described in the present chapter.

Random projection Although the concept of random projection is intrinsically related to dimensionality reduction through *Johnson-Lindenstrauss Lemma* [68], to our knowledge there are very few contributions to MDS that involve neither sparse or dense random projection if any. Moreover, since random projection belongs to a broader class of sketching techniques, including random sampling to some extent, it offers a large variety of fast alternatives. In fact, many random projection techniques such as the original Fast Johnson Lindenstrauss Transforms (FJLT) [5], rely on sparse or very sparse projection [3], which is in essence very close to column selection and performs similarly. On the other hand, dense random projection based LRAs [64, 45, 115] represent significant alternatives to iterative EVD algorithms, as they share the same complexity, namely $\mathcal{O}(rn^2)$, but usually perform significantly better. Despite involving more intensive computations than sparse projection or random sampling they have many benefits such as better accuracy, robustness and low variability. Furthermore, they provide factorization in many practical forms such as SVD, Cholesky Decomposition or Interpolative Decomposition, and they can be tuned to reach near optimal accuracy at a relatively low extra cost. Finally, since the efficiency of the random projection based LRA algorithms mostly depend on the ability to apply fast matrix multiplication, it can be improved by exploiting the structure of the matrix and it can be parallelized straightforwardly.

8.1.3 Origin and nature of the data

Workflow We integrated our new algorithm in an operational chain that consists in the three following steps:

- Compute distance matrices from sequences provided by NGS and store them on *iRods*. Please refer to [55] for further details on the pairwise distance computations.
- Transfer the distance matrix $\mathbf{D}$ associated with a sample to *plafrim2*, perform MDS on the full sample or subsamples and store associated components $\mathbf{X}$.

- Post-treat components on a local or a cluster computer and provide some data to be analyzed, namely statistics and visuals.

We mostly contributed to the improvement of the second step, while focusing on providing relevant information for the third step that can be analyzed easily. In this section, we define the goals of our contribution in terms of performance and storage improvement, error estimation and visualization. In particular, we discuss the specificity of the data involved in this chain and illustrate it on small subsets of full samples for which exact full SVD is affordable.

Sample (Lake Geneva)	compute distances S-W on *Turing*	transfer and convert *iRods→plafrim*	subsample 10^4 reads	perform MDS full SVD
10^5 reads	4h	20min + 1h40	30s	20min

Table 8.1: Some average running times related to expensive operations involved in the operational chain. Observations: The large amount of time required to convert distance matrices from text files to binaries actually represent a significant gain compared to the initial state of the chain (text files read from python). In fact, since operations such as subsampling may be repeated numerous times for a given sample, it is recommended to store the full distance matrix on the disk in a format that allows for a fast loading of the full sample in memory.

Samples The samples studied in this thesis were collected in Lake Geneva at various time $t = 1, \dots, 10$ of the year, that we denote L_t. They correspond to diatoms scraped from 5 stones each month between April 2012 and March 2013 in order to investigate a seasonal dynamics. Some individuals have previously been identified while other remain unknown. Table 8.2 shows the size of the different samples involved in this chapter, that go from about 70.10^3 to 140.10^3 reads. Therefore, samples can have quite different sizes. We expect the point clouds to have different shapes while always present clusters and comparable numerical rank.

sample name	L_1	L_2	L_3	L_4	L_5	L_6
#sequences	72083	98492	72897	136450	75218	99594

Table 8.2: Size of diatoms samples from Lake Geneva at various time of the year. Observations: The number of sequences, *a.k.a.*, reads, can vary by a factor of 2, while the average number of reads is approximately 100.000.

Distances & Similarities As opposed to the previous applications considered in the thesis, where the point cloud is known a priori (or generated randomly) and the interactions are computed using kernel functions, here the data are given as raw distance matrices, *a.k.a.* dissimilarity matrices. There are several ways of comparing DNA sequences (see [61, 126]), the dissimilarity used here is computed from a local alignment score [61, p. 232] using Smith-Waterman algorithm [109]. The entries of the distance matrices have a very low resolution, namely 0.05 for a maximum value below 650. In particular, the distance matrices can be multiplied by 100 and stored as short unsigned integers. Then, the similarity matrix S can be assembled very efficiently using single precision arithmetics and then stored in memory using single float. For $n = 10.000$ the similarity matrix is computed in 0.2s and stored

using 400MBytes, therefore a full sample takes about 40GBytes in memory, which fits the 128GBytes available on *plafrim2/miriel* nodes and leaves room for storing a large number of components, namely up to $r = n$.

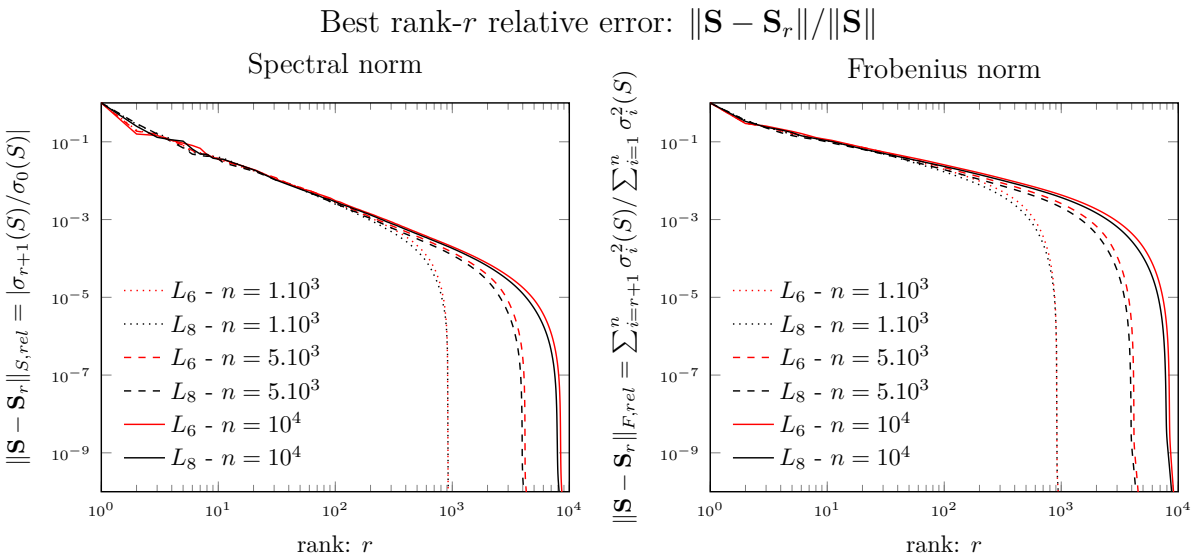

Figure 8.1: Best rank-r error *w.r.t.* the rank in term of Frobenius (right) and Spectral (left) norm for a subset of a full sample from Lake Geneva at various times. $\mathbf{S}_r$ is obtained by mean of a full SVD, therefore we can only afford to compute small subsamples, namely up to $n = 10^4$.

Spectrum In the following we will mostly focus on the properties and the low-rank structure of the similarity matrix $\mathbf{S}$, that will often be refer to as a similarity or a covariance matrix, since it represents correlations between sequences of the sample. In order to design an efficient LRA technique, we first need to ensure that the similarity matrix fulfills the hypothesis of *low numerical rank*, *i.e.*, $r << n$. The numerical rank is usually defined in term of the reconstruction error on the covariance and cannot be lower than the best rank-r error provided by the SVD. As shown on Figure 8.1, for subsamples up to 10.000 reads selected from a full sample at various times, the reconstruction error reach a value of 10^{-3} in Spectral norm and about 10^{-2} in Frobenius norm for a rank of about $r = 200$, *i.e.*, 2% compression. However, one needs to set the rank to $r \approx n/2$ in order to reach a relative error of 10^{-3} in Frobenius norm. Moreover, the best rank-r relative errors of all subsamples (of any size and origin) almost coïncide for the first $r = 100$ eigenvalues. In particular, the similarity of all subsamples can be approximated with $r = 10$ within 10% of accuracy in Frobenius norm and $r = 40$ within 0.1% of accuracy in Spectral norm. These remarks on the numerical rank will be confirmed in Section 8.2 on larger subsamples and full samples. Finally, despite being rather flat, the spectrum of the similarity matrices associated to our samples decreases sufficiently fast (Figure 8.1 left) to be well approximated by random projection.

Distortion A key issue in our application is to ensure a certain accuracy when reproducing the point cloud, that we quantify in term of the distortion $\delta(\mathbf{X})$, *i.e.*, the representativity error of the computed components on the distance. First of all, the numerical rank should

ideally be determined *w.r.t.* to the distortion, thus providing an application-oriented criterion. Second of all, the relation between the distortion and the accuracy of the low rank approximation of the similarity matrix should be investigated in order to set the target accuracy of our LRA algorithm *w.r.t.* to the target distortion. Figure 8.2 shows the evolution of the distortion *w.r.t.* to the rank r for the first 1.000 entries of a full sample. We can observe that the distortion decreases then reaches a plateau at about $r = 10$, where it stays slightly inferior to 10%, and then decreases again until it reaches its minimum value, namely about 0.1 to 0.01%. If 10% distortion seems sufficient then only 20 components can be used to represent the point cloud, for better accuracy one should consider much larger r, namely between 200 and 500. On the other hand, if we look at the fraction of the energy of the similarity matrix associated to negative eigenvalues, we observe that it grows fast once the first negative eigenvalue occurs. However, the relative contribution of the negative eigenvalues to the Frobenius norm, defined as

$$\varepsilon_r^- = \frac{\sum_{i=1}^{r^-}(\lambda_i^-)^2}{\sum_{i=1}^{r}(\lambda_i)^2}, \tag{8.19}$$

stops increasing after $r = 200$, where it reaches its maximum value, namely 5 to 6%. This illustrates the correlation between distortion and negative singular values mentioned earlier on.

Coherence A crucial information when performing subsampling is the *coherence* of the matrix, please refer to [111, 91] for nice introductions to the concept of matrix coherence and its relation to the performance of the Nyström method. If each column contains the same amount of information on the matrix, *i.e.*, information is spread equally throughout the columns, then the matrix is incoherent. As a result, columns can be sampled uniformly at random in order to accurately represent the full matrix. On the other hand, if only certain columns concentrate valuable information on the full sample, then the matrix is coherent and columns should be selected more carefully. In order to quantify the importance of a column we usually associate the i-th column with a *leverage* score ℓ_i^r for $i = 1, \ldots, n$ defined as the Euclidean norm of the rows of the singular vectors of the input matrix, namely

$$\ell_i^r = \|\mathbf{U}_i\|_2 = \sqrt{U_{i1}^2 + \ldots + U_{ir}^2}. \tag{8.20}$$

In particular, the leverage scores quantify the correlation between the columns and the basis formed by the first r eigenvectors. A common technique to select columns properly for coherent matrices is based on random sampling with probability proportional to the leverage scores [79, 57], hence the higher the score of a column the higher the probability to sample that column. However, computing the leverage scores of a matrix requires computing the rank-r SVD of the input matrix, which is in general not affordable. A common alternative is approximating those scores by mean of a Randomized SVD (rSVD) [46], which make the design of an efficient rSVD even more relevant for MDS applications. On the other hand,

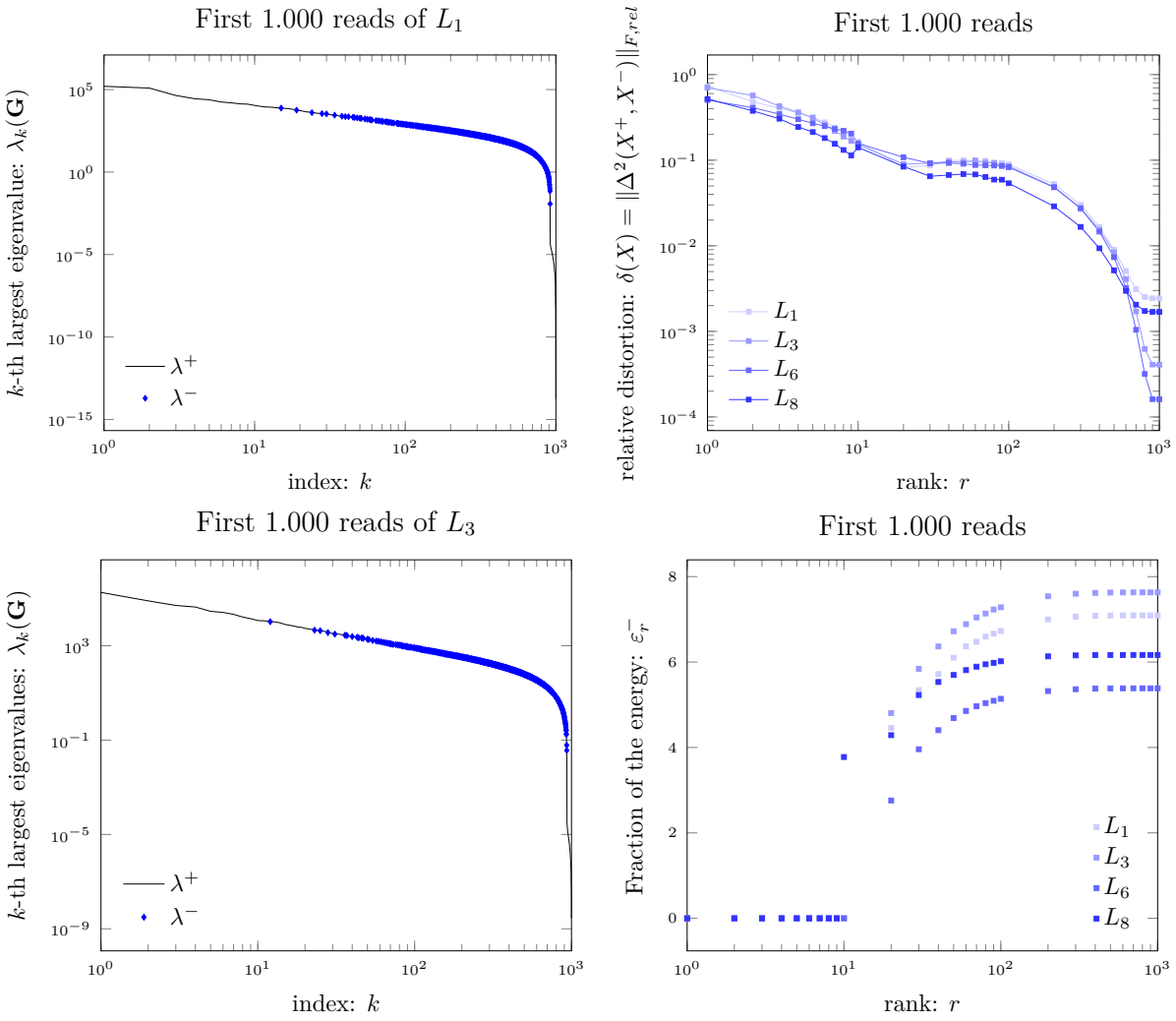

Figure 8.2: Left: Eigenvalues of the matrix associated with the first reads of sample L_1 (top) and L_3 (bottom) with highlight on the negative eigenvalues (blue diamonds). Right: Distortion (top) and contribution of the negative eigenvalues to the Frobenius norm (bottom) *w.r.t.* number of components or rank r of the Truncated SVD (tSVD) for various sample L_t. Observation: The correlation between negative eigenvalues and distortion is illustrated by the regime $r = 10, \ldots, 100$. In fact, the distortion first decreases with the rank then as negative eigenvalues occur the distortion is affected and reaches a plateau before decreasing fast again.

there exist other methods to select relevant columns in a matrix that perform relatively good in practice, *e.g.*, deterministic approach based on the pivoted QR Decomposition [50], semi probabilistic approaches on k-cluster partitioning [4] or combination of randomized and deterministic algorithms [20]. The matrices studied here are expected to have low coherence since all individuals are supposed to have similar significance, as a matter of fact their presence in the sample is inherently random. This assumption will be verified in Section 8.2 since the computation of the coherence of the full sample can only be achieved by mean of a randomized SVD.

Visualization The main goal of our approach is to achieve an efficient and handy visualization technique that provides relevant information on some real-life samples of about 100.000 sequences at various time of the year, namely about 10 measures corresponding to different months. First of all, we would like to be able to characterize the shape of the point clouds in the first few dimensions, identify potential clustering and relate clusters to identified species. Second of all, we want to compare these shapes between various times individually or using cross correlations. Finally, we would like to provide handy visualization technique to characterize clustering in larger dimensions.

8.2 Random projection aided MDS

In this section, we present our main contribution, namely a random projection based MDS algorithm, that allows for treating samples up to several hundred thousand reads on a single node in reasonable time, namely within a few minutes. The algorithm also allows for :

- computing distortion compared to the input distance,
- approximating the spectrum of the matrix,
- and evaluating the coherence of the matrix to be used for improving a subsequent subsampling.

We first discuss the benefits of random projection and present our approach. Then, we setup the parameters of the algorithm and we analyze the distortion on small subsamples. Finally, we analyze the performance of the algorithm on full samples.

8.2.1 Fast MDS based on random projection

Here we introduce a new random projection based MDS algorithm and show how it can contribute to the design of more efficient randomized MDS algorithm in general. Then, we perform simulations on subsamples with an intermediate size, namely 10, 20 or 30 thousand reads, in order to calibrate the input rank and the tune up parameters.

Random Projection Our prior analysis of the data (Section 8.1.3) showed that similarity matrices involved in our application could be well-treated by random projection in term of both accuracy and performance. In particular, we highlighted the relatively low numerical rank of the similarity matrix $\mathbf{S}$ given the low target distortion, namely about 10%. Furthermore, within this range of accuracy, the spectrum associated to each sample appeared to decrease sufficiently fast and behave independently of the time t or subsample size. Therefore, we have implemented an MDS algorithm relying on a random projection LRA technique, namely the rSVD (Algorithm 6) introduced in Chapter 3. In this approach, the similarity matrix is approximated as follows

$$\mathbf{S} \approx (\mathbf{Q}\mathbf{Q}^{\mathbf{T}})\mathbf{S}(\mathbf{Q}\mathbf{Q}^{\mathbf{T}}), \tag{8.21}$$

namely the projection on a rank-r approximation $\mathbf{Q}$ of its range, itself obtained by dense Gaussian random projection (3.4). While being inherently related to dimensionality reduction, this approach presents many numerical benefits compared to random sampling technique, such as strong theoretical error bounds, near optimal accuracy in practice and good performance, while leaving significant room for improvement. Although the theory behind this approach is relatively involved, the method is easily implemented, stable and rather robust, which makes it both attractive and accessible to computational biologist for instance. Furthermore, the factorization in EVD form provided by the rSVD gives useful information on the structure of the similarity matrix, *e.g.*, coherence, rank and any quantity that can be computed from the approximate eigen-pairs. In particular, we can estimate the reconstruction error in Spectral norm, compute the energy of the approximate similarity or even detect negative eigenvalues and evaluate the associated distortion.

