Large Scale Multidimensional Scaling for the Study of Biodiversity

Romain Peressoni

université de BORDEAUX

Inria

THÈSE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR
DE L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE

MATHÉMATIQUES ET INFORMATIQUE

MATHÉMATIQUES APPLIQUÉES ET CALCUL SCIENTIFIQUE

Par **Romain PERESSONI**

Positionnement multidimensionnel à grande échelle pour l'étude de la biodiversité

Sous la direction de : **Olivier COULAUD**
Co-directeur : **Emmanuel AGULLO**

Soutenue le mardi 13 juin 2023

Membres du jury :

M. Emmanuel AGULLO	Chargé de recherche	Inria Bordeaux	Co-Directeur
M. Olivier COULAUD	Directeur de recherche	Inria Bordeaux	Directeur
M. Alain FRANC	Directeur de recherche	Inrae, Inria Bordeaux	Invité
M. Pierre MAGAL	Professeur	Université de Bordeaux	Invité
Mme Sandrine MOUYSSET	Maîtresse de conférences	Université Toulouse III	Examinatrice
M. Raymond NAMYST	Professeur	Université de Bordeaux	Président
M. Emmanuel PARADIS	Directeur de recherche	IRD	Rapporteur
M. Bruno RAFFIN	Directeur de recherche	Inria Grenoble Alpes	Rapporteur
M. Gaël VAROQUAUX	Directeur de recherche	Inria Saclay	Examinateur

Positionnement multidimensionnel à grande échelle pour l'étude de la biodiversité

Résumé : Le positionnement multidimensionnel (MDS) est un algorithme classique de réduction de dimensions et une méthode de visualisation. La MDS prend une matrice de dissimilarités (ou de distances) en entrée et produit un nuage de points dans une dimension inférieure. Chaque objet présent dans la matrice d'entrée est associé à un point dans le nuage de points, la distance entre les points reflétant au mieux la distance d'entrée. La principale étape de calcul de la MDS est une symmetric eigenvalue decomposition (sEVD). Blanchard et al. (2016) ainsi que Paradis (2018) ont montré que cette étape sEVD de la MDS pouvait être réalisée avec succès à l'aide de techniques de randomisation. Deux algorithmes de randomisation ont été considérés, à savoir la randomized singular value decomposition (RSVD) et la randomized sEVD (RsEVD) (voir Halko et al. (2011)). Pour traiter des problèmes de tailles encore plus importantes, une MDS à haute performance basée sur ces techniques de randomisation a été récemment proposée, basée sur le formalisme RSVD.

Le chapitre 1 présente ces résultats comme contexte de la présente thèse. La première partie de cette thèse, présentée dans le chapitre 2, revient sur ces résultats. Nous ouvrons ce chapitre en évaluant l'impact du choix de la technique de randomisation pour réaliser la sEVD sur le comportement numérique de la MDS. D'une part, la RSVD standard peut (légèrement) briser la symétrie implicitement supposée dans la MDS mais ne nécessite qu'une seule projection. D'autre part, la RsEVD standard préserve la symétrie mais nécessite deux projections. Une étude expérimentale menée sur un ensemble de données de biodiversité à petite échelle confirme que les deux approches peuvent être pertinentes. Cela nous incite à proposer une version HPC de la RsEVD en plus de l'algorithme RSVD déjà disponible. Nous sommes donc maintenant en mesure d'effectuer des MDS HPC avec l'une ou l'autre variante d'algorithme aléatoire. Ces deux variantes sont dominées par des produits de matrices (MM). Leurs performances et leur consommation de mémoire sont donc des clés pour une MDS basée sur de tels algorithmes. Nous intégrons donc une MM générale à base de tâches (GEMM) récemment proposée ainsi qu'une nouvelle MM symétrique à base de tâches (SYMM). La MDS qui en résulte est nettement plus performante que la version de base du chapitre 1 sur un ensemble de données de biodiversité à grande échelle.