Subsampling As discussed in Chapter 3, the design of an efficient randomized algorithm mostly relies on a good knowledge of the input data and matrices (structure, clustering,…) and it can often benefit from efficient combinations of both random sampling and random projection techniques. In our application, since working on the full sample or on a random subset should in essence lead to similar conclusions for a given sample, it can be very useful to perform a few computations on subsamples before performing computations on the full dataset. This allows us to draw preliminary conclusions on the structure of the similarity matrix in order to ultimately tune up our algorithm. Since our random projection algorithm can be used to efficiently approximate the coherence of the matrix [46], we can verify by analysis of the leverage scores whether or not subsets of the full sample can be selected uniformly at random. If it is not the case, then the algorithm provides useful information to select a more relevant subset. In that sense, our contribution not only provides an alternative to column selection algorithms with many extra benefits, but it could as well help design a more efficient column selection approach.

Fixed rank or accuracy Our new approach involves very little modification on the original MDS algorithm, as it only relies on a new way to perform the EVD of the similarity matrix $\mathbf{S}$. In that sense, the target accuracy or the rank of the algorithm is set in the same fashion as a standard tSVD. In order to validate and calibrate our algorithm it is more convenient to prescribe the number of components that we want to visualize. Therefore, we prefer the fixed rank rSVD. Since we do not want to visualize more than a few hundred components, we will determine the prescribed rank to be the rank that leads to a few percent distortion on a length-n subset of the full sample such that $r << n$. The associated reconstruction error on the covariance matrix will ultimately be taken as input of the fixed accuracy rSVD.

Oversampling and Power iterations As opposed to the tSVD, the rSVD provides approximate rank-r factorization, therefore it needs to be carefully tuned in order to make this approximation *near optimal* at a small extra cost. Figure 8.3 shows the accuracy of the rSVD on the spectrum of the similarity matrix for the sample L_6. The rSVD accurately estimates the first 100 eigenvalues even without oversampling ($s = 0$), but it may fail to identify the last few eigenvalues. The figure also shows that significant oversampling ($s = r$) can substantially improve the accuracy. Although the accuracy still decreases slightly on the last part of the spectrum, the associated eigenvalues hardly contribute to the reconstruction error (Figure 8.1). For instance, for $n = 10.000$ random reads from L_6, the reconstruction error achieved for $r = 200$ equals 2.1% in Frobenius norm while the best rank-r is 1.6%. Finally, simulations showed that this issue can be fixed by a single power iteration, nevertheless we prefer to avoid that and tolerate this small difference in order to develop a faster algorithm.

8.2.2 Validation of the method

Here we validate our method on various subsamples of a given L_t sample, namely L_6. We first show the benefits of using random projection to select a relevant subset of the full sample. Then, we discuss the accuracy of our approach on various subsamples and compare it with full samples.

Subsampling First of all, we want to verify whether or not uniform sampling of the columns applies well to our datasets. Let us analyze the leverage scores associated to the sample L_6. Figure 8.4 shows some statistics such as the mean, variance and extrema of the leverage scores up to $r = 1.000$. In particular, we observe that for an artificial sample with $n = 10.000$ (Figure 8.4 left) the scores are spread equally around the mean and have a rather small variance. This shows that the similarity matrix associated with this sample is *incoherent*, *i.e.*, all columns contain about the same amount of information on the sample. However, the full sample exhibit a different behavior (Figure 8.4 right) in the sense that some leverage scores are significantly larger than the rest and make the distribution deviates from

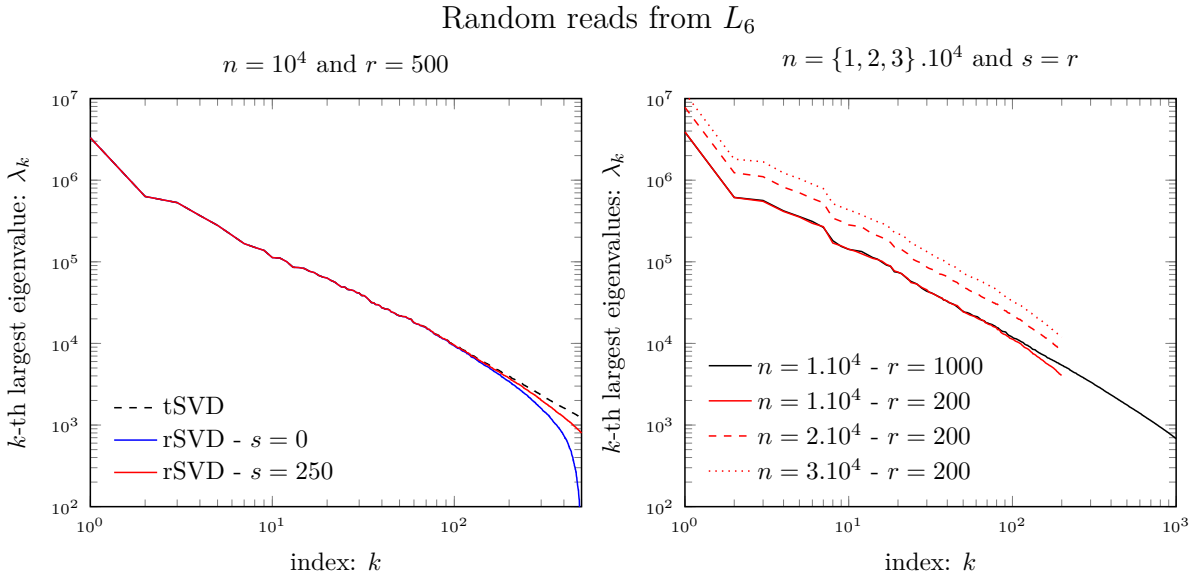

Figure 8.3: Left: First 500 eigenvalues of a random subsample of L_6 compared to approximate values obtained by rSVD without ($s = 0$) or with ($s = r/2$) oversampling. Right: First 200 approximate eigenvalues obtained by rSVD with significant oversampling $s = r$ on larger subsamples: $n = 10.000$, $n = 20.000$ and $n = 30.000$. We use the result of a rSVD with $r = 1.000$ as an overkill to compare the approximate spectrum to a near optimal spectrum, but also to show that the rSVD delivers a result with very low variability. Observations: The approximate spectrum differs slightly from the exact one only for the smallest values, namely $r > 100$. Furthermore, as the size of the subsample n increases, the spectrum gets shifted upwards but still has the same shape, *i.e.*, same reconstruction error, but the distortion may differ.

its mean. On the other hand, the variance remains relatively small. Hence, uniform sampling may still be relevant for selecting columns in a Nyström based approach or simply selecting a relevant subset of the sample in a subsampled random projection approach. But a finer analysis of the leverage scores may lead to a more accurate representation at a lower cost.

Distortion In order to properly set the rank r of the rSVD we need to quantify the distortion on subsamples with intermediate number of reads, namely $n = 10.000$ to $n = 30.000$. Although $r = 200$ components lead to a reconstruction error of the similarity matrix of about 10^{-3} in Spectral norm and 10^{-2} in Frobenius norm (Figure 8.3), it only leads to a distortion of about 12% for 10.000 reads and 17% for 30.000 reads (Figure 8.5). Moreover, for $n = 10.000$ we need to use 5 as much components to reach 8% distortion, *i.e.*, $r = 1000$. Hence, based on the distortion criterion the ranks of our matrices are not extremely low, namely $r \approx n/10$.

8.2.3 Performance on real-life samples

Here we discuss the computational cost of the method in terms of memory and running time. We also discuss the time required to compute various statistics on the point cloud and the similarity matrix.

Memory footprint Obviously, since the distance and similarity matrices are stored and applied in a dense way, the cost is quadratic in n. In particular, storing the distance matrix

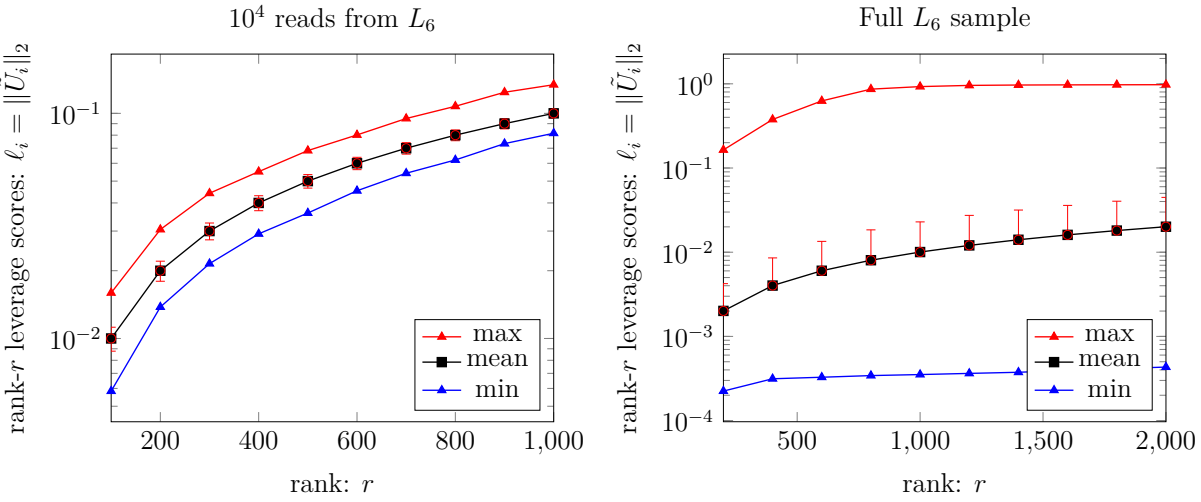

Figure 8.4: Analysis of the coherence of the similarity matrix associated with 10.000 reads from the sample L_6 (left) and the full sample (right). The graphs represent the mean, variance, minimum and maximum values of the approximate leverage scores provided by rSVD *w.r.t.* the input rank r. Observations: For $n = 10.000$ the minimum and maximum values of the leverage scores are relatively close to the mean ($\pm 20\%$), while the variance is very small, which shows that leverage scores are equally spread around the mean and rarely deviate from this value. Therefore, the matrix has low coherence and sampling with probability proportional to the leverage scores boils down to sampling uniformly. On the other hand, the leverage scores associated with the full sample are distributed in a much more complex fashion, which would imply sampling with probability proportional to them.

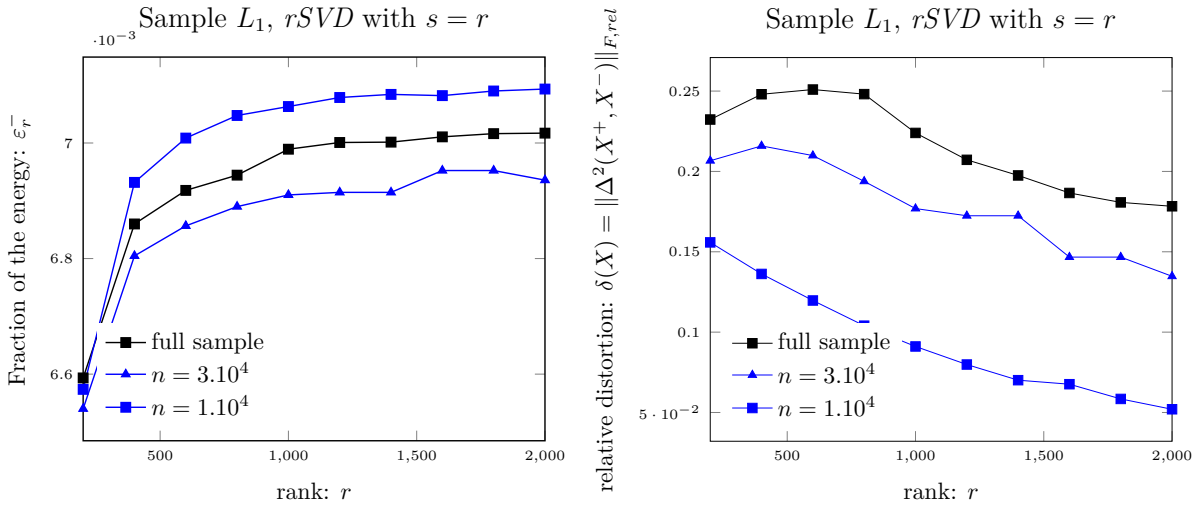

Figure 8.5: Analysis of the point cloud distortion *w.r.t.* the input rank r of the rSVD for 10.000 reads from the sample L_1 or the full sample. Left: Contribution of the negative eigenvalue to the energy of the approximated similarity. Right: Distortion of the point cloud. Observations: The contribution of the negative eigenvalues to the energy of the representation is still around 8% while the proportion of eigenvalues is always around 50%. On the other hand, for a given rank, the distortion grows with the problem size.

using short integers takes about 40GigaBytes for a full sample with 100.000 individuals and the similarity matrix 20GigaBytes. Since the computers used for our simulations, namely *plafrim2/miriel*, have 128GB RAM, the matrices associated with the computation of a full sample are relatively large but still fit in memory and leave significant room for the storage of the components $\mathbf{X}$ even for large r.

Running times Figure 8.6 shows various experimental running times measured during the computation of a subsample with $n = 10.000$ reads and a full sample of about 100.000 reads. First of all, reading the distance matrix and computing the similarity is rather cheap. In particular for the full sample, it requires as much time as computing the rSVD with $r = 200$, namely about 100s. Second of all, Figure 8.6 confirms that the cost of the rSVD grows linearly with the rank and quadratically with n. Third of all, as expected, the computation of the actual distortion of the point cloud representation has a similar complexity as the rSVD, namely $\mathcal{O}(rn^2)$. However, its magnitude is 10 times higher than the rSVD, which makes its computation intractable to large r, *e.g.*, 6hours for $r = 2000$. Hence, a better understanding of the distortion is required here in order to develop estimators for this application before addressing faster factorization techniques. This work will be addressed in a forthcoming article, that introduces our random projection based MDS algorithm. Finally, the computation of the coherence is performed on the full sample by mean of a rSVD and only introduces a small $\mathcal{O}(nr)$ extra cost for computing the leverage scores.

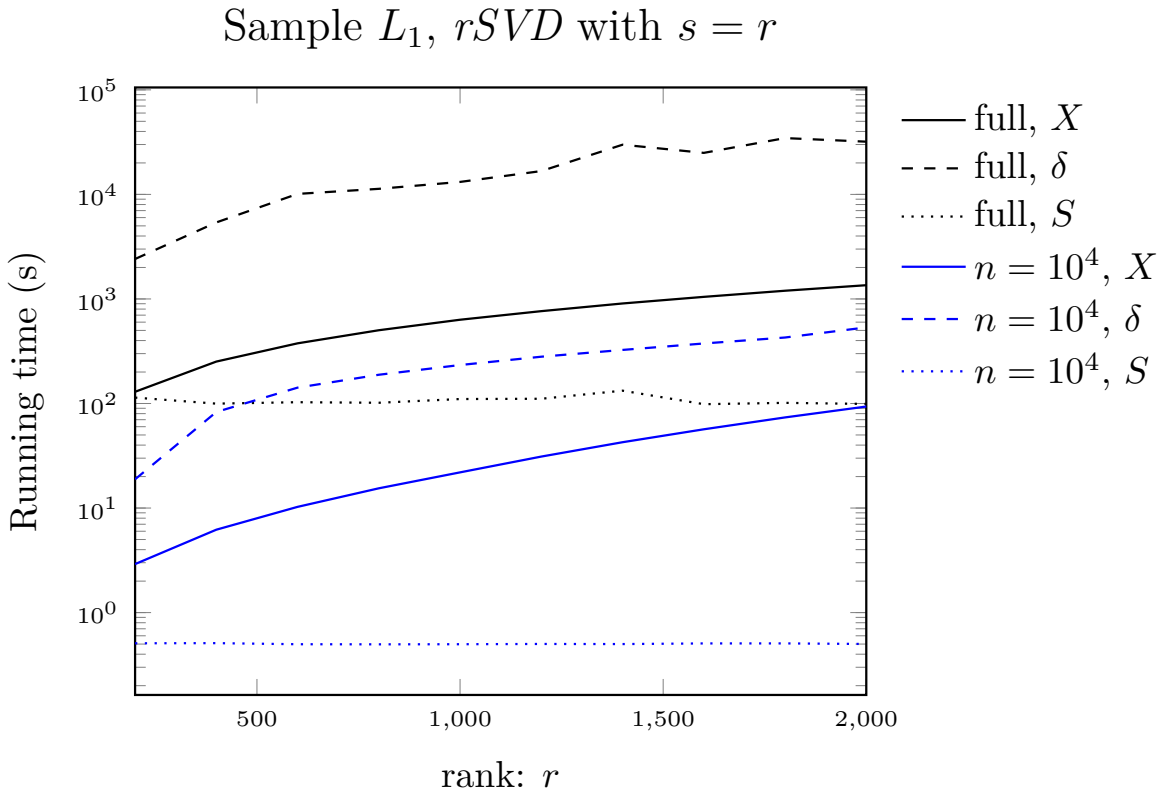

Figure 8.6: Running times of the rSVD (in seconds) *w.r.t.* the input rank r for the full L_1 sample ($n \approx 70.000$) and a subsample with $n = 10.000$. The plot shows the time required to read the distance and compute the similarity matrix S (dotted lines), the running time of the rSVD (solid lines) and the time required to compute the distortion δ (dashed lines).

8.3 Visualization of the point cloud

As shown in Section 8.2, enforcing a few percent distortion as a target threshold of our method can result in relatively high ranks. Although this can still be addressed efficiently using random projection, it is rather demanding in term of error computation. However, the analysis of the components of the point cloud is currently not done in a large number of dimensions. Until high dimension analysis is made possible, visualizing data in small dimensions, *e.g.*, $r \leq 10$, still provides numerous crucial information on the datasets. Here, we show representations of the full samples in 3D for easy visualization of the point clouds and basic comparison of their shapes. Then, we discuss the visualization of a cloud obtained by our method in higher dimensions.

8.3.1 Representation in 3D

The easiest way of visualizing a point cloud is to project the value of the components in 3D. This can be done by mean of three 2D projections on the first feature pairs, namely $(F1, F2)$, $(F1, F3)$ and $(F2, F3)$.

Subsample We first consider the cloud associated with a subsample, namely 10.000 random reads out of L_6. Figure 8.7 represents the position of each individual on the first feature pairs. First of all, we observe that certain regions of the domain associated to a feature pair are more concentrated than others. Second of all, these regions can have various shapes and be well separated from the others, for instance in the plane $(F3, F2)$. Another specificity of this cloud is that a small cluster lies very far from the rest of the cloud in the first dimension $F1$ and is surrounded by all the other individuals in $(F2, F3)$. Finally, such clustering of the data can be better shown by coloring the identified individuals *w.r.t.* their species. Hence, association between known species and features can be better understood. In particular, the presence of individuals from the species *Nitzichia dissipita* in the isolated cluster can probably help understand the meaning of feature $F1$.

Full Sample Let us now consider the cloud associated with the full sample L_6 computed by mean of a rank $r = 50$ rSVD. Figure 8.8 represents the position of each individual on the first feature pairs, while Figure 8.9 represents the concentration of the cloud. First of all, without both coloring and concentration, it would be harder to identify clusters than for the smaller subsample (Figure 8.7). Despite being 10 times as dense, the cloud associated with the full sample has a similar shape than that of the subsample. This can be observed more clearly on Figure 8.9, where the concentration of the population is represented in logarithmic scale. In particular, this figure shows that only very small regions of the full domain contain most of the individuals and are surrounded by regions of fast decreasing population. Second of all, the previous remarks on clustering of known individuals *w.r.t.* their species hold even stronger for the full sample. Moreover, a large part of the cloud contains unknown individuals but still exhibits a certain structure and a significant density,

which suggests a deeper analysis of the data. In particular, a combined analysis of the concentration and the identified individuals seems crucial in order to better characterize regions of high concentration.

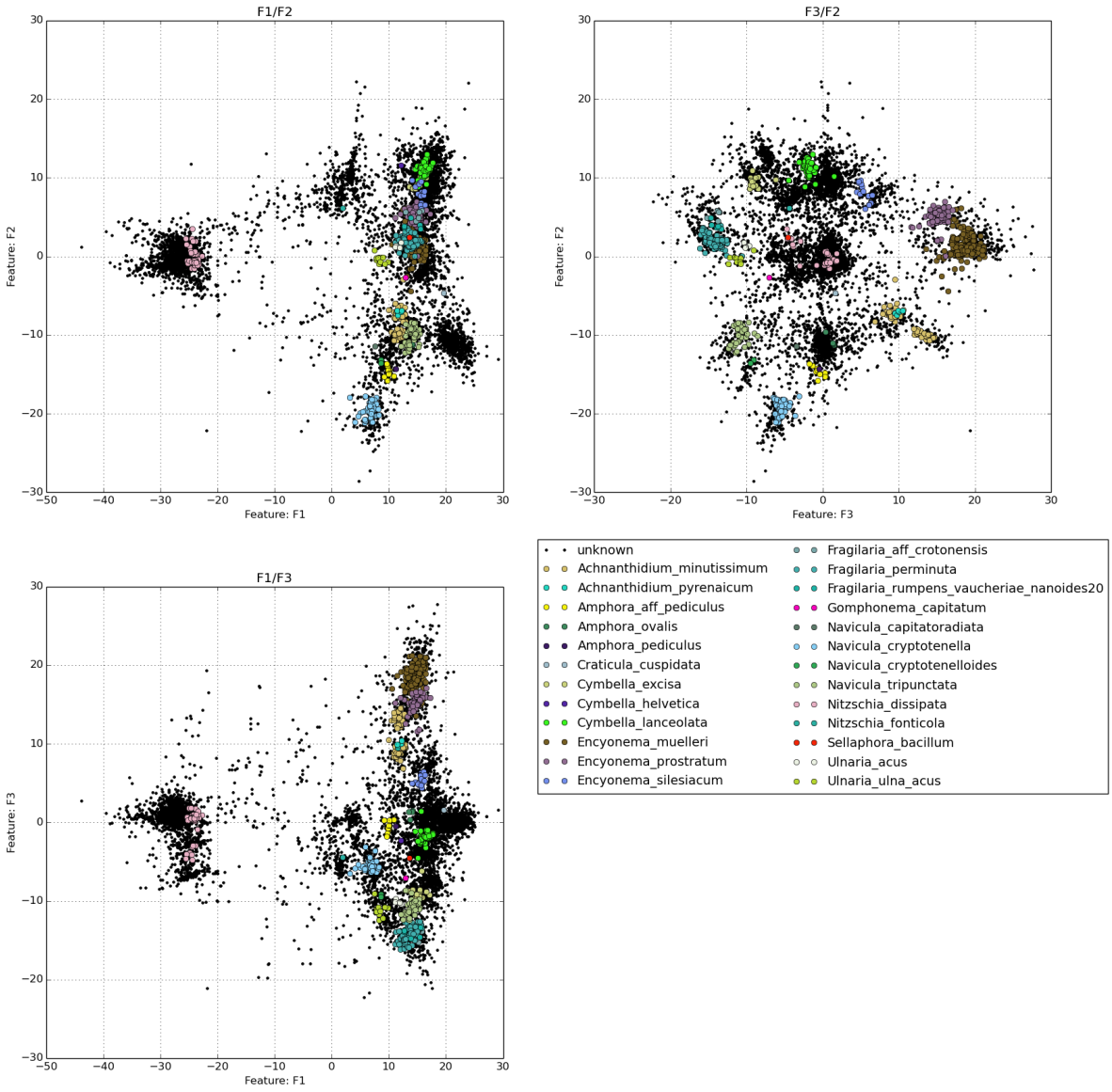

Figure 8.7: Representation of the point cloud associated to 10.000 random reads of the sample L_6 on the first 3 features $F1, F2, F3$ using 2D plots. The components were computed using rSVD with $s = r = 200$. Identified species are represented using large colored markers, while unknown species are represented in black. Observation: Such representation allows for easily verifying that individuals of the same identified species gather in clusters. However random subsampling does not allow to represent all individuals and in particular all the previously identified ones.

Variations in time Let us recall that samples L_1 to L_{10} represent monthly sampling over 10 months at the same location, namely Lake Geneva. Comparison between various samples can be discussed in 3D, where the point clouds exhibit significant differences along time but similarities as well. In particular, we still observe clustering and highly concentrated areas. However, the magnitude of the features, *i.e.*, the size of the domain containing the cloud,

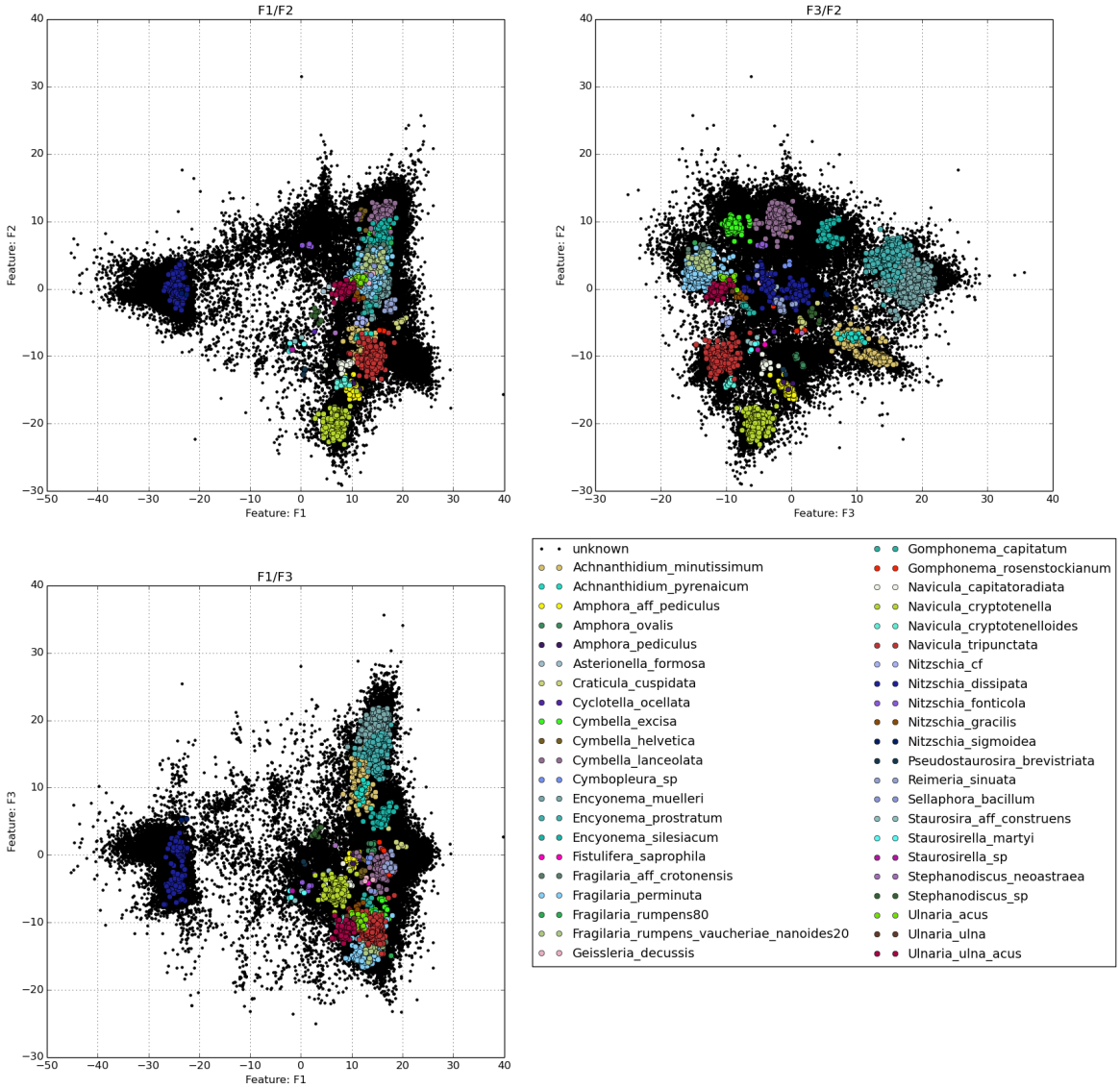

Figure 8.8: Representation of the point cloud associated the full sample L_6 on the first 3 features $(F1, F2, F3)$ using 2D plots. The components were computed using rSVD with $s = r = 200$. Identified species are represented using large colored markers, while unknown species are represented in black. Observation: Such representation allows for easily verifying that individuals of the same species gather in clusters. Moreover, individuals sharing the same root like the Encyonema are themselves located in cluster or at least small bounded domains. Finally, in such low dimensions, the cloud has a similar shape than the subsample with $n = 10.000$ although the distortion is supposed to be a lot higher because of the larger size and the smaller input rank of the rSVD.

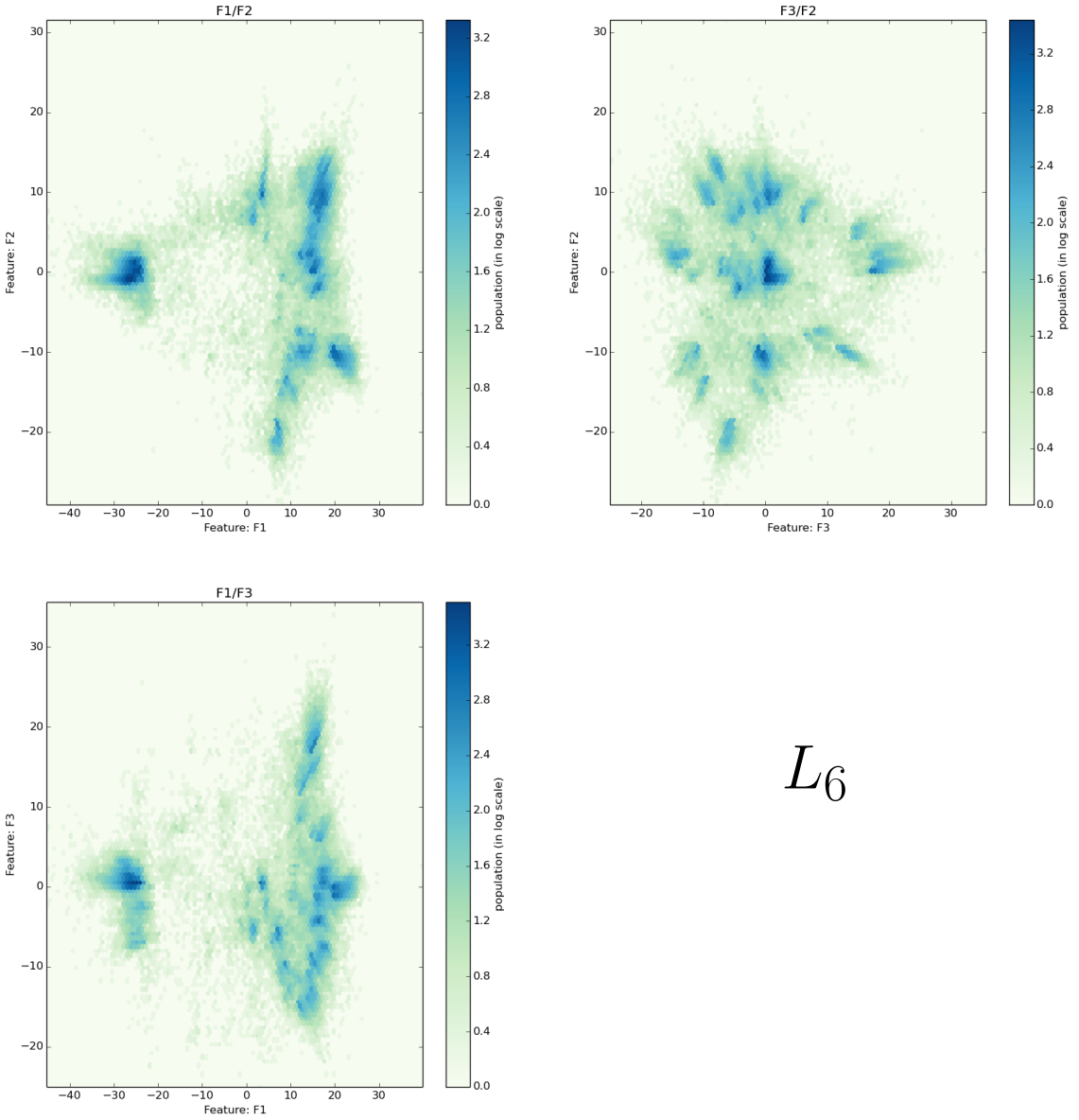

Figure 8.9: Population of the point cloud associated the full L_6 sample represented on the first 3 features $(F1, F2, F3)$ using 2D plots. The components were computed using rSVD with $s = r = 50$. Identified species are represented using large colored markers, while unknown species are represented in black. Observation: Such representation allows for easily verifying that reads concentrate on very specific part of the feature domains. This can provide crucial information on identified individuals but also on unknown ones and may lead to the discovery of new species.

may vary by a factor of 2 to 3 in certain dimensions. For instance, L_1 and L_2 spread over a similar domain, namely a box of size roughly equal to 60 and centered in $(0, 0, 0)$, while L_3 and L_4 spread over a smaller box that is slightly shifted from the origin. Moreover, rotation or translation of the point clouds should be considered in the analysis of the shape as the result of MDS is only defined up to a combination of such transformations.

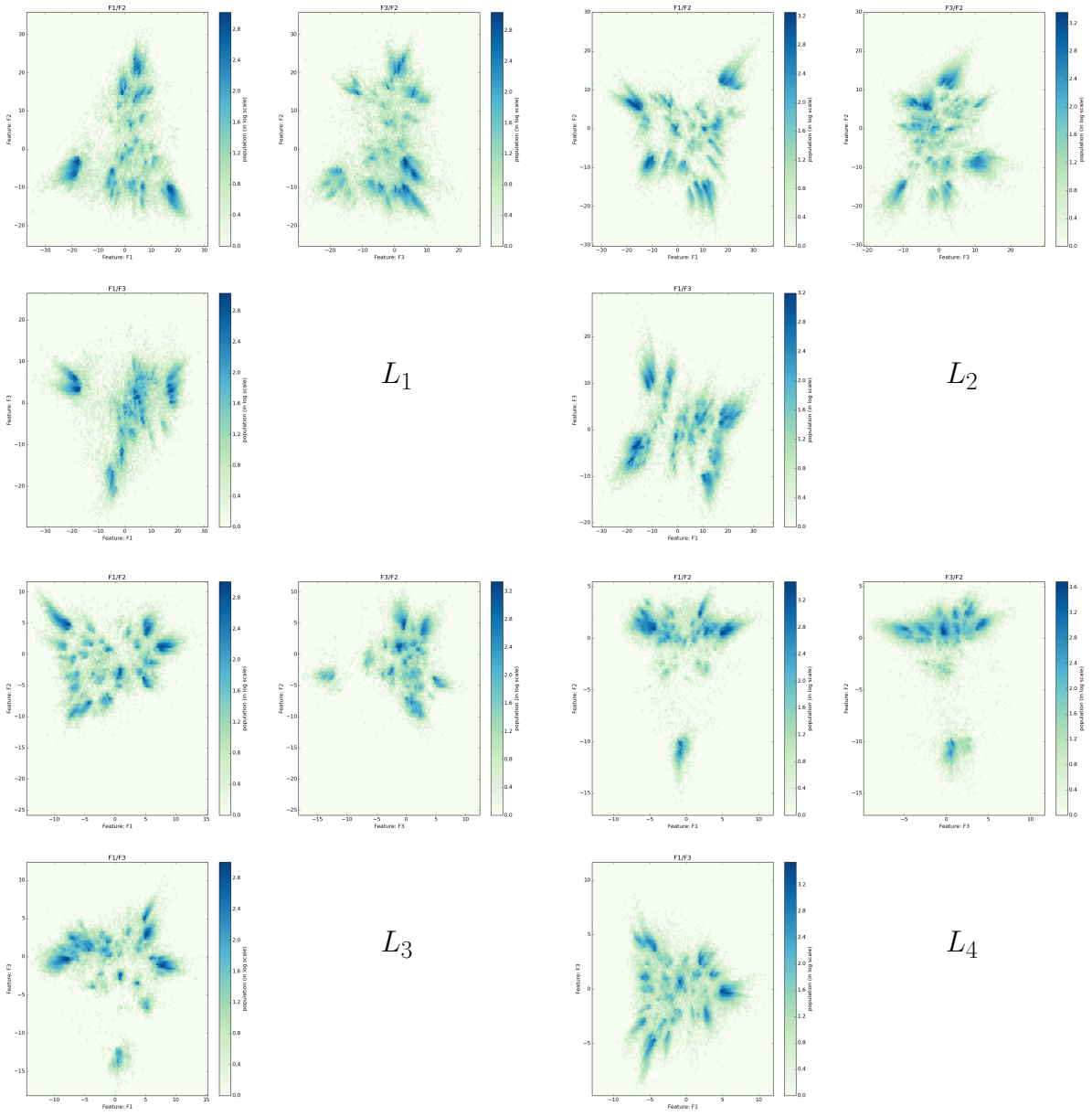

Figure 8.10: Population of the point clouds associated to various samples represented on the first 3 features $(F1, F2, F3)$ using 2D plots. The components were computed using rSVD with $s = r = 50$ on the full samples: L_1 (top left), L_2 (top right), L_3 (bottom left) and L_4 (bottom right). Observation: The shape of the cloud varies significantly along the time, while still presenting similar characteristics such as clear clustering and highly concentrated areas.

8.3.2 Representation in many dimensions

Parallel Coordinates In order to visualize the point cloud in many dimensions a common alternative to 2D plots is the representation in *parallel coordinates* such as the one displayed

on Figure 8.11. This technique offers various advantages as it decouples the features and displays them on a single axis. As shown on Figure 8.11 (top left), the observation of all reads on the same plot without characterization of the concentration cannot be exploited, which suggests the use of a different plotting software with better rendering than *matplotlib* or a finer pretreatment of the components. However, the expected clustering of the reads can be confirmed by observing that individuals of the same species (top right and bottom left), follow a similar path along the first few dimensions, while spreading only on a fraction of the full domain. This observation is made more clear by the observation of each species separately (bottom left). As the index of the feature grows the associated eigenvalue of the similarity matrix decreases, however it has almost no impact on the first 10 features and can only be noticed on more than 100 features.