La deuxième partie de cette thèse traite de la comparaison des nuages de points résultant de MDS à grande échelle et envisage la possibilité de travailler sur des nuages de points réduits. Le chapitre 3 traite de la comparaison de nuages de points issus de MDS à grande échelle, tels que ceux obtenus dans la première partie de la thèse. Nous montrons qu'une analyse Procustéenne peut être réalisée efficacement avec seulement quelques points de repère partagés, ce qui permet de réduire considérablement la partie de la matrice d'entrée à calculer. Le chapitre 4 vise à construire une MDS réduite qui opère sur un sous-ensemble de points du nuage. L'algorithme peut être apparenté à de l'échantillonnage uniforme. Cependant, contrairement à la plupart des publications sur l'échantillonnage qui visent à compresser un échantillon original, nous construisons un échantillon d'un modèle sans supposer que nous disposons d'un échantillon original et sans connaître le modèle. La technique de référence utilisée dans les chapitres 3 et 4 exige que nous calculions quelques éléments hors diagonaux en plus des grands blocs diagonaux. Le chapitre 5 examine la possibilité de le faire sans aucun élément hors (blocs) diagonaux.

Mots-clés : Positionnement multidimensionnel (MDS), plongement linéaire aléatoire, échantillonnage uniforme, décomposition symétrique en valeurs propres, calcul haute performance (HPC), programmation à base de tâches, produits de matrices , SYMM, MDS réduite, distance de Hausdorff, biodiversité, métabarcoding, comparaison de nuages de points, réduction de dimension, algèbre linéaire numérique, théorie de perturbation de matrice

Équipe-projet CONCACE
Centre Inria de l'université de Bordeaux
200 avenue de la vielle Tour
33405 Talence, France.

Large Scale Multidimensional Scaling for the Study of Biodiversity

Abstract: Multidimensional scaling (MDS) is a classical dimension reduction algorithm and visualization method. MDS takes a matrix of dissimilarities (or distances) as input and produces a point cloud in lower dimension. Each object present in the input matrix is associated with a point in a Cartesian space, the pairwise distance between the points preserving as well as possible the distance between the entries. The main computational step of MDS is a symmetric eigenvalue decomposition (sEVD). Blanchard et al. (2016) as well as Paradis (2018) showed that this sEVD step of the MDS could be successfully performed with randomization techniques. Two randomization algorithms have been considered, namely, the randomized singular value decomposition (RSVD) and randomized sEVD (RsEVD) (see Halko et al. (2011)). To process problems of even larger size, a high-performance MDS design based on these randomization techniques has furthermore been recently proposed, based on the RSVD formalism. Chapter 1 present these results as the background of the present thesis.

The first part of this thesis, presented in Chapter 2, revisits these results. We open this chapter by assessing the impact of the choice of the randomized technique for performing the sEVD on the numerical behaviour of the MDS. On the one hand, the standard RSVD may (slightly) break the implicitly assumed symmetry of the MDS but requires only a single projection. On the other hand, the standard RsEVD preserves the symmetry but requires two projections. An experimental study conducted on a small-scale biodiversity dataset confirms

that both approaches may be relevant. This motivates us to propose a HPC design of RsEVD in addition to the available RSVD algorithm described in background. We are thus now able to perform HPC MDS with either variant of the randomized algorithm. Both variants are dominated by matrix-matrix multiplications (MM). Their performance and memory consumption are thus keys for that of the overall MDS based on such randomized algorithms. We thus incorporate a recently proposed task-based general MM (GEMM) as well as a new task-based symmetric MM (SYMM). The resulting MDS significantly outperforms the original background version from Chapter 1 on a large-scale biodiversity dataset.

The second part of this thesis deals with the comparison of point clouds resulting from large scale MDS and considers the possibility of working on reduced point clouds. Chapter 3 deals with the comparison of point clouds from large scale MDS, such as those obtained in the first part of the thesis. We show that a Procrustes analysis can be efficiently performed with only a few shared landmarks, significantly reducing the portion of the input matrix to be computed. Chapter 4 aims to construct a reduced MDS that operates on a subset of the points of the cloud. The algorithm can be related to uniform randomized matrix sampling. However, unlike most of the literature on randomized matrix sampling, which aims to compress an original sample, we build a sample of a model without assuming that we have an original representative (larger) sample and without knowing the model. The landmark technique used in chapters 3 and 4 requires us to compute a few off-diagonal elements in addition to large diagonal blocks. Chapter 5 investigates whether it is possible to do this without any off-(block-)diagonal elements at all.

Keywords: multidimensional scaling (MDS), randomized linear embedding, uniform randomized sampling, symmetric eigenvalue decomposition, singular value decomposition, high-performance computing (HPC), task-based programming, matrix multiplication, SYMM, reduced MDS, Hausdorff distance, biodiversity, metabarcoding, point cloud comparison, dimension reduction, numerical linear algebra, matrix perturbation theory

CONCACE Project-team
Centre Inria de l'université de Bordeaux
200 avenue de la vielle Tour
33405 Talence, France.

Acknowledgements

I consider myself lucky to have been surrounded by many kind, supportive and overall amazing people during the journey that was this Ph.D. In this section, I want to take the time to thank them for the positive impact they had not only on this work but sometimes on my life as well.

First I want to thank my family: my parents Catherine and Christophe, as well as my grandparents Mireille and Marc, for always supporting me in any and every way they could. I am also very grateful to the rest of my family, even though not everyone could come to my defense, I am glad I could count on the support of my brothers and sister Matthis, Margaux, Guillaume and Joshua, my brother in law Lucas, my other grandmothers Suzanne and Paulette, my aunts and uncles Jean-Claude, Dominique, Philippe, Agathe, Sylvie, Cathy, my cousins Thomas, Lucas, Valentina, Madlie, Gabriel, Anaïs, Matthieu and Timothée. I also want to thank my first cousin once removed Brigitte, as well as Bruno and my godmother Marion.

Among all the friends that supported me throughout this journey, I want to thank in particular Baptiste Merliot and Matthieu Simonin. The regular encounters I had with either one of you really allowed me to clear my mind of work related matters when I needed it.

I also want to thank the people from the CONCACE team (and by extension the Topal team as well) for providing me with such a warm and welcoming work environment. I obviously have to start by thanking my fellow Ph.D students (now all doctors) Marek Felšöci, Martina Iannacito, Mathieu Vérité and Yanfei Xiang as we started our respective adventures together. I have many amazing memories of our time together both at inria and during our outings together. Of course, this sentiment also extends to others who started before, Aurélien Falco, Esragül Korkmaz and Alena Shilova or after, Jean-François David and Xunyi Zhao. I also want to thank everyone else from both teams, Luc, Gilles, Pierre (Esterie), Julia, Mathieu, Pierre (Ramet), Lionel, Abdou, Olivier. It was a real pleasure to work alongside you all. Thank you Antoine Gicquel for your last minute help in preparing my defense. I also want to thank the members of the SED team, in particular Ludovic Courtès for your help with guix as well as Florent Pruvost for your constant help (on way too many issues to list here) throughout my work. Thank you as well to Alfredo Buttari and Antoine Jego, I learned a lot in the course of our collaboration.

Last, but far from least, I want to thank my advisors Olivier Coulaud and Emmanuel Agullo. Olivier, thank you for your help, patience and support. Your explanations were always welcome and accessible and your advice wise. Manu, thank you for everything you did for me. Our conversations, whether they were about work or not, whether they were serious or just goofing around, were always pleasant. I learned a lot from you, you taught me so much about linear algebra, you showed me how to write and present my work with care. You pushed me to not settle for anything less than greatness, and as a result I can now look back on this work with pride.

Again, to everyone, thank you from the bottom of my heart.

Résumé en français

On s'attend souvent à ce que les points d'un espaces de grande dimension fassent partie d'une variété inconnue de dimension faible, la question étant alors de savoir comment la révéler. De nombreuses méthodes ont été proposées pour y parvenir, telles que la cartographie de Sammon, l'analyse en composantes curvilignes, le *stochastic neighbour embedding* (et t-SNE), l'*isomap* ou encore les *Laplacian eigenmaps*. Ces méthodes atteignent leurs limites en terme de temps d'exécution à partir d'environ 10 000 éléments. Une autre méthode classique est le positionnement multidimensionnel (MDS), qui vise à transformer les distances entre un ensemble de m éléments en une configuration de m points dans un espace cartésien.