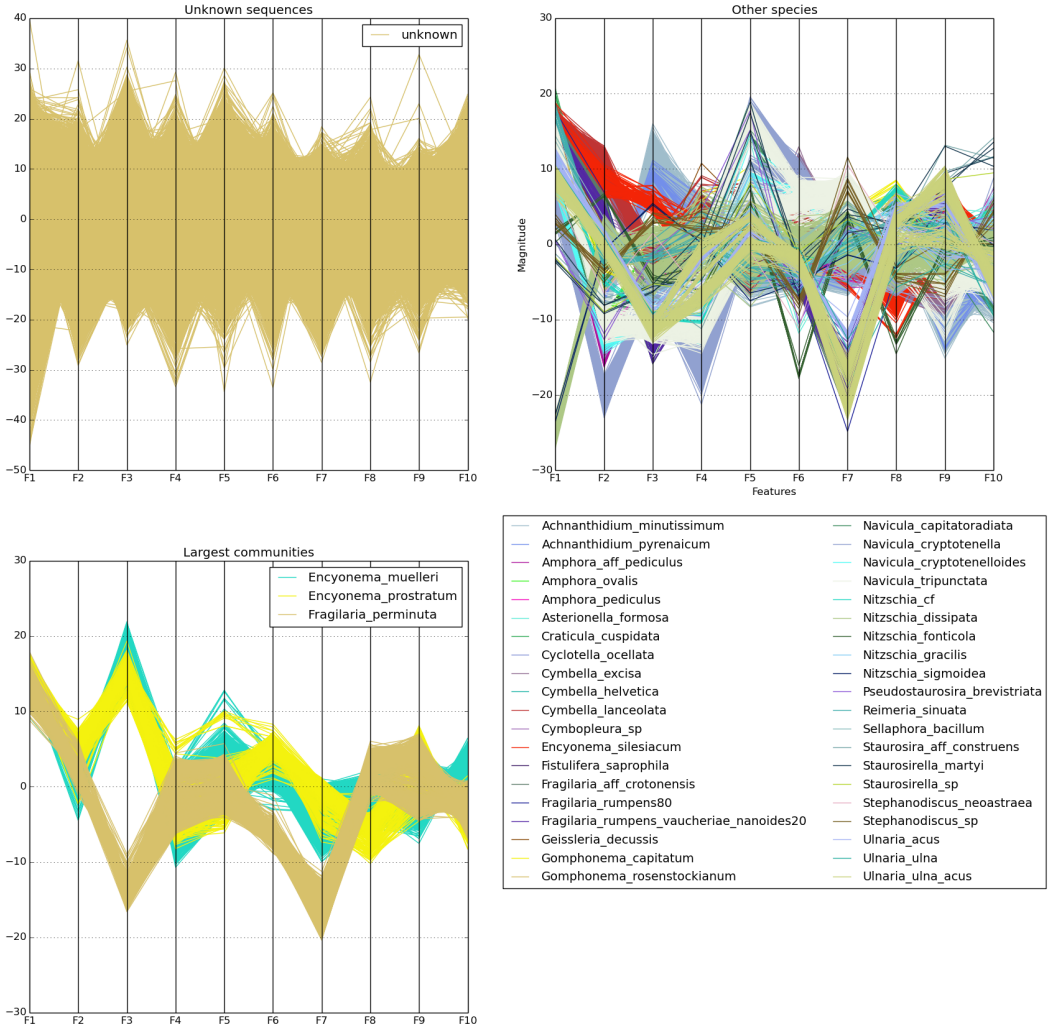

Figure 8.11: Representation of the point cloud associated to the full sample L_6 on the first 10 features $(F1, \ldots, F10)$ using *parallel coordinates* from *matplotlib*. The components were computed using rSVD with $s = r = 50$.

Conclusion

Conclusion

Let us summarize our various achievements and contributions one topic after another.

Fast Multipole Method We developed a highly competitive interpolation based Fast Multipole Method (FMM), called the *ufmm*, relying on a very simple idea and benefiting from the performance of the Fast Fourier Transform (FFT). It provides a kernel independent `M2L`-optimized approach that may suffer from instabilities at very high accuracies but performs significantly better in all cases of interest than the state-of-the-art approaches, namely the *bbfmm* and optimized variants. This method was implemented in the open-source parallel HPC library *ScalFMM* and it is already used by several users for diverse applications such as Molecular Dynamics or Boundary element methods for elastostatics. In this thesis, we used the *ufmm* in various computationally intensive applications with rather low target accuracies in order to demonstrate its good performance and illustrate its genericity.

Dislocation Dynamics We implemented the isotropic elastic force and energy farfield computation based on the *ufmm* and *bbfmm* within *OptiDis*, an operational parallel Dislocation Dynamics (DD) code developed in collaboration with the CEA Saclay. In particular, it already allowed for simulating the phenomenon of clear band channels in Zirconium alloys, as presented in Arnaud Etcheverry's thesis [51]. An efficient summation scheme was proposed along with several variants minimizing the cost of certain operators. We adapted certain features of the code to accurately and efficiently handle segments within an octree structure, exact integration of the farfield as well as the specific nature of the interaction kernel (tensorial, homogeneous, singular,…). In particular, this lead us to develop a generic formalism for tensorial interactions within *ScalFMM* library. Additionally, this allowed us to illustrate the rather high memory requirements of the *ufmm*, compared to the *bbfmm*, while proposing alternative formulations that could compensate this phenomenon. Moreover, this issue arise for high tree depths and only affects the storage of the expansions that is usually spread over several nodes in a shared memory parallelism.

Sampling from GRF A radically different application was addressed, that consisted in accelerating the matrix multiplication involved in a Randomized Low-Rank Approximation (LRA) algorithm by mean of the *ufmm* and thus provide low-rank square root of covariance matrices. This allowed us to extend the capabilities of our algorithm to matrix-to-matrix multiplication using a multi-*rhs* FMM implementation. The matrix studied here were covariance kernel matrices used for sampling from Gaussian Random Field (GRF), *i.e.*, generating multivariate Gaussian Random Variables. Several correlation kernels were tested, such as Gaussian, Exponential and several other Matérn correlations, in order to generate GRFs on artificial heterogeneous distributions of points. We designed a smooth variant of the *ufmm* in order to represent smooth kernels more efficiently. This was tested on the Gaussian kernel but this should have ideally been tested on a less nice kernel, for which there does not al-

ready exist fast algorithms. To our knowledge, this contribution is one of the rare attempts at combining randomized LRA with the data sparse representation of a matrix. The results were promising and setting the FMM accuracy *w.r.t.* to the global low-rank approximation turned out to be extremely easy.

Random projection aided MDS We addressed another shortcoming in randomized numerical linear algebra, namely the lack of fast Multidimensional Scaling (MDS) algorithm based on random projection. Application of a Randomized Eigen Value Decomposition (EVD) on the similarity matrix allowed for deriving an efficient MDS algorithm that provides useful information on the structure of the matrix. This method is complementary with the state-of-the-art method, namely Nyström factorization, while being applicable to any similarity matrix. We provided a discussion and several experiment to relate the distortion with the presence of negative eigenvalue in the spectrum and a best or near-best low-rank representation. The method was applied on biological datasets coming from real-life samples and an operational workflow was organized during the thesis in order to provide visualization data and thus help design a geometric view on biodiversity. The algorithmic and numerical aspects of our random projection based approach to MDS is the topic of a forthcoming paper, that will also present the idea behind this geometric view on biodiversity, with strong references on the theory of dimensionality reduction.

Perspectives

A significant amount of work is still ongoing in order to consolidate these contributions, however our current point of view allows us to draw a considerable number of perspectives.

Uniform FMM The main drawback of the *ufmm* should be dealt with in a near future, namely the relatively high amount of memory required for the storage of the expansions in Fourier domain. A first idea would be to transfer expansions to Fourier domain only when it is actually required. Then, an interesting problem would be the application of the *ufmm* in a directional Fast Multipole Boundary Element Formulation [113, 86] where the cost of the `M2L` can be critical but the interaction lists can be very large as well. Another issue that we need to address would be the regularization of the equispaced interpolation involved in the *ufmm*, as instability are very likely to occur for oscillatory kernels for instance. Additionally, the significant improvement of the `M2L` step should allow the implementation of much complex FMM schemes such as a variable order FMM [112] or $\mathcal{H}^2$-methods [18]. Most importantly, the *ufmm* and its smooth variant, the *smooth-ufmm*, are in essence so close from the general $\mathcal{H}$-matrix formats, therefore we hope that our modest contribution can motivate the design of a unified framework for hierarchical low-rank matrix approximations behind the $\mathcal{H}^2$ format that can benefit to both the FMM and the $\mathcal{H}$-matrix communities. In our opinion, a first step would be for us to compare the performance of our method with

recent interpolation based FMM [30] and commonly used interpolation based $\mathcal{H}^2$-methods [63, 44].

Anisotropic DD First of all, our fast approach to isotropic DD simulations should allow to perform the accurate simulation of massive ensembles of dislocations and thus better understand the plastic behavior of complex crystalline materials. Although our contribution to the isotropic force evaluation was in our opinion not novel enough, we think that writing an article on an interpolation based FMM formulation for both isotropic and anisotropic elastic fields would represent a significant and novel contribution to the domain of Multiscale Materials Modeling. In fact, anisotropic models should allow to better simulate and understand the behavior of materials that exhibit large anisotropic coefficients like *Fe*. During this thesis, an efficient representation of the anisotropic force field based on the *Stroh* formalism and expansions in spherical harmonics were also addressed, that should allow for an efficient fast multipole implementation in a similar fashion as the recent methodology presented in [9]. Special care should be taken in order to derive an equivalent non-singular formalism for anisotropy as well as analytic expressions for the integrals of the expansion in spherical harmonics. The *ufmm* should efficiently address the high dimensional interactions involved in anisotropic DD simulations and make the comparison of various formulations easier.

Applications to Data Assimilation We hope that our $\mathcal{H}^2$-powered randomized Singular Value Decomposition (SVD) algorithm can help solving large scales problems of geostatistics and that it can be beneficial to the computational geosciences community. Therefore, we are considering applying the method to a more active topic of geostatistics, namely data assimilation via Kalman Filtering. Recent advances in Kalman Filtering have brought *ensemble* variants and thus low-rank square root algorithms in front of the scene, namely *Ensemble*-[52], *Unscented*-Kalman Filters [70] and reduced order variants [96, 33]. As these methods rely either on efficient computation of a covariance matrix square root or generation of correlated noises, they provide an ideal framework for the application of our algorithm. However, designing a relevant application to emphasize the benefits of our approach is a rather challenging issue. During this thesis we met or exchanged with people from several teams of the data assimilation community, namely hydrologists, climatologists and oceanographists from either CERFACS or Stanford Civil Engineering Department, but only recently came up with an application that could benefit to a broader community.

Linear scaling MDS We are planning on developing a linear scaling factorization technique in a near future in order to address very large biological datasets in a more efficient way. This approach should benefit from the clustering phenomenon occurring on the point clouds and will benefit from recent advances in k-means algorithms and graph partitioning. Moreover, we would like to be able to better characterize and compare the shape of the clouds between various time of the year. In particular, we wish to analyze the point cloud

associated to several samples at the same time. Another crucial perspective consists in designing new metrics as well as efficient application-oriented visualization tools in the spirit of Chris Johnson's work, *e.g.*, [102].

Concluding remarks Randomized Numerical Linear Algebra (NLA) is a relatively young research area compared to the FMM or $\mathcal{H}$-matrices and it is most definitely much wider. However, recently the combination of both randomized and hierarchical algorithms seems to be a topic of broad interest [118, 90]. A crucial issue here is to develop new hierarchical methods with lower dependence in the ambient dimension, a topic that we wish to address very soon. We hope that our open-source library for the randomized low rank approximation of matrices can benefit from the variety of topics addressed in this thesis and from our upcoming publications. Finally, we hope that the hierarchical and randomized NLA communities can keep growing and that we can continue exchanging on these beautiful topics, their complementarity as well as their numerous applications.

Appendices

A

Tensorial interpolation based FMM

Contents

A.1 Nature of tensorial interactions

Although the Fast Multipole Method (FMM) was originally designed for particle interactions with $k(r) = 1/r$ [59], *i.e.*, a scalar valued kernel, many n-body problems formulated in terms of Boundary Integral Equations [16, 94] have arisen in the past decade that involve tensorial valued kernels [113, 106, 31] and that can be very well treated by the FMM. The tensorial nature of the kernel appears as soon as we consider that densities are multivariate quantities evaluated on grid points $\mathbf{x}$, *i.e.*,

$$\mathbf{w}(\mathbf{x}) = \{w_J(\mathbf{x})\}_{J=1\ldots d_w} \tag{A.1}$$

where d_w denotes the dimension of $\mathbf{w}(\mathbf{x})$. For instance, in most BEM formulations used for isotropic elasticity the Green's function is given as

$$G_{ij}(r) = \frac{1}{8\pi\mu}(\delta_{ij} r_{,pp} - \frac{1}{2(1-\nu)} r_{,ij}) \tag{A.2}$$

for the infinite elastic space [31], where μ and ν denote the Lamé coefficients, whereas for acoustics it is given as a scalar valued function,

$$G(r) = \frac{1}{4\pi r} \tag{A.3}$$

namely Laplace kernel, or Helmholtz kernel in the frequency domain [86, 110]. In elasticity problems, G is tensorial because it represents the relation between a multivariate density $\mathbf{w}$ (namely a point force) and the displacement $\mathbf{u}$. This tensorial nature can become problematic for problems involving large number of parameters, *e.g.*, viscoelasticity and poroelasticity [106], highly oscillatory kernels, *e.g.*, elastodynamics in time or frequency domain [113], or multi-layered spaces [32]. In particular, the author of the present contributed to the design of an efficient interpolation based directional Fast Multipole BEM formulation for elastodynamics in Laplace domain and results are discussed in Thomas Traub's PhD thesis [113]. In the present thesis we used tensorial interactions in the context of Dislocation Dynamics (DD) simulations (see Chapter 2), a special case of elastostatic interactions between line segments.

A.2 A generic formalism

Let us first define a generic formalism for tensorial interactions using a direct formulation. Then, we describe the associated fast multipole summation scheme. For the sake of clarity and genericity we denote d the ambient dimension, although most of the time $d = 3$.

A.2.1 Direct formulation

First of all, multivariate potentials (resp. densities) are written in bold characters as $\mathbf{p}$ (resp. $\mathbf{w}$) and defined by d_p (resp. d_w) components identified by a multi-index $\mathbf{I}$ (resp. $\mathbf{J}$) of order o_p (resp. o_w), such that $|\mathbf{I}| = \max_{i \leq o_p}(I_i) \leq d$ and $d_w = d^{o_w}$. Then, the kernel of interaction is denoted $\mathbf{k}$ and defined for any grid points $\mathbf{x}$ and $\mathbf{y}$ as

$$\mathbf{k}(\mathbf{x}, \mathbf{y}) = \{k_{\mathbf{IJ}}(\mathbf{x}, \mathbf{y})\}_{|\mathbf{I}| \leq d, |\mathbf{J}| \leq d} \tag{A.4}$$

Finally, the $\mathbf{I}$-th component of the potential generated at the target point $\mathbf{x}$ by all components of the densities located at the source points $\mathbf{y}$ reads as

$$p_{\mathbf{I}}(\mathbf{x}) = \sum_{\Pi(\mathbf{J}) \leq d_w} k_{\mathbf{IJ}}(\mathbf{x}, \mathbf{y}) w_{\mathbf{J}}(\mathbf{y}),\ \forall \mathbf{I} / \Pi(\mathbf{I}) \leq d_p \tag{A.5}$$

where $\Pi(\mathbf{I}) = \Pi_{i \leq o_p} I_i$.

A.2.2 Fast Multipole formulation

Computing such interactions on a large number of points can be done in $\mathcal{O}(n)$ operations using FMM. The steps of the associated Fast Multipole summation scheme are defined by the following equations:

- P2M: Aggregate contributions of the d_w components of density $\mathbf{w}$ located at source points $\mathbf{y}$ into d_w multipole expansions defined on source interpolation nodes $\bar{\mathbf{y}}_{\boldsymbol{\beta}}$

$$\left(\mathcal{M}_{\boldsymbol{\beta}}\right)_{\mathbf{J}} = \sum_{\mathbf{y}} S_{\boldsymbol{\beta}}(\mathbf{y}) w_{\mathbf{J}}(\mathbf{y}),\ \forall |\mathbf{J}| \leq d_w \tag{A.6}$$

- M2L: Transfer d_w multipole expansions $\left(\mathcal{M}_{\boldsymbol{\beta}}\right)_{\mathbf{J}}$ into d_ϕ local expansions $\left(\mathcal{L}_{\boldsymbol{\alpha}}\right)_{\mathbf{I}}$ defined on target interpolation nodes $\bar{\mathbf{x}}_{\boldsymbol{\alpha}}$

$$\left(\mathcal{L}_{\boldsymbol{\alpha}}\right)_{\mathbf{I}} = \sum_{\Pi(\mathbf{J}) \leq d_w} \sum_{|\boldsymbol{\beta}| \leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{\mathbf{IJ}} \left(\mathcal{M}_{\boldsymbol{\beta}}\right)_{\mathbf{J}},\ \forall |\mathbf{I}| \leq d_\phi \tag{A.7}$$

- L2P: Aggregate contributions of d_ϕ local expansions from target interpolation nodes $\bar{\mathbf{x}}_{\boldsymbol{\alpha}}$ into the d_ϕ components of the potential $\boldsymbol{\phi}$ located at all target points $\mathbf{x}$

$$\bar{\phi}_{\mathbf{I}}(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}| \leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) \left(\mathcal{L}_{\boldsymbol{\alpha}}\right)_{\mathbf{I}},\ \forall |\mathbf{I}| \leq d_\phi \tag{A.8}$$

A.3 Matrix-to-Vector interactions

A.3.1 Direct formulation

Let us for instance consider the case where potentials and densities are vectors of dimension $d_\phi = d_w = d$ and $\mathbf{k}$ is a d-by-d matrix, *i.e.*, $d_k = d \times d$. Then, $\mathbf{I}$ and $\mathbf{J}$ are simple indices, that can be denoted i and j, and the summation over the indices reads as

$$\phi_i(\mathbf{x}) = \sum_{j=1}^{d} k_{ij}(\mathbf{x}, \mathbf{y}) w_j(\mathbf{y}),\ \forall i = 1, \dots, d \tag{A.9}$$

A.3.2 Fast Multipole formulation

- P2M: Aggregate contributions of the d components of density $\mathbf{w}$ located at source points $\mathbf{y}$ into d multipole expansions defined on source interpolation nodes $\bar{\mathbf{y}}_{\boldsymbol{\beta}}$

$$\left(\mathcal{M}_{\boldsymbol{\beta}}\right)_j = \sum_{\mathbf{y}} S_{\boldsymbol{\beta}}(\mathbf{y}) w_j(\mathbf{y}),\ \forall j \leq d_w \tag{A.10}$$

- M2L: Transfer d multipole expansions $\left(\mathcal{M}_{\boldsymbol{\beta}}\right)_j$ into d local expansions $\left(\mathcal{L}_{\boldsymbol{\alpha}}\right)_i$ defined on target interpolation nodes $\bar{\mathbf{x}}_{\boldsymbol{\alpha}}$

$$\left(\mathcal{L}_{\boldsymbol{\alpha}}\right)_i = \sum_{j \leq d} \sum_{|\boldsymbol{\beta}| \leq p} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{ij} \left(\mathcal{M}_{\boldsymbol{\beta}}\right)_j,\ \forall i \leq d \tag{A.11}$$

- L2P: Aggregate contributions of d local expansions from target interpolation nodes $\bar{\mathbf{x}}_{\boldsymbol{\alpha}}$ into the d components of the potential $\boldsymbol{\phi}$ located at all target points $\mathbf{x}$

$$\bar{\phi}_i(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}| \leq p} S_{\boldsymbol{\alpha}}(\mathbf{x}) \left(\mathcal{L}_{\boldsymbol{\alpha}}\right)_i,\ \forall i \leq d \tag{A.12}$$

A.4 Optimizations and numerical complexities

A.4.1 Multidimensional interpolators

The L2P, resp. P2M, steps must be applied for d_p, resp. d_w, potentials, resp. density. In a naïve implementation, this would result in multiplying the cost of these steps by their associated dimension. Most importantly, the steps described above require that d_w multipole expansions and d_p local expansions are stored during the algorithm. We recall that storing an expansion costs $\mathcal{O}(p^3)$ bytes, more precisely $8 \times (p+1)^3$ for a real valued expansion in double precision. Each cell of the octree contains at least* 2 expansions, therefore it must be multiplied by the number of cells, *e.g.*, $\#_{cells}(L) = (2^3)^L = 8^L$ for an arbitrary level L.