La motivation initiale de cette thèse ainsi que les expériences numériques qui y figurent sont basées sur des données relatives à la biodiversité. En particulier, notre travail s'inscrit dans le contexte du *metabarcoding*, une branche de la biologie moléculaire qui cherche à l'identifier des unités taxonomiques opérationnelles (OTU, un groupe d'individus proches génétiquement) à partir d'échantillons contenant de grandes quantités de matériel génétique obtenu par séquençage ADN à grande échelle. Il s'agit d'une extension de la méthode de *DNA barcoding* qui est l'étude et l'identification d'espèces à partir de matériel ADN. Le *metabarcoding* permet d'étudier la biodiversité lorsqu'il n'est pas possible d'identifier directement les individus ou lorsque le nombre d'individus est trop important pour être classé à l'aide d'autres méthodes. Une MDS sur une matrice de distance obtenue à partir d'un ensemble de séquences ADN est une façon d'étudier ces échantillons. Les nuages de points résultant de la MDS sur de tels échantillons peuvent alors être utilisés pour l'étude de la biodiversité.

D'un point de vue algébrique (simplifié), la MDS cherche à transformer une matrice de distance $D \in \mathbb{R}^{m \times m}$ en une matrice de position $X \in \mathbb{R}^{m \times k_{MDS}}$ souvent appelée matrice de coordonées. La matrice X peut alors être interprétée comme un ensemble de m points (les lignes de X) en dimension k_{MDS}. Souvent, on choisit $k_{MDS} = 2$ afin de permettre une représentation graphique de cette matrice de coordonnées sous la forme d'un nuage de points en deux dimensions.

D'un point de vue calculatoire (également simplifié), la première étape de la MDS consiste à calculer une matrice de *produits scalaires* G à partir de la matrice D, cette matrice de produits scalaires est souvent appelée matrice de *Gram* dans la littérature. La deuxième étape de la MDS consiste en une décomposition symétrique en valeurs propres (sEVD) de G qui permet de calculer X. La MDS est présentée de manière plus complète dans le Chapitre 1.

Lorsque l'on considère de très grands jeux de données, utiliser une sEVD déterministe standard sur l'ensemble de la matrice de Gram peut s'avérer hors de portée en raison de contraintes d'empreinte mémoire ou de temps d'exécution. Pour ces raisons, les MDS à grande échelle ont souvent été traitées par des approches heuristiques, qui peuvent donner de bons résultats dans la pratique, bien qu'aucune garantie ne puisse être donnée quant à leur qualité. Pour effectuer des MDS robustes à grande échelle, il est possible d'utiliser des algorithmes aléatoires. Il existe plusieurs classes de ces méthodes. Une première classe d'application des méthodes aléatoires à la MDS sont les méthodes de *Nyström*. Dans le contexte de la MDS, ces méthodes ont été introduites sous le nom de Landmark MDS (LMDS) par Silva et Tenenbaum (2002). La LMDS

consiste à effectuer une MDS sur une sous-matrice principale de la matrice de distance et à interpoler les inconnues restantes. Une deuxième classe d'application des algorithmes aléatoires à la MDS est l'approximation par *échantillonnage*. Elle a été introduite dans l'écosystème MDS sous le nom de Pivot MDS par Brandes et Pich (2006) et a été développée dans la thèse de Klimenta (2012). Il s'agit d'effectuer une MDS partielle sur une sous-matrice principale. La troisième catégorie d'application des méthodes aléatoires à la MDS est le plongement linéaire (aussi connu sous le nom de projection aléatoire dans les années 2000 et 2010). L'algorithme de plongement linéaire aléatoire le plus connu est la décomposition aléatoire en valeurs singulières (RSVD). Cette méthode consiste à effectuer une décomposition en valeurs singulières (SVD) par le biais d'une approche probabiliste rapide, en garantissant la qualité de la solution. Son utilisation dans le cadre de la MDS (RSVD-MDS) a été étudiée par Blanchard et al. (2016) ainsi que Paradis (2018) et a permis de traiter de grands ensembles de données tout en préservant la robustesse numérique de la sEVD-MDS standard.