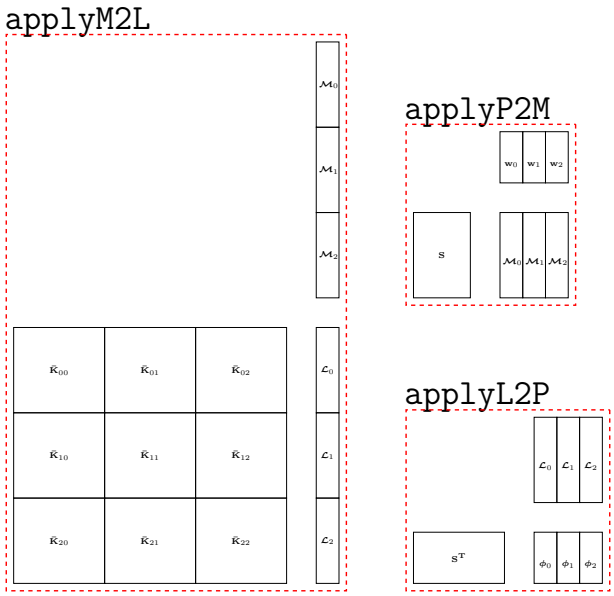

Figure A.1: Schematic view of the tensorial M2L step in case #1.

Hence, the cost of storing the expansions grows dramatically with the depth of the octree.

A.4.2 Tensorial M2L

The tensorial nature of the interactions also affects the critical M2L step. In fact, all components $k_{\mathbf{IJ}}$ need to be evaluated, stored and applied to the local expansions. Therefore, the cost of the precomputation is roughly multiplied by the number of components d_k. If some components depend on each other by a constant factor (*e.g.*, symmetric tensors), then less components need to be stored. We denote d_k^{min} the number of independent components of the tensor of interactions $\mathbf{k}$ that actually need to be stored. Finally, in (A.7) the summation over $\mathbf{J}$ is done outside the loop over the interpolation grid in order to preserve the optimized application of the M2L operators.

*For each expansion (local and multipole) ScalFMM uses an extra array to store a compressed expansion for `Chebyshev FMM` or a transformed (complex valued) expansion for `Uniform FMM`.

B

Distribution of particles on various geometries

Contents

Here we give advanced statistics on various distributions of points *w.r.t.* to the geometry and the depth of the octree.

B.1 Geometries with various heterogeneity

Figure B.1 represents actual distribution of points for all geometries considered in this thesis. It also shows associated 2D distributions inside a quadtree with $\bar{L} = 3, 4, 5$. This gives an estimation of the filling of the tree in each case, as well as the number of nearfield and farfield interactions.

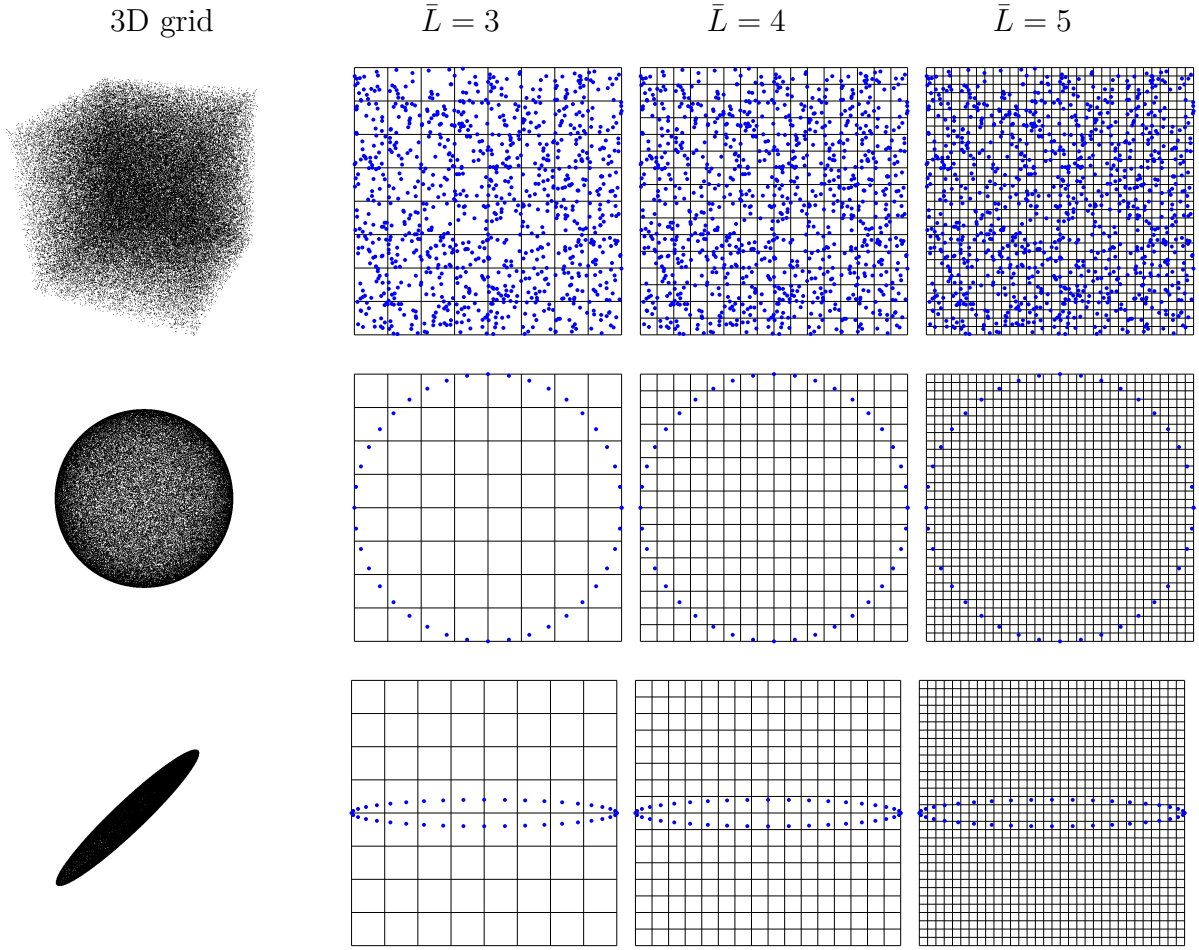

Figure B.1: Uniform distribution of particles in the unit cube (top), the unit sphere (center) and the prolate sphere with ratio 10 (bottom).

B.2 Statistics on the octree

The evolution of number of non-empty cells *w.r.t.* the number of level for $n = 10^6$ particles is represented on Figure B.2. In particular, we observe that the unit cube is fully populated up to $\bar{L} = 6$ then the filling rate decreases with the tree depth. Meanwhile filling rates of the unit and prolate spheres have similar decreasing speed, but as expected the prolate sphere has more empty leaves than the unit sphere. Figure B.3 represents the total and average

number of nearfield and farfield interactions *w.r.t.* to the tree depth $\bar{L}$. When the particles are distributed in the unit cube, the number of points is not high enough to have a sufficient density and maintain the filling rate.

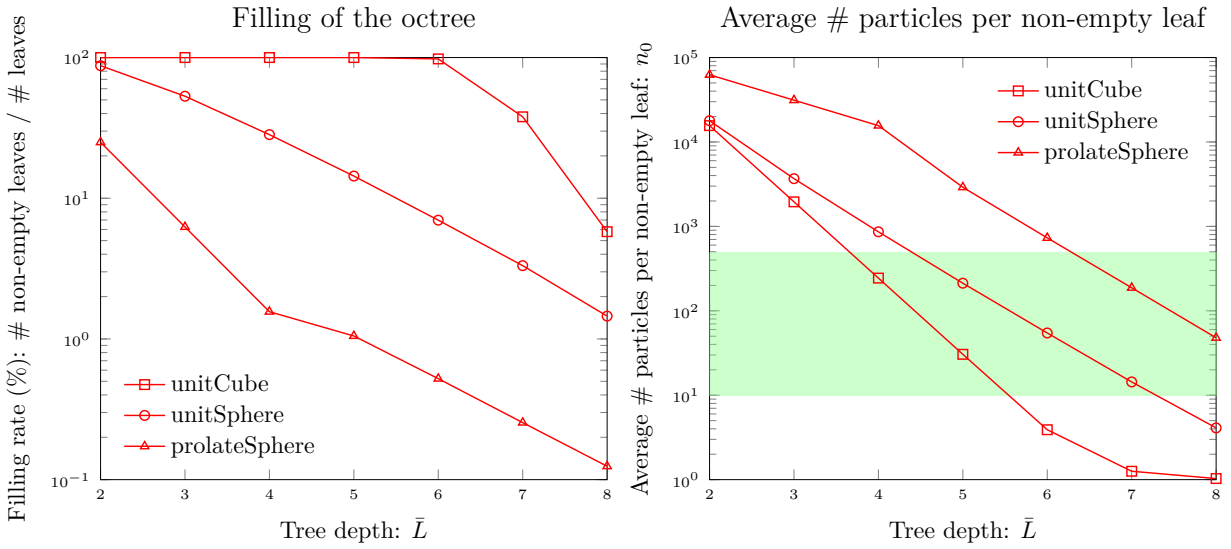

Figure B.2: Left: Filling rate at the leaf level *w.r.t.* to tree depth $\bar{L}$ for various geometries. When the points are distributed on a surface, *e.g.*, the unit sphere or the prolate sphere, the tree is very sparse (low filling rate). Indeed, the number of non-empty leaves differs significantly from the maximum number of leaves, *i.e.*, $8^{\bar{L}}$. Right: Average number of particles per leaf *w.r.t.* to $\bar{L}$ for various geometries with $n = 10^6$ particles. In practice, leaves contain between 10 and 500 particles in average (green area) depending on the application, *i.e.*, on the nature of the kernel, the accuracy and the expansion technique.

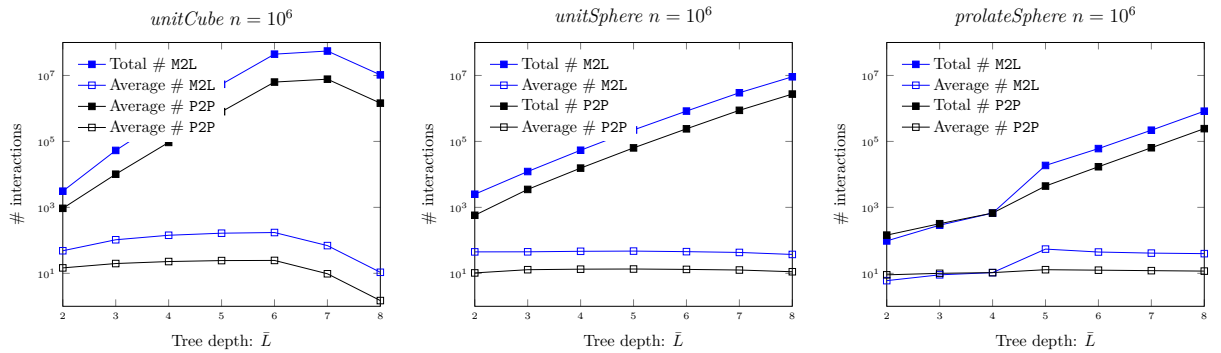

Figure B.3: Total and average number of leaf-level M2L and P2P operators for various distributions of particles ($n = 10^6$) *w.r.t.* $\bar{L}$. Observations: The average number of operators stays relatively constant for all geometries, in particular we quickly reach the maximum number of operators for the unit cube, namely 27 P2P and 189 M2L, while the unit sphere only has 15 P2P and 50 M2L in average. The prolate sphere exhibit 2 regimes, namely $\bar{L} \leq 4$ or $\bar{L} \geq 5$, whether farfield interactions are considered between particles lying on the small section or not.

C

Performance of the multi-*rhs* Uniform FMM

Contents

C.1 Numerical benchmarks

Here we present comparative results on the convergence of the multi-*rhs* *ufmm* and *smooth-ufmm* *w.r.t.* the interpolation order p for various geometries: the unit sphere, a cube and a prolate sphere. In particular we analyze the effect of the length scale on the error of the Fast Multipole Method (FMM) for the Gaussian correlation kernel for $\ell = 0.5$ or $\ell = 1.0$. All computations were performed on a cluster computer, namely *plafrim/mirabelle*: Hexa-core Westmere Intel Xeon X5670 @ 2.93GHz with 96GB ram and 12MB L3 Cache. Results are presented on Figure C.1 for the cube, Figure C.1 for the unit sphere and Figure C.1 for the prolate sphere. They show the computational time of a fast multipole matrix multiplication per *rhs* for a total of 10 *rhs*.

C.2 Performance of the *ufmm* and the *smooth-ufmm*.

The *ufmm* and the *smooth-ufmm* exhibit similar performances if the tree depth is set to the same value. However, as the depth decreases, the *smooth-ufmm* becomes significantly cheaper. Hence, the results presented here corroborate the one discussed in Section 5.4. Precomputing the P2P operators takes about 3 to 8 seconds depending on the geometry and the tree depth, which is approximately the time required to apply the *ufmm* to 1 *rhs* with an intermediate accuracy.

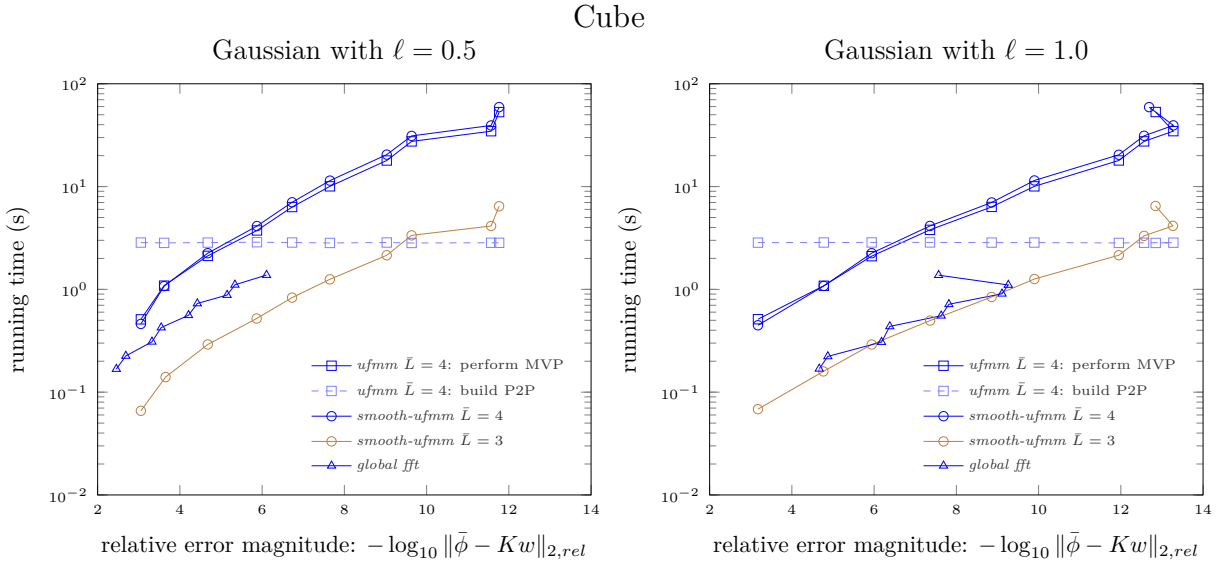

Figure C.1: Running time of a multi-rhs FMM *w.r.t.* the relative error magnitude using various algorithms: *ufmm* and *smooth-ufmm* with $p = 2 \dots 11$, and *global fft* with $p = 6 \dots 14$. We used 72.000 particles randomly distributed in a cube. Observations: *ufmm* and *smooth-ufmm* have approximately the same cost when they share the same tree depth. The *smooth-ufmm* is significantly faster if we choose an optimal tree depth, *e.g.*, $\bar{L} = 3$ represented in solid brown lines. If the Gaussian decreases sufficiently slow, *i.e.*, the rank is sufficiently low, then the cost of the *global fft* is similar to the optimal *smooth-ufmm*. However, the *global fft* exhibits an instability for the highest interpolation order.

C.3 Performance of the *global fft*

The *global fft* will always have the same cost at a given interpolation order, since this method is oblivious of the shape of the distribution. On the other hand, the cost of the hierarchical methods may vary significantly from one distribution to another. Let us for instance consider a Gaussian correlation with $\ell = 0.5$. If the particles are distributed in the cube (*i.e.*, an homogeneous distribution) the cost of the *global fft* lies between the *ufmm* and the *smooth-ufmm* with optimal $\bar{L}$, see Figure C.1. If the particles are distributed on a sphere (*i.e.*, an heterogeneous distribution) then the hierarchical methods become faster than the *global fft*, see Figure C.2.

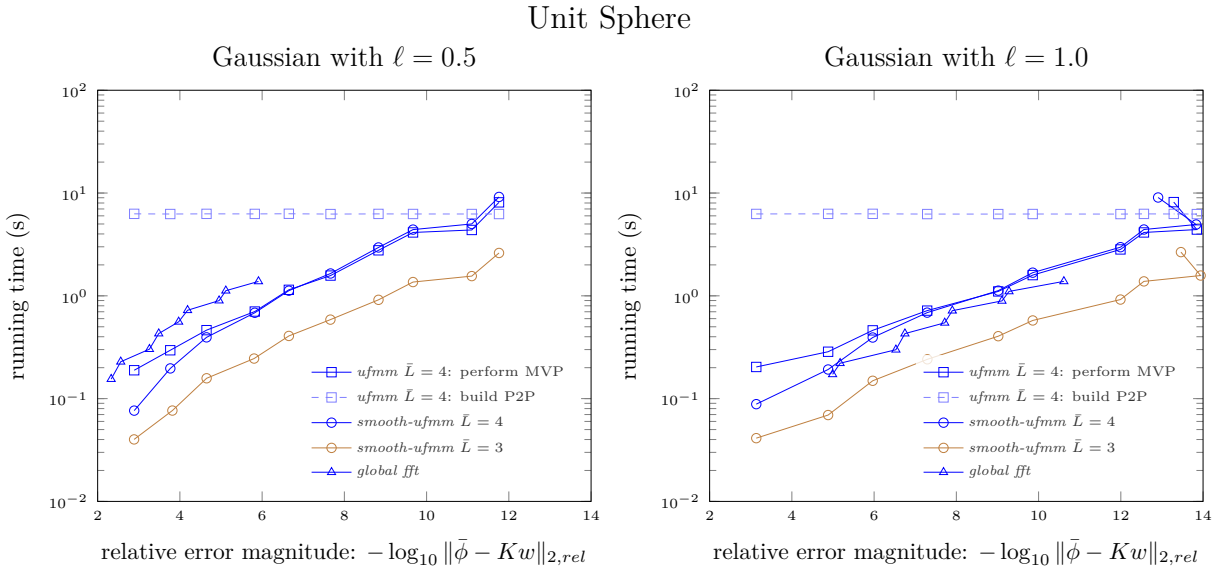

Figure C.2: Running time of a multi-*rhs* FMM *w.r.t.* the relative error magnitude using various algorithms: *ufmm* and *smooth-ufmm* with $p = 2 \dots 11$, and *global fft* with $p = 6 \dots 14$. We used 72.000 particles randomly distributed on the unit sphere. Observations: *ufmm* and *smooth-ufmm* have approximately the same cost when they share the same tree depth. The *smooth-ufmm* is significantly faster if we choose an optimal tree depth, *e.g.*, $\bar{L} = 3$ represented in brown. If the Gaussian decreases sufficiently slow (*i.e.*, the rank is sufficiently low), then the cost of the *global fft* is slightly lower than the *ufmm* but the optimal *smooth-ufmm* still performs better.

C.4 Other distributions

Fig. C.3 confirms the previous observations on the prolate sphere, *i.e.*, a highly heterogeneous distribution. The associated octree has larger depths ($\bar{L} = 6$) than that of the unit sphere ($\bar{L} = 4$), however in the lowest level a large number of cells are empty. Consequently, the running times are only slightly larger than for the unit sphere.

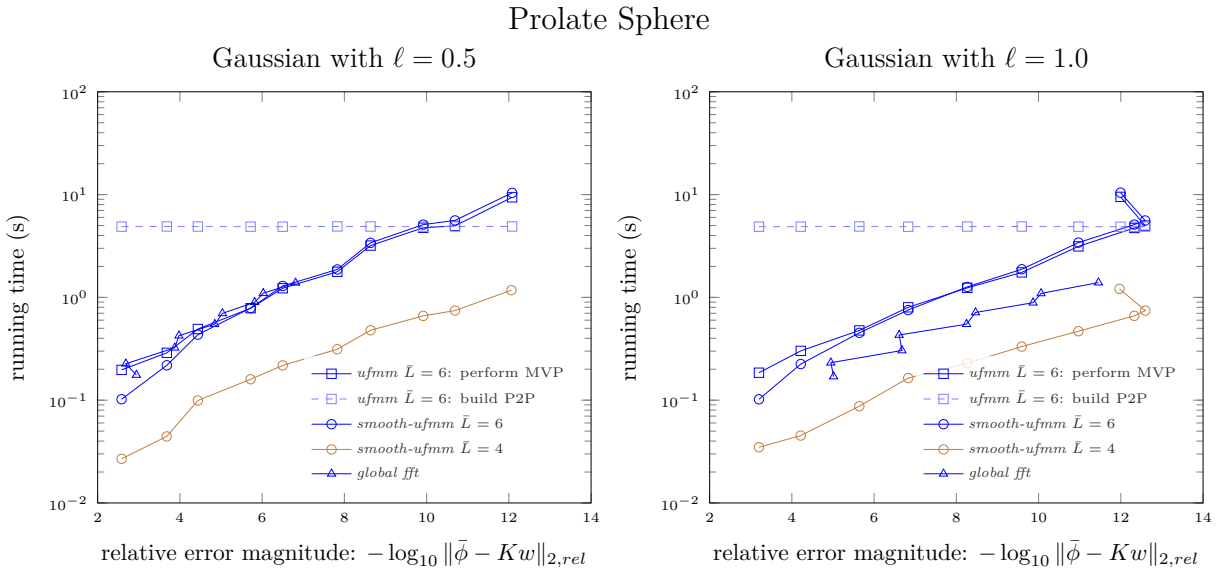

Figure C.3: Running time of a multi-*rhs* FMM *w.r.t.* the relative error magnitude using various algorithms: *ufmm* and *smooth-ufmm* with $p = 2 \dots 11$, and *global fft* with $p = 6 \dots 14$. We used 72.000 particles randomly distributed on a prolate sphere (ratio 1:1:10). Observations: *ufmm* and *smooth-ufmm* have approximately the same cost when they share the same tree depth. The *smooth-ufmm* is significantly faster if we choose an optimal tree depth, *e.g.*, $\bar{L} = 4$ represented in brown. If the Gaussian decreases sufficiently slow (*i.e.*, the rank is sufficiently low), then the *global fft* performs relatively well compared to the hierarchical variants.

D

Optimized Fast Multipole DD simulations

Contents

D.1 M2L-Optimized stress computation

In this section we start by introducing an existing approach used to optimize the number of M2L operators and operations per interaction in the case of isotropic elastic forces computation. Then, we present the principle of our new method based on refactoring the M2L operations. Finally, we apply our method on the evaluation of the isotropic elastic energy.

D.1.1 Existing approach

Arsenlis approach In [8] the isotropic elastic stress field (2.4) is written as a combination of permutation symbols and a 5^{th}-order tensor G_{ijklm} defined as

$$G_{ijklm}(\mathbf{x}) = \oint_C R_{,klm}(\mathbf{x}-\mathbf{y}) b_i dy_j \tag{D.1}$$

Therefore, the contributions to the elastic stress read as

$$\sigma^A_{ij}(\mathbf{x}) = \varepsilon_{jmk} G_{kimpp}(\mathbf{x}) \tag{D.2}$$
$$\sigma^B_{ij}(\mathbf{x}) = \varepsilon_{nmk} G_{knmij}(\mathbf{x}) \tag{D.3}$$

Let us determine the number of independent components of $\mathbf{G}$ that are actually involved in the computation of $\boldsymbol{\sigma}$. First of all, given the symmetries of $R_{,klm}$ only the following 10 components (instead of $3^3 = 27$) of $G_{ij\ldots}$ need to be computed for all $(i,j) \in \{1,\ldots,3\}^2$

$$\begin{array}{llll} G_{ij111} & G_{ij122} & G_{ij133} & G_{ij123} \\ G_{ij211} & G_{ij222} & G_{ij233} & \\ G_{ij311} & G_{ij322} & G_{ij333} & \end{array}$$

Besides only the following components of $G_{\ldots lm}$ for all $(l,m) \in \{1,\ldots,3\}^2$ are actually required when the permutation symbols are applied:

$$\begin{array}{lll} \sigma^A_{11} = -G_{213pp} + G_{312pp}, & \sigma^A_{12} = -G_{311pp} + G_{113pp}, & \sigma^A_{13} = +G_{211pp} - G_{112pp}, \\ \sigma^A_{21} = -G_{223pp} + G_{322pp}, & \sigma^A_{22} = -G_{321pp} + G_{123pp}, & \sigma^A_{23} = +G_{221pp} - G_{122pp}, \\ \sigma^A_{31} = -G_{233pp} + G_{332pp}, & \sigma^A_{32} = -G_{331pp} + G_{133pp}, & \sigma^A_{33} = +G_{231pp} - G_{132pp}, \end{array}$$

$$\sigma^B_{lm} = +G_{231lm} - G_{321lm} - G_{132lm} + G_{312lm} + G_{123lm} - G_{213lm},$$

The other components are multiplied by 0. If we combine these 2 optimizations we can reduce the set of computed components from $3^5 = 243$ to $n = n_A + n_B - n_{AinterB} = 18 \times 3 + 6 \times 6 - (6 \times 3 + 6) = 66$. In fact,

- The computation of σ^A_{ij} for all $(i,j) \in \{1,\ldots,3\}^2$ involves the following $(3 \times 6) \times 3 = 54$

components of $\mathbf{G}$

$$G_{213pp}\ G_{312pp}\ G_{311pp}\ G_{113pp}\ G_{211pp}\ G_{112pp}$$
$$G_{223pp}\ G_{322pp}\ G_{321pp}\ G_{123pp}\ G_{221pp}\ G_{122pp}$$
$$G_{233pp}\ G_{332pp}\ G_{331pp}\ G_{133pp}\ G_{231pp}\ G_{132pp}$$

- While the computation of σ_{ij}^B for all $(i,j) \in \{1,\dots,3\}^2$ with $i < j$ involves the following $6 \times 6 = 36$ components of $\mathbf{G}$

$$G_{21312}\ G_{21313}\ G_{21323}\ G_{31212}\ G_{31213}\ G_{31223}$$
$$G_{21123}\ G_{21133}\ G_{21233}\ G_{31122}\ G_{31123}\ G_{31322}$$
$$G_{32112}\ G_{32113}\ G_{32123}\ G_{12312}\ G_{12313}\ G_{12323}$$
$$G_{32211}\ G_{32311}\ G_{32123}\ G_{12123}\ G_{12133}\ G_{12233}$$
$$G_{23112}\ G_{23113}\ G_{23123}\ G_{13212}\ G_{13213}\ G_{13223}$$
$$G_{23211}\ G_{23311}\ G_{23123}\ G_{13122}\ G_{13123}\ G_{13322}$$

- The 24 components shared by σ_{ij}^A and σ_{ij}^B are colored in grey.

We could not recover the factor of 6 mentioned in [8], *i.e.*, the 40 components out of $3^5 = 243$ using this approach. Even considering $\sigma_{ij}^A + \sigma_{ji}^A$ instead of σ_{ij}^A does not seem to change the result. Moreover, Arsenlis et al. [8] affirms that none of the elements of G_{iiklm}, G_{ijkli} and G_{ijklj} are required in the computation, but we found that it is only true for the computation of σ^B. On the other hand, G_{ijill} is indeed not involved in the computation of either σ^A or σ^B. The aforementioned paper is known to contain a few typos, therefore we developed our own approach. Besides we do not think this is the most relevant way of optimizing M2L operations, especially when considering polynomial interpolation.

D.1.2 Refactoring M2L operations

Principle In order to define an efficient summation scheme, we want to identify the optimal definition of the multipole expansions that leads to the minimum number of M2L operations. Our method consists in rewriting the full expression of each component of $\boldsymbol{\sigma}$ and refactoring it *w.r.t.* to the components of $\mathbf{R}$. Let us first denote $G_{ijklm} = G_{IJ}$ with

$I_{ij} \in [0,8]$ and $J_{klm} \in [0,9]$. We also denote $\mathbf{m}$ the multipole such that $G_{IJ} = R_J m_I$.

$$
\begin{aligned}
\sigma_0^A &= -G_{36} - G_{37} - G_{38} + G_{63} + G_{64} + G_{65} \\
\sigma_1^A &= -G_{60} - G_{61} - G_{62} + G_{06} + G_{07} + G_{08} \\
\sigma_2^A &= +G_{30} + G_{31} + G_{32} - G_{03} - G_{04} - G_{05} \\
\sigma_3^A &= -G_{46} - G_{47} - G_{48} + G_{73} + G_{74} + G_{75} \\
\sigma_4^A &= -G_{70} - G_{71} - G_{72} + G_{16} + G_{17} + G_{18} \\
\sigma_5^A &= +G_{40} - G_{41} - G_{42} - G_{13} - G_{14} - G_{15} \\
\sigma_6^A &= -G_{56} - G_{57} - G_{58} + G_{83} + G_{84} + G_{85} \\
\sigma_7^A &= -G_{80} - G_{81} - G_{82} + G_{26} + G_{27} + G_{28} \\
\sigma_8^A &= +G_{50} + G_{51} + G_{52} - G_{23} - G_{24} - G_{25}
\end{aligned}
$$

Computing σ_{ij}^A and summing with the transposed matrix may be a good idea in order to obtain $\sigma_{ij}^{A'} = \sigma_{ij}^A + \sigma_{ji}^A$ but it is actually a little slower than computing $\sigma_{ij}^{A'}$ directly as

$$
\begin{aligned}
\sigma_0^{A'} &= -2m_3R_6 - 2m_3R_7 - 2m_3R_8 + 2m_6R_3 + 2m_6R_4 + 2m_6R_5 \\
\sigma_1^{A'} &= m_7R_3 + m_7R_4 + m_7R_5 - m_6R_0 - m_6R_1 - m_6R_2 \\
&\quad + (m_0 - m_4)R_6 + (m_0 - m_4)R_7 + (m_0 - m_4)R_8 \\
\sigma_2^{A'} &= +m_3R_0 + m_3R_1 + m_3R_2 - m_5R_6 - m_5R_7 - m_5R_8 \\
&\quad + (m_8 - m_0)R_3 + (m_8 - m_0)R_4 + (m_8 - m_0)R_5 \\
\sigma_3^{A'} &= -2m_7R_0 - 2m_7R_1 - 2m_7R_2 + 2m_1R_6 + 2m_1R_7 + 2m_1R_8 \\
\sigma_4^{A'} &= +m_2R_6 + m_2R_7 + m_2R_8 - m_1R_3 - m_1R_4 - m_1R_5 \\
&\quad + (m_4 - m_8)R_0 + (m_4 - m_8)R_1 + (m_4 - m_8)R_3 \\
\sigma_5^{A'} &= +2m_5R_0 + 2m_5R_1 + 2m_5R_2 - 2m_2R_3 - 2m_2R_4 - 2m_2R_5
\end{aligned}
$$

Similarly, for σ^B we have

$$
\begin{aligned}
\sigma_0^B &= +(m_5 - m_7)R_0 + (m_6 - m_2)R_3 + (m_1 - m_3)R_6 \\
\sigma_1^B &= +(m_5 - m_7)R_3 + (m_6 - m_2)R_1 + (m_1 - m_3)R_9 \\
\sigma_2^B &= +(m_5 - m_7)R_6 + (m_6 - m_2)R_9 + (m_1 - m_3)R_2 \\
\sigma_3^B &= +(m_5 - m_7)R_1 + (m_6 - m_2)R_4 + (m_1 - m_3)R_7 \\
\sigma_4^B &= +(m_5 - m_7)R_9 + (m_6 - m_2)R_7 + (m_1 - m_3)R_5 \\
\sigma_5^B &= +(m_5 - m_7)R_2 + (m_6 - m_2)R_5 + (m_1 - m_3)R_8
\end{aligned}
$$

Finally, the expression of the elastic stress field $\sigma = \sigma^{A'} + c_b\sigma^{B'}$ (with $c_b = 2/(1-\nu)$) reads as

$$
\begin{aligned}
\sigma_0 =& -2m_3R_6 + 2m_6R_3 \\
&- c_b(m_5 - m_7)R_1 - (c_b(m_6 - m_2) - 2m_6)R_4 - (2m_3 + c_b(m_1 - m_3))R_7 \\
&- c_b(m_5 - m_7)R_2 - (c_b(m_6 - m_2) - 2m_6)R_5 - (2m_3 + c_b(m_1 - m_3))R_8 \\
\sigma_1 =& + m_7R_4 + m_7R_5 - m_6R_0 - m_6R_2 + (m_0 - m_4)R_6 + (m_0 - m_4)R_7 + (m_0 - m_4)R_8 \\
&+ (c_b(m_5 - m_7) + m_7)R_3 + (c_b(m_6 - m_2) - m_6)R_1 + c_b(m_1 - m_3)R_9 \\
\sigma_2 =& + m_3R_0 + m_3R_1 - m_5R_7 - m_5R_8 + (m_8 - m_0)R_3 + (m_8 - m_0)R_4 + (m_8 - m_0)R_5 \\
&+ (c_b(m_5 - m_7) - m_5)R_6 + c_b(m_6 - m_2)R_9 + (c_b(m_1 - m_3) + m_3)R_2 \\
\sigma_3 =& -2m_7R_1 + 2m_1R_7 + 2m_1R_8 \\
&- (c_b(m_5 - m_7) + 2m_7)R_0 - c_b(m_6 - m_2)R_3 - (c_b(m_1 - m_3) - 2m_1)R_6 \\
&- (c_b(m_5 - m_7) + 2m_7)R_2 - c_b(m_6 - m_2)R_5 - (c_b(m_1 - m_3) - 2m_1)R_8 \\
\sigma_4 =& + c_b(m_5 - m_7)R_9 + c_b(m_6 - m_2)R_7 + c_b(m_1 - m_3)R_5 \\
\sigma_5 =& - c_b(m_5 - m_7)R_0 - c_b(m_6 - m_2)R_3 - c_b(m_1 - m_3)R_6 \\
&- c_b(m_5 - m_7)R_1 - c_b(m_6 - m_2)R_4 - c_b(m_1 - m_3)R_7
\end{aligned}
$$

Using the previous forms of σ^A and σ^B the applications of the M2L operators are optimal in the sense that the minimum number of $R_{\cdot\cdot\cdot}$-to-$m_{\cdot\cdot}$ products is performed, namely 46, and the minimum number of multipole expansions is stored, namely the 9 components of m. Until now this is the cheapest method in term of memory requirements, however it requires assembling the term in front of each component of $\mathbf{R}$ just before the application of the M2L operator, *i.e.*, approximately 189 times per cell for a full octree.

Final formulation A good compromise can be found in order to accelerate the M2L step, while minimizing the memory requirements. First of all, if we transfer the contributions to $\sigma^{A'}$ and σ^B independently at the M2L step, we only need to store a total of 12 multipole components, namely (m_i) for $i \in \{1,2,3,5,6,7\}$, $(m_0 - m_4)$, $(m_8 - m_0)$ and $(m_8 - m_4)$ for $\sigma^{A'}$, and $(m_5 - m_7)$, $(m_6 - m_2)$ and $(m_1 - m_3)$ for σ^B. Then, we can still assemble the local contributions to σ into 6 local expansions after the M2L. Finally, if we allow the storage of 3 extra components of $\mathbf{R}$, namely R_{ipp} for $i \in \{1,\dots,3\}$, we reduce the number of M2L operations to $3 \times 5 = 15$ for $\sigma^{A'}$, *i.e.*, a total of 51 M2L operations for σ.

D.2 M2L-Optimized energy computation

Here, we present our implementation of the farfield contributions to the isotropic elastic energy, whose expression is recalled in Section 2.2.2. In order to remain consistent with formula already implemented in *OptiDis*, we used the formula from [28]. However, the efficient fast multipole implementation is more challenging than that of the stress field, therefore we propose to use a different formula in order to reduce the cost of computing the energy using Fast Multipole Method (FMM).