Dans le cadre de cette thèse, nous avons porté un intérêt particulier au schéma d'accès aux données de ces différentes méthodes. En effet, le coût du calcul de la matrice de distance est dans beaucoup de cas plus important que celui du calcul de la MDS en elle-même. Dans ce cas, limiter le nombre d'entrées de la matrice de distance auxquelles il est nécessaire d'accéder (et donc qu'il est nécessaire de calculer) peut avoir un impact beaucoup plus important que l'optimisation de la MDS directement. Le Chapitre 2 de cette thèse se concentre sur les méthodes de plongement linéaire, qui nécessitent l'accès à toute la matrice de distance dans le cadre d'un produit de matrices avec une matrice gaussienne. La LMDS (qui correspond donc aux méthodes de *Nyström*) est introduite dans le Chapitre 3. Cette méthode nécessite en théorie uniquement de calculer une bande de taille $(k_{MDS} + 1) \times m$ de la matrice de distance. Les chapitres 4 et 5 traitent des méthodes d'échantillonnages. La méthode la plus simple, qui est également celle que nous considérons ici consiste à sélectionner aléatoirement (à une permutation près) une sous-matrice principale de la matrice de distance, on parle alors d'*uniform sampling*. Il est cependant important de noter que dans la littérature, la plupart des méthodes présentées commencent par donner un poids aux différentes lignes et colonnes de la matrice avant de faire un échantillonage qui prend en compte ces scores, dans ce cas on parle d'*importance sampling*. Bien que cette étape préliminaire permette de donner un meilleur échantillonage *a priori* qu'un simple échantillonage uniforme, de tels scores ne peuvent être obtenus qu'au prix du calcul de toute la matrice de distance.

Au delà des méthodes randomisées présentées, une autre classe d'algorithmes utilisés pour réaliser des MDS à grande échelle se base sur des méthodes de type *divide-and-conquer*. Le principe de ces méthodes est de calculer des MDS indépendantes sur des sous-matrices principales qui se recouvrent puis d'aligner les résultats à l'aide de l'analyse Procrustéenne. Cette méthode a été introduite par Yang et al. sous le nom de FastMDS. Ketpreechasawat a indépendament présenté une méthode similaire appelée *hierarchical landmark charting*. Lee et Choi présentent ces méthodes sous le nom Landmark MDS Ensemble (LMDSE). Ces méthodes ne sont cependant pas étudiées dans le cadre de cette thèse.

Le calcul de MDS à grande échelle avec ces algorithmes nécessite donc une attention particulière dans leur conception. Agullo et al. ont récemment proposé une MDS *high-performance computing* (HPC) basée sur le plongement linéaire. D'un point de vue numérique, cette MDS correspond à la RSVD-MDS de Blanchard et al. Du point de vue du HPC, cette MDS consiste en une implémentation à base de tâches de l'algorithme RSVD-MDS au sein d'une pile logicielle efficace comprenant des solveurs numériques, des systèmes d'exécution et des couches de communication à la pointe de la technologie. Il en résulté la possibilité d'obtenir une MDS robuste et applicable à de grands ensembles de données sur des supercalculateurs modernes. Cette thèse peut être considérée comme la continuation de ce travail, que nous présentons en arrière-plan dans le Chapitre 1.