D.2.1 Fast Multipole computation of the isotropic energy

Interpolation If we replace the expression (6.3) of the interpolant $\tilde{\mathbf{k}}$ in (2.8) we get an approximation of the elastic energy field $\boldsymbol{\sigma}$, that reads as

$$E((\mathcal{C})) \approx \tilde{E}((\mathcal{C})) = -\frac{\mu}{8\pi}(\tilde{E}_A + \frac{2}{1-\nu}(\tilde{E}_B - \tilde{E}_C + \nu\tilde{E}_D)) \tag{D.4}$$

where contributions $\tilde{E}_A$, $\tilde{E}_B$, $\tilde{E}_C$ and $\tilde{E}_D$ are defined as

$$\begin{aligned}
\tilde{E}_A &= \sum_{|\boldsymbol{\alpha}|\leq p+1} \oint_{(\mathcal{C})} S_{\boldsymbol{\alpha}}(\mathbf{x}) b_i dx_i \sum_{|\boldsymbol{\beta}|\leq p+1} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{\ell\ell} \oint_{(\mathcal{C})} S_{\boldsymbol{\beta}}(\mathbf{x}') b'_j dx'_j \\
\tilde{E}_B &= \sum_{|\boldsymbol{\alpha}|\leq p+1} \oint_{(\mathcal{C})} S_{\boldsymbol{\alpha}}(\mathbf{x}) b_j dx_\ell \sum_{|\boldsymbol{\beta}|\leq p+1} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{ij} \oint_{(\mathcal{C})} S_{\boldsymbol{\beta}}(\mathbf{x}') b'_i dx'_\ell \\
\tilde{E}_C &= \sum_{|\boldsymbol{\alpha}|\leq p+1} \oint_{(\mathcal{C})} S_{\boldsymbol{\alpha}}(\mathbf{x}) b_i dx_j \sum_{|\boldsymbol{\beta}|\leq p+1} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{\ell\ell} \oint_{(\mathcal{C})} S_{\boldsymbol{\beta}}(\mathbf{x}') b'_i dx'_j \\
\tilde{E}_D &= \sum_{|\boldsymbol{\alpha}|\leq p+1} \oint_{(\mathcal{C})} S_{\boldsymbol{\alpha}}(\mathbf{x}) b_j dx_i \sum_{|\boldsymbol{\beta}|\leq p+1} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{\ell\ell} \oint_{(\mathcal{C})} S_{\boldsymbol{\beta}}(\mathbf{x}') b'_i dx'_j
\end{aligned} \tag{D.5}$$

These 4 contributions can also be expressed as the sum of the contributions of each segment composing the network.

Fast Multipole summation scheme Equation (D.4) can be computed using a fast multipole summation scheme, that consists in the following steps:

- First, aggregate segments contributions at interpolation nodes

$$(\mathcal{M}_{\boldsymbol{\beta}})_{ij} = \int_{[\mathbf{x}^1\mathbf{x}^2]} S_{\boldsymbol{\beta}}(\mathbf{x}') b'_i t'_j dx' \tag{P2M-E}$$

- Then, transfer multipole expansion to local expansion for E_C or E_D

$$\mathcal{L}^A_{\boldsymbol{\alpha}} = \sum_{|\boldsymbol{\beta}|\leq p+1} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{kk} (\mathcal{M}_{\boldsymbol{\beta}})_{ii} \tag{M2L-EA}$$

$$(\mathcal{L}^B_{\boldsymbol{\alpha}})_{jk} = \sum_{|\boldsymbol{\beta}|\leq p+1} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{ij} (\mathcal{M}_{\boldsymbol{\beta}})_{ik} \tag{M2L-EB}$$

$$(\mathcal{L}^{C,D}_{\boldsymbol{\alpha}})_{ij} = \sum_{|\boldsymbol{\beta}|\leq p+1} \left(\bar{K}_{\boldsymbol{\alpha}\boldsymbol{\beta}}\right)_{kk} (\mathcal{M}_{\boldsymbol{\beta}})_{ij} \tag{M2L-ECD}$$

- Finally, accumulate contributions at target points

$$E_A(\left[\mathbf{x}^1\mathbf{x}^2\right], \left[\mathbf{x}^3\mathbf{x}^4\right]) = \sum_{|\boldsymbol{\alpha}|\leq p+1} \int_{[\mathbf{x}^3\mathbf{x}^4]} S_{\boldsymbol{\alpha}}(\mathbf{x}) b_j t_j \mathcal{L}^A_{\boldsymbol{\alpha}} dx \tag{L2P-EA}$$

$$E_B(\left[\mathbf{x}^1\mathbf{x}^2\right], \left[\mathbf{x}^3\mathbf{x}^4\right]) = \sum_{|\boldsymbol{\alpha}|\leq p+1} \int_{[\mathbf{x}^3\mathbf{x}^4]} S_{\boldsymbol{\alpha}}(\mathbf{x}) b_j t_k (\mathcal{L}^B_{\boldsymbol{\alpha}})_{jk} dx \tag{L2P-EB}$$

$$E_C(\left[\mathbf{x}^1\mathbf{x}^2\right], \left[\mathbf{x}^3\mathbf{x}^4\right]) = \sum_{|\boldsymbol{\alpha}|\leq p+1} \int_{[\mathbf{x}^3\mathbf{x}^4]} S_{\boldsymbol{\alpha}}(\mathbf{x}) b_i t_j (\mathcal{L}^{C,D}_{\boldsymbol{\alpha}})_{ij} dx \tag{L2P-EC}$$

$$E_D(\left[\mathbf{x}^1\mathbf{x}^2\right], \left[\mathbf{x}^3\mathbf{x}^4\right]) = \sum_{|\boldsymbol{\alpha}|\leq p+1} \int_{[\mathbf{x}^3\mathbf{x}^4]} S_{\boldsymbol{\alpha}}(\mathbf{x}) b_j t_i (\mathcal{L}^{C,D}_{\boldsymbol{\alpha}})_{ji} dx \tag{L2P-ED}$$

All terms share the same multipole expansions, namely

$$\boldsymbol{\mathcal{M}}_{\boldsymbol{\beta}} = \oint_C S_{\boldsymbol{\beta}}(\mathbf{y}) \mathbf{b} \otimes \mathbf{t} dx', \tag{D.6}$$

while the L2P, resp. M2L, step differs for each term except E_B and E_C, resp. E_C and E_D.

Cost estimation In order to perform this summation, we need to store 7 M2L components, namely 6 for $\tilde{E}_B$ and 1 for the other terms. Moreover, we need to store 9 components of $\boldsymbol{\mathcal{M}}_{\boldsymbol{\beta}}$. On the other hand, one need to apply 47 M2L operations in order to compute all terms of the energy, namely 1 for $\tilde{E}_A$, 27 for $\tilde{E}_B$ and 9 for $\tilde{E}_{C,D}$.

D.2.2 Efficient implementation

Direct Approach In order to evaluate the isotropic elastic energy via FMM we need to compute 9 multipole expansions at the P2M step, namely

$$M_{ij} = \oint_{(C)} S(x') b_i dx'_j, \text{ for } i,j = 0\ldots 2$$

Then, apply $R_{,kk}$ to each of these expansions, *i.e.*, 9 M2L operations, and store the results in 9 local expansions in order to compute E_A, E_C and E_D at the L2P step. For E_B, we need to apply each $R_{,ij}$ to the M_{jk} expansions, *i.e.*, $(3\times 3)\times 3 = 27$ M2L operations, and store the results in $(3\times 3) = 9$ local expansions. Let us use the superscript A for all operators used

for computing $E_{A,C,D}$ and B for E_B. Hence, $i, j = 0 \dots 2$ the local expansions reads as

$$L^A_{ij} = R_{kk} M_{ij} \text{ and } L^B_{jk} = R_{ij} M_{jk}$$

If we denote $I_{ij} \in [0, 6]$ the index mapping from $[0, 2] \times [0, 2]$ to the set of symmetric components indices plus one index for the trace of $\mathbf{R}$, namely R_{kk}, then $L^A_I = R_6 M_I$ and

$$\begin{aligned}
L^B_0 &= R_0 M_0 + R_1 M_3 + R_2 M_6, \quad L^B_3 = R_1 M_0 + R_3 M_3 + R_4 M_6, \quad L^B_6 = R_2 M_0 + R_4 M_3 + R_5 M_6, \\
L^B_1 &= R_0 M_1 + R_1 M_4 + R_2 M_7, \quad L^B_4 = R_1 M_1 + R_3 M_4 + R_4 M_7, \quad L^B_7 = R_2 M_1 + R_4 M_4 + R_5 M_7, \\
L^B_2 &= R_0 M_2 + R_1 M_5 + R_2 M_8, \quad L^B_5 = R_1 M_2 + R_3 M_5 + R_4 M_8, \quad L^B_8 = R_2 M_2 + R_4 M_5 + R_5 M_8,
\end{aligned}$$

In conclusion, for the approach based on Cai's formula, the local expansions are stored in a vector of size 18 and involve 36 M2L operations with 9 different multipole expansions:

$$L'_I = L^B_I \text{ and } L'_{9+I} = L^A_I = R_6 M_I,\ I = 0 \dots 8$$

Mura's variant Let us recall the equation of the energy derived from Mura's formula

$$\begin{aligned}
M_{ij} &= \oint S(x') b'_j dx'_i \\
L^{AB}_{nj} &= R_{kk} M_{nj} \text{ and } L^C_i = R_{ij} \varepsilon_{jmn} M_{nm}
\end{aligned}$$

In this variant, 2 different local expansions have to be stored and contribute to the 3 terms of the energy. The first one is equal to the L^A in Cai's variant and can thus be written as

$$L^{AB}_I = R_6 M_I$$

Let us denote $M'_i = -\varepsilon_{imn} M_{mn}$ then the second kind of local expansion reads as

$$\begin{aligned}
L^C_0 &= R_{00} M'_0 + R_{01} M'_1 + R_{02} M'_2 \\
L^C_1 &= R_{10} M'_0 + R_{11} M'_1 + R_{12} M'_2 \\
L^C_2 &= R_{20} M'_0 + R_{21} M'_1 + R_{22} M'_2
\end{aligned}$$

In conclusion, for the approach based on Mura's formula, the local expansions are stored in a vector of size 12 and involve 18 M2L operations with 12 different multipole expansions:

$$\begin{aligned}
L'_0 &= R_0(M_7 - M_5) + R_1(M_2 - M_6) + R_2(M_3 - M_1) \\
L'_1 &= R_1(M_7 - M_5) + R_3(M_2 - M_6) + R_4(M_3 - M_1) \\
L'_2 &= R_2(M_7 - M_5) + R_4(M_2 - M_6) + R_5(M_3 - M_1) \\
L'_{3+I} &= R_6 M_I,\ I = 0 \dots 8
\end{aligned}$$

D.3 Numerical/analytical integration over segments

In this subsection we discuss the evaluation of the integral over the segments. The elastic stress field (2.3), respectively the force field (2.7) and energy (2.8) involves 1, respectively 2 successive, integrals over a line in $\mathcal{R}^3$. The evaluation of the integrand is done via polynomial interpolation, hence the numerical integration can be achieved exactly using a Gaussian quadrature with a well adjusted size. Let us recall the integral of an interpolated function in 1D over a subinterval $[x_1, x_2]$ of a larger interval $[a, b]$

$$\begin{aligned}\int_{[x_1,x_2]} f(x)dx \approx I_p &= \int_{[x_1,x_2]} \sum_{m=0}^{p} S^p(\Phi^{-1}(x), \bar{x}_m) f(\Phi(\bar{x}_m))dx \\ &= \sum_{m=0}^{p} f(\Phi(\bar{x}_m)) \int_{[x_1,x_2]} S^p(\Phi^{-1}(x), \bar{x}_m)dx\end{aligned}$$

A usual method to compute such integrals numerically is to use a Gaussian quadrature. This requires to map the interval of integration to $[-1, 1]$ first

$$\begin{aligned}I_p &= \sum_{m=0}^{p} f(\Phi(\bar{x}_m)) \int_{[-1,1]} S^p(\Phi^{-1}(\frac{x_2 - x_1}{2}X + \frac{x_1 + x_2}{2}), \bar{x}_m)\frac{x_1 + x_2}{2}dX \\ &= \sum_{m=0}^{p} f(\Phi(\bar{x}_m)) \int_{[-1,1]} S^p(\Phi^{-1}(\frac{L}{2}X + c_S), \bar{x}_m)\frac{L}{2}dX \\ &= \sum_{m=0}^{p} f(\Phi(\bar{x}_m)) \int_{[-1,1]} S^p(\frac{2}{b-a}((\frac{L}{2}X + c_S) - \frac{a+b}{2}), \bar{x}_m)\frac{L}{2}dX \\ &= \sum_{m=0}^{p} f(\Phi(\bar{x}_m)) \int_{[-1,1]} S^p(\frac{2}{W}((\frac{L}{2}X + c_S) - c_B), \bar{x}_m)\frac{L}{2}dX\end{aligned}$$

Once this is done the integral is computed by summing the results of evaluating the integrand at Gauss point X_q weighted by corresponding Gauss weights w_q such that

$$\int_{[-1,1]} g(X)dX \approx \sum_{q=1}^{Q} w_q g(X_q) \tag{D.7}$$

where Q denotes the size of the quadrature. Gaussian quadrature of size Q can achieve exact integration for polynomial integrands of order at most $2Q + 1$ which make them very competitive.

Index, Glossary and Bibliography

Acronyms

ACA Adative Cross Approximation. 27

ARRF Adaptive Randomized Range Finder. 40, 95

BBF structured Block Basis Factorization. 43

DD Dislocation Dynamics. 4, 47, 129, 131, 167

EVD Eigen Value Decomposition. 39, 109, 110, 116, 117, 130

FFT Fast Fourier Transform. 25, 27, 28, 52, 54, 57, 59, 90, 99, 129, 165

FJLT Fast Johnson Lindenstrauss Transforms. 42, 110

FMA Fast Multipole Algorithm. 10–14, 16, 173

FMM Fast Multipole Method. 3–5, 10, 11, 14, 16, 17, 20, 21, 23–25, 35, 42, 43, 46–48, 52, 58, 60, 61, 66, 67, 70, 71, 74, 76–80, 90, 94–100, 102, 106, 129–132, 136, 137, 146–148, 154, 155, 165, 167–169, 171, 173

GRF Gaussian Random Field. 5, 47, 88–90, 92, 95, 129

GRV Gaussian Random Variable. 88, 168

LRA Low-Rank Approximation. 4, 5, 38–43, 46–48, 103, 110, 113, 116, 129, 130

MDS Multidimensional Scaling. 48, 106, 107, 109–111, 113, 115–117, 120, 125, 130

MEKA Memory Efficient Kernel Approximation. 43

NGS New Generation Sequencing. 106, 110

NLA Numerical Linear Algebra. 3–5, 132

RSI Randomized Subspace Iterations. 40, 41, 95

rSVD Randomized SVD. 38–41, 90–95, 98–100, 102, 113, 116–126, 168, 169

SRFT Subsampled Randomized Fourier Transform. 42, 43

SRHT Subsampled Randomized Hadamard Transform. 42, 43

SVD Singular Value Decomposition. 27, 38–41, 43, 48, 90, 92, 93, 98, 100–103, 108–113, 115, 131

tSVD Truncated SVD. 114, 117

List of FMM steps

L2L Transfer Local expansion from parent to child cell. 13, 14, 16, 22–25, 56, 60, 63

L2P Transfer contribution from Local expansion to Particle. 13, 14, 16, 22–26, 56, 60, 63, 73, 77, 79, 95–97, 137, 138, 155

M2L Transfer from Multipole to Local expansion. xvii, xviii, 13–16, 20, 22–27, 46, 52–60, 63, 65, 66, 70, 73, 74, 76–78, 80, 95–97, 129, 130, 135, 137–139, 143, 149–151, 153, 155, 156, 165, 167, 169

M2M Transfer Multipole expansion from child to parent cell. 13, 14, 16, 22–25, 56, 60, 63

P2M Transfer contribution from Particle to Multipole expansion. 13, 14, 16, 22–26, 56, 60, 63, 73, 75, 77–79, 95–97, 137, 138, 155

P2P Transfer contribution from Particle to Particle. xvii, 14, 15, 22, 23, 25, 58–60, 79, 87, 95, 97–99, 143, 146, 169

List of FMM variants

bbfmm Black-Box FMM [54], *i.e.*, Chebyshev interpolation based FMM. xvi, 17, 21, 22, 24–26, 36, 46, 47, 51, 52, 58–67, 76–78, 81–83, 93, 97, 129, 167, 171, 173

compressed-bbfmm Black-Box FMM with global compression of the `M2L` operators. 26, 56, 63, 173

global fft Bivariate interpolation on a single domain using an equispaced grid followed by a Fast Fourier Transform. This method coïncindes with the Smooth Uniform FMM for $\bar{L} = 0$. xviii, 99, 145–148

kifmm Kernel-Independent FMM [127]. 17

smooth-ufmm Uniform FMM for globally smooth kernels. xvi, xviii, 51, 52, 58–62, 65–67, 71, 99–103, 130, 145–148, 167, 171

symmetric-bbfmm Black-Box FMM with reduction of the interaction list and individual compression of the `M2L` operators (only applicable to symmetric kernels). 26, 27, 63, 65, 167, 173

ufmm Uniform FMM, *i.e.*, Lagrange interpolation based FMM with Fast Fourier Transform (FFT) acceleration. xi, xiii, xvi, xviii, 46–48, 51, 52, 55–67, 76–78, 81–83, 93, 95, 97, 99–103, 129–131, 145–148, 167, 169, 171, 173

List of figures

List of tables

List of Algorithms

References

[1] Petter Abrahamsen. *A review of Gaussian random fields and correlation functions.* Norsk Regnesentral/Norwegian Computing Center, 1997.

[2] M. Abramowitz and I. A. Stegun. *Handbook of mathematical functions, with formulas, graphs, and mathematical tables.* Dover Publication, 1991.

[3] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

[4] Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. Adaptive sampling for k-means clustering. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 15–28. Springer, 2009.

[5] Nir Ailon and Bernard Chazelle. The fast johnson-lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on Computing*, 39(1):302–322, 2009.

[6] Sivaram Ambikasaran, Daniel Foreman-Mackey, Leslie Greengard, David W Hogg, and Michael O'Neil. Fast direct methods for gaussian processes and the analysis of nasa kepler mission data. *arXiv preprint arXiv:1403.6015*, 2014.

[7] Sivaram Ambikasaran and Michael O'Neil. Fast symmetric factorization of hierarchical matrices with applications. *arXiv preprint arXiv:1405.0223*, 2014.

[8] Athanasios Arsenlis, Wei Cai, Meijie Tang, Moono Rhee, Tomas Oppelstrup, Gregg Hommes, Tom G Pierce, and Vasily V Bulatov. Enabling strain hardening simulations with dislocation dynamics. *Modelling and Simulation in Materials Science and Engineering*, 15(6):553, 2007.

[9] S Aubry and A Arsenlis. Use of spherical harmonics for dislocation dynamics in anisotropic elastic media. *Modelling and Simulation in Materials Science and Engineering*, 21(6):065013, 2013.

[10] Adolfo J Banchio and John F Brady. Accelerated stokesian dynamics: Brownian motion. *The Journal of chemical physics*, 118(22):10323–10332, 2003.

[11] Mario Bebendorf. *Hierarchical matrices: a means to efficiently solve elliptic boundary value problems*, volume 63. Springer, 2008.

[12] Alberto Bellin and Yoram Rubin. Hydro_gen: A spatially distributed random field generator for correlated properties. *Stochastic Hydrology and Hydraulics*, 10(4):253–278, 1996.

[13] Serge Belongie, Charless Fowlkes, Fan Chung, and Jitendra Malik. Spectral partitioning with indefinite kernels using the nyström extension. In *European conference on computer vision*, pages 531–542. Springer, 2002.

[14] Edmund Bertschinger. Multiscale gaussian random fields and their application to cosmological simulations. *The astrophysical journal supplement series*, 137(1):1, 2001.

[15] Pierre Blanchard, Eric Darve, and Olivier Coulaud. Fast hierarchical algorithms for generating gaussian random fields. (preprint). 2015.

[16] Marc Bonnet. Boundary integral equation methods for solids and fluids. *Meccanica*, 34(4):301–302, 1999.

[17] Steffen Börm and Lars Grasedyck. Hybrid cross approximation of integral operators. *Numerische Mathematik*, 101(2):221–249, 2005.

[18] Steffen Börm, Maike Löhndorf, and Jens M Melenk. Approximation of integral operators by variable-order interpolation. *Numerische Mathematik*, 99(4):605–643, 2005.

[19] Christos Boutsidis and Alex Gittens. Improved matrix algorithms via the subsampled randomized hadamard transform. *SIAM Journal on Matrix Analysis and Applications*, 34(3):1301–1340, 2013.

[20] Christos Boutsidis, Michael W Mahoney, and Petros Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 968–977. Society for Industrial and Applied Mathematics, 2009.

[21] John P Boyd. Defeating the runge phenomenon for equispaced polynomial interpolation via tikhonov regularization. *Applied Mathematics Letters*, 5(6):57–59, 1992.

[22] John P Boyd. Trouble with gegenbauer reconstruction for defeating gibbs' phenomenon: Runge phenomenon in the diagonal limit of gegenbauer polynomial approximations. *Journal of Computational Physics*, 204(1):253–264, 2005.

[23] John P Boyd and Jun Rong Ong. Exponentially-convergent strategies for defeating the runge phenomenon for the approximation of non-periodic functions, part i: Single-interval schemes. commun. *Comput. Phys*, 5(2-4):484–497, 2009.

[24] John P Boyd and Jun Rong Ong. Exponentially-convergent strategies for defeating the runge phenomenon for the approximation of non-periodic functions, part two: Multi-interval polynomial schemes and multidomain chebyshev interpolation. *Applied Numerical Mathematics*, 61(4):460–472, 2011.

[25] John P Boyd and Fei Xu. Divergence (runge phenomenon) for least-squares polynomial approximation on an equispaced grid and mock–chebyshev subset interpolation. *Applied Mathematics and Computation*, 210(1):158–168, 2009.

[26] LM Brown. The self-stress of dislocations and the shape of extended nodes. *Philosophical Magazine*, 10(105):441–466, 1964.

[27] Vasily Bulatov and Wei Cai. *Computer simulations of dislocations*, volume 3. Oxford University Press, 2006.

[28] Wei Cai, Athanasios Arsenlis, Christopher R Weinberger, and Vasily V Bulatov. A non-singular continuum theory of dislocations. *Journal of the Mechanics and Physics of Solids*, 54(3):561–587, 2006.

[29] Julien Carron, Melody Wolk, and Istvan Szapudi. On fast generation of cosmological random fields. *Monthly Notices of the Royal Astronomical Society*, 444(1):994–1000, 2014.

[30] Fabien Casenave. An empirical interpolation based fast summation method for translation invariant kernels. *arXiv preprint arXiv:1408.0210*, 2014.

[31] S. Chaillat. *Fast Multipole Method for 3-D elastodynamic boundary integral equations. Application to seismic wave propagation.* PhD thesis, ENPC, 2008.

[32] S. Chaillat and M. Bonnet. A new fast multipole formulation for the elastodynamic half-space fundamental solution. PrePrint, 2012.

[33] J Chandrasekar, IS Kim, DS Bernstein, and AJ Ridley. Reduced-rank unscented kalman filtering using cholesky-based decomposition. *International Journal of Control*, 81(11):1779–1792, 2008.

[34] S Chandrasekaran, M Gu, XS Li, and J Xia. Some fast algorithms for hierarchically semiseparable matrices. *Report LBNL-62897 (Lawrence Berkeley Nat. Lab., Berkeley, 2007)*, 2008.

[35] T.F. Cox and M. A. A. Cox. *Multidimensional Scaling - Second edition*, volume 88 of *Monographs on Statistics and Applied Probability*. Chapman & al., 2001.

[36] Michael W Davis. Generating large stochastic simulations—the matrix polynomial approximation method. *Mathematical Geology*, 19(2):99–107, 1987.

[37] Michael W Davis. Production of conditional simulations via the lu triangular decomposition of the covariance matrix. *Mathematical Geology*, 19(2):91–98, 1987.

[38] Tomás Díaz de la Rubia and Vasily V Bulatov. Materials research by means of multiscale computer simulation. *MRS bulletin*, 26(03):169–175, 2001.

[39] Vin De Silva and Joshua B Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. *Advances in neural information processing systems*, pages 721–728, 2003.

[40] Vin De Silva and Joshua B Tenenbaum. Sparse multidimensional scaling using landmark points. Technical report, Technical report, Stanford University, 2004.

[41] R De Wit. Some relations for straight dislocations. *physica status solidi (b)*, 20(2):567–573, 1967.

[42] Vahid Dehdari and Clayton V. Deutsch. Applications of randomized methods for decomposing and simulating from large covariance matrices. In *Geostatistics Oslo 2012*, volume 17 of *Quantitative Geology and Geostatistics*, pages 15–26. Springer Netherlands, 2012.

[43] CR Dietrich and GN Newsam. A fast and exact method for multidimensional gaussian stochastic simulations. *Water Resources Research*, 29(8):2861–2869, 1993.

[44] Jürgen Dölz, Helmut Harbrecht, and Michael Peters. An interpolation-based fast multipole method for higher-order boundary elements on parametric surfaces. *International Journal for Numerical Methods in Engineering*, 2016.

[45] Eleni Drinea, Petros Drineas, and Patrick Huggins. A randomized singular value decomposition algorithm for image processing applications. In *Image Processing, Panhellenic Conference on Informatics, PCI*, volume 2001, page 279. Citeseer, 2001.

[46] Petros Drineas, Malik Magdon-Ismail, Michael W Mahoney, and David P Woodruff. Fast approximation of matrix coherence and statistical leverage. *The Journal of Machine Learning Research*, 13(1):3475–3506, 2012.

[47] Petros Drineas and Michael W Mahoney. On the nyström method for approximating a gram matrix for improved kernel-based learning. *The Journal of Machine Learning Research*, 6:2153–2175, 2005.

[48] JA Elliott. Novel approaches to multiscale modelling in materials science. *International Materials Reviews*, 56(4):207–225, 2011.

[49] William D Elliott and John A Board, Jr. Fast fourier transform accelerated fast multipole algorithm. *SIAM Journal on Scientific Computing*, 17(2):398–415, 1996.

[50] Björn Engquist, Lexing Ying, et al. A fast directional algorithm for high frequency acoustic scattering in two dimensions. *Communications in Mathematical Sciences*, 7(2):327–345, 2009.

[51] Arnaud Etcheverry. *Simulation de la dynamique des dislocations à très grande échelle*. PhD thesis, Université de Bordeaux, 2015.

[52] Geir Evensen. The ensemble kalman filter: Theoretical formulation and practical implementation. *Ocean dynamics*, 53(4):343–367, 2003.

[53] Christos Faloutsos and King-Ip Lin. *FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets*, volume 24. ACM, 1995.

[54] William Fong and Eric Darve. The black-box fast multipole method. *Journal of Computational Physics*, 228(23):8712–8725, 2009.

[55] J-M Frigerio, Frédéric Rimet, Agnès Bouchez, Emilie Chancerel, Philippe Chaumeil, Franck Salin, Sylvie Thérond, Maria Kahlert, and Alain Franc. diagno-syst: a tool for accurate inventories in metabarcoding. *arXiv preprint arXiv:1611.09410*, 2016.

[56] Andrej Gisbrecht and Frank-Michael Schleif. Metric and non-metric proximity transformations at linear costs. *Neurocomputing*, 167:643–657, 2015.

[57] Alex Gittens and Michael W Mahoney. Revisiting the nystrom method for improved large-scale machine learning. *arXiv preprint arXiv:1303.1849*, 2013.

[58] L. Goldfarb. A unified approach to pattern recognition. *Pattern Recognition*, 17(5):575–582, 1984.

[59] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.

[60] Leslie Greengard and Vladimir Rokhlin. *On the efficient implementation of the fast multipole algorithm*. Yale University, Department of Computer Science, 1988.

[61] D. Gusfield. *Algorithms on strings, trees, and sequences*. Cambridge University Press, Cambridge, UK, 1997.

[62] A Gutjahr, D McKay, and JL Wilson. Fast fourier transform methods for random field generation. *Eos Trans. AGU*, 68(44):1265, 1987.

[63] Wolfgang Hackbusch and Steffen Börm. H2-matrix approximation of integral operators by interpolation. *Applied numerical mathematics*, 43(1):129–143, 2002.

[64] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[65] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[66] A. J. Izenman. *Modern Multivariate Statistical Techniques*. Springer, NY, 2008.

[67] Mark E Johnson. *Multivariate statistical simulation: A guide to selecting and generating continuous multivariate distributions*. John Wiley & Sons, 2013.

[68] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[69] Andre G Journel. Geostatistics for conditional simulation of ore bodies. *Economic Geology*, 69(5):673–687, 1974.

[70] Simon J Julier and Jeffrey K Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[71] PK Kitanidis and J Lee. Principal component geostatistical approach for large-dimensional inverse problems. *Water resources research*, 50(7):5428–5443, 2014.

[72] Ladislas Kubin. *Dislocations, mesoscale simulations and plastic flow*, volume 5. Oxford University Press, 2013.

[73] Mickaële Le Ravalec, Benoît Noetinger, and Lin Y Hu. The fft moving average (fft-ma) generator: An efficient numerical method for generating and conditioning gaussian simulations. *Mathematical Geology*, 32(6):701–723, 2000.

[74] Ping Li, Trevor J Hastie, and Kenneth W Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 287–296. ACM, 2006.

[75] Edo Liberty. *Accelerated dense random projections*. PhD thesis, Citeseer, 2009.

[76] Edo Liberty, Nir Ailon, and Amit Singer. Dense fast random projections and lean walsh transforms. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 512–522. Springer, 2008.

[77] Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167–20172, 2007.

[78] Michael W Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning*, 3(2):123–224, 2011.

[79] Michael W Mahoney and Petros Drineas. Cur matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009.

[80] Aristotelis Mantoglou. Digital simulation of multivariate two-and three-dimensional stochastic processes with a spectral turning bands method. *Mathematical Geology*, 19(2):129–149, 1987.

[81] Aristotelis Mantoglou and John L Wilson. The turning bands method for simulation of random fields using line generation by a spectral method. *Water Resources Research*, 18(5):1379–1394, 1982.

[82] Per-Gunnar Martinsson. A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1251–1274, 2011.

[83] Per-Gunnar Martinsson and Vladimir Rokhlin. An accelerated kernel-independent fast multipole method in one dimension. *SIAM Journal on Scientific Computing*, 29(3):1160–1178, 2007.

[84] Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. A randomized algorithm for the approximation of matrices. Technical report, DTIC Document, 2006.

[85] Bertil Matérn. *Spatial variation*, volume 36. Springer Science & Business Media, 2013.

[86] Matthias Messner. *Fast Boundary Element Method in Acoustics*. PhD thesis, Technischen Universität Graz, Graz, AUSTRIA, 2012.

[87] Matthias Messner, Bérenger Bramas, Olivier Coulaud, and Eric Darve. Optimized m2l kernels for the chebyshev interpolation based fast multipole method. *arXiv preprint arXiv:1210.7292*, 2012.

[88] Matthias Messner, Martin Schanz, and Eric Darve. Fast directional multilevel summation for oscillatory kernels based on chebyshev interpolation. *Journal of Computational Physics*, 231(4):1175–1196, February 2012.

[89] Michael Meßner. A fast multipole galerkin boundary element method for the heat equation.

[90] Victor Minden, Anil Damle, Kenneth L Ho, and Lexing Ying. Fast spatial gaussian process maximum likelihood estimation via skeletonization factorizations. *arXiv preprint arXiv:1603.08057*, 2016.

[91] Mehryar Mohri and Ameet Talwalkar. Can matrix coherence be efficiently and accurately estimated? In *AISTATS*, pages 534–542, 2011.

[92] Toshio Mura. *Micro-mechanics of Defects in Solids*, volume 3. Springer, 1987.

[93] FRN Nabarro. Dislocations in a simple cubic lattice. *Proceedings of the Physical Society*, 59(2):256, 1947.

[94] N. Nishimura. Fast multipole accelerated boundary integral equation methods. *Appl. Mech. Rev.*, 55:299–324, 2002.

[95] Dean S Oliver. Moving averages for gaussian simulation in two and three dimensions. *Mathematical Geology*, 27(8):939–960, 1995.

[96] Lauren E Padilla and Clarence W Rowley. An adaptive-covariance-rank algorithm for the unscented kalman filter. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 1324–1329. IEEE, 2010.

[97] V. Y. Pan, J. H. Reif, and S. R. Tate. The power of combining the techniques of algebraic and numerical computing: Improved approximate multipoint polynomial evaluation and improved multipole algorithms. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, SFCS '92, pages 703–713, Washington, DC, USA, 1992. IEEE Computer Society.

[98] R Peierls. The size of a dislocation. *Proceedings of the Physical Society*, 52(1):34, 1940.

[99] E. Pekalska and R. P. W. Duin. *The dissimilarity representation for pattern recognition. Foundations and applications.* World Scientific, Singapore, 2005.

[100] Ue-Li Pen. Generating cosmological gaussian random fields. *The Astrophysical Journal Letters*, 490(2):L127, 1997.

[101] John Platt. Fastmap, metricmap, and landmark mds are all nystrom algorithms. In *AISTATS*, 2005.

[102] Theresa-Marie Rhyne, Melanie Tory, Tamara Munzner, Matthew O Ward, Chris Johnson, and David H Laidlaw. Information and scientific visualization: Separate but equal or happy together at last. In *IEEE Visualization*, volume 3, pages 611–614. Seattle, 2003.

[103] TJ Rivlin. The lebesgue constants for polynomial interpolation. In *Functional Analysis and its Applications*, pages 422–437. Springer, 1974.

[104] MJL Robin, AL Gutjahr, EA Sudicky, and JL Wilson. Cross-correlated random field generation with the direct fourier transform method. *Water Resources Research*, 29(7):2385–2397, 1993.

[105] Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *47th Annual IEEE Symposium on Foundations of Computer Science, 2006, FOCS'06.*, pages 143–152. IEEE, 2006.

[106] Martin Schanz. *Wave propagation in viscoelastic and poroelastic continua: a boundary element approach*, volume 2. Springer Science & Business Media, 2012.

[107] Masanobu Shinozuka and C-M Jan. Digital simulation of random processes and its applications. *Journal of sound and vibration*, 25(1):111–128, 1972.

[108] Si Si, Cho-Jui Hsieh, and Inderjit Dhillon. Memory efficient kernel approximation. In *Proceedings of The 31st International Conference on Machine Learning*, pages 701–709, 2014.

[109] P. D. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[110] G. Sylvand. *La méthode multipôle rapide en électromagnétisme : Performances, parallélisation, applications.* PhD thesis, ENPC, 2002.

[111] Ameet Talwalkar and Afshin Rostamizadeh. Matrix coherence and the nystrom method. *arXiv preprint arXiv:1004.2008*, 2010.

[112] Johannes Tausch. The variable order fast multipole method for boundary integral equations of the second kind. *Computing*, 72(3-4):267–291, 2004.

[113] Thomas Traub. A kernel interpolation based fast multipole method for elastodynamic problems.

[114] Joel A Tropp. Improved analysis of the subsampled randomized hadamard transform. *Advances in Adaptive Data Analysis*, 3(01n02):115–126, 2011.

[115] S. S. Vempala. *The Random Projection Method*, volume 65 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Sciences*. American Mathematical Society, 2004.

[116] Jason Tsong-Li Wang, Xiong Wang, King-Ip Lin, Dennis Shasha, Bruce A Shapiro, and Kaizhong Zhang. Evaluating a class of distance-mapping algorithms for data mining and clustering. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 307–311. ACM, 1999.

[117] Peng Wang, Chunhua Shen, and Anton van den Hengel. Efficient sdp inference for fully-connected crfs based on low-rank decomposition. *arXiv preprint arXiv:1504.01492*, 2015.

[118] Ruoxi Wang, Yingzhou Li, Michael W Mahoney, and Eric Darve. Structured block basis factorization for scalable kernel matrix evaluation. *arXiv preprint arXiv:1505.00398*, 2015.

[119] Shusen Wang and Zhihua Zhang. Improving cur matrix decomposition and the nyström approximation via adaptive sampling. *The Journal of Machine Learning Research*, 14(1):2729–2769, 2013.

[120] Zhiqiang Wang, Nasr Ghoniem, and Richard LeSar. Multipole representation of the elastic field of dislocation ensembles. *Physical Review B*, 69(17):174102, 2004.

[121] Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Proceedings of the 14th Annual Conference on Neural Information Processing Systems*, number EPFL-CONF-161322, pages 682–688, 2001.

[122] Andrew TA Wood and Grace Chan. Simulation of stationary gaussian processes in [0, 1] d. *Journal of computational and graphical statistics*, 3(4):409–432, 1994.

[123] David P Woodruff. Sketching as a tool for numerical linear algebra. *arXiv preprint arXiv:1411.4357*, 2014.

[124] Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3):335 – 366, 2008.

[125] Jianlin Xia, Shivkumar Chandrasekaran, Ming Gu, and Xiaoye S Li. Fast algorithms for hierarchically semiseparable matrices. *Numerical Linear Algebra with Applications*, 17(6):953–976, 2010.

[126] Z. Yang. *Computational Molecular Evolution*. Oxford Series in Ecology and Evolution. Oxford University Press, 2006.

[127] Lexing Ying, George Biros, and Denis Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *Journal of Computational Physics*, 196(2):591–626, 2004.

[128] Olivier Zahm. *Model order reduction methods for parameter-dependent equations– Applications in Uncertainty Quantification.* PhD thesis, École Centrale Nantes, 2015.

[129] Kai Zhang, Ivor W Tsang, and James T Kwok. Improved nyström low-rank approximation and error analysis. In *Proceedings of the 25th international conference on Machine learning*, pages 1232–1239. ACM, 2008.

[130] Degang Zhao, Jingfang Huang, and Yang Xiang. A new version fast multipole method for evaluating the stress field of dislocation ensembles. *Modelling and Simulation in Materials Science and Engineering*, 18(4):045006, 2010.