La première partie de cette thèse, présentée au Chapitre 2, revisite les différentes conceptions numériques et HPC de la MDS basés sur le plongement linéaire, présentée dans le Chapitre 1. Nous débutons ce chapitre en évaluant l'impact du choix de la technique de plongement linéaire aléatoire pour réaliser l'étape de sEVD sur le comportement numérique de la MDS. D'une part, la RSVD standard peut (légèrement) impacter la symétrie implicitement supposée de la matrice de distance, mais ne nécessite qu'une seule projection. D'autre part, la sEVD randomisée standard (RsEVD) préserve la symétrie mais nécessite deux projections. Une étude expérimentale menée sur un ensemble de données de biodiversité à petite échelle confirme que les deux approches peuvent être pertinentes. Ceci nous motive à proposer une conception HPC de la RsEVD en plus de l'algorithme RSVD déjà disponible. Nous sommes alors en mesure de proposer une MDS HPC utilisant les variantes RSVD et RsEVD de l'algorithme de plongement linéaire randomisé. Ces deux variantes étant dominées par des produits de matrices (que l'on appelle MM pour *matrix multiplication*) leurs performances et leur consommation mémoire sont donc clés pour une MDS basée sur de tels algorithmes. Nous incorporons donc une MM générale à base de tâches (GEMM) récemment proposée ainsi qu'une nouvelle MM symétrique à base de tâches (SYMM). Nous montrons expérimentalement que dans le cas distribué, les bibliothèques standard obtiennent des performances moindres avec SYMM qu'avec GEMM. Nous montrons également qu'une conception efficace des schémas de communication peut réduire considérablement cet écart. Enfin, nous montrons qu'une partie de cet écart s'explique par une intensité arithmétique (IA) plus faible du SYMM 2D Bloc Cyclique (2D BC) par rapport au GEMM 2D (d'un facteur de 2). Nous considérons deux distributions de données alternatives, SBC et TBC. SBC est une adaptation directe au cas de la multiplication de matrices d'une étude sur la décomposition de Cholesky. TBC est une adaptation dans le cadre distribué (et parallèle) des idées sous-jacentes de TBS, un algorithme séquentiel *out-of-core*. Nous prouvons que SBC et TBC améliorent l'IA de SYMM d'un facteur de $\sqrt{2}$ et 2 respectivement, égalant ainsi en particulier celle de 2D BC GEMM pour ce dernier. Nous évaluons la MDS résultante sur un ensemble de données de biodiversité à grande échelle correspondant à des échantillons de diatomées prélevés dans le lac Léman à intervalles mensuels. Nous comparons également cette MDS améliorée avec la MDS HPC de référence présentée au Chapitre 1.

Les codes résultants de ces travaux ont été intégrés dans la bibliothèque *fmr* utilisée dans le cadre de la MDS du Chapitre 1. Alors qu'il fallait auparavant choisir entre la performance (GEMM), ou une meilleure consommation mémoire (SYMM), nous avons montré que notre approche à base de tâches combinée à la nouvelle distribution TBC permettent désormais d'obtenir une implémentation de SYMM dont la performance est compétitive avec celle de GEMM. Cette étude montre également que les algorithmes impliquant des distributions de données et de tâches très irrégulières peuvent être mis en œuvre avec un code facile à écrire, à lire et à maintenir, tout en garantissant une performance compétitive.

La deuxième partie de cette thèse traite de la comparaison de nuages de points, soit comme une fin en soi (chapitres 3 et 5), soit comme un moyen de développer un nouvel algorithme itératif d'échantillonnage aléatoire (Chapitre 4).

Le Chapitre 3 vise spécifiquement à comparer des nuages de points issus de MDS à grande échelle, tels que ceux obtenus dans la première partie de la thèse. Le problème principal pour comparer des nuages de points provenant de MDS est que différentes exécutions de la MDS peuvent résulter en des translations, rotations ou réflections des nuages de points en sortie, la MDS n'étant définie qu'à une isométrie près. Ce problème est lié au problème bien connu de l'analyse Procrustéenne. Nous montrons que l'analyse Procrustéenne peut être réalisée efficacement avec seulement quelques repères communs aux différents nuages, ce qui réduit considérablement la quantité de la matrice d'entrée à calculer.

L'alignement des nuages de points à des fins de comparaison permet également de superposer les nuages de points alignés et de reconstruire le nuage de points complet. Cela peut être

considéré comme une nouvelle vision de l'algorithme de FastMDS, mais motivé à l'origine par la comparaison des nuages de points intermédiaires plutôt que par le calcul direct de la MDS. Nous appelons cette méthode *Block Diagonal Landmark Procrustes MDS* (BDLPMDS).

Une fois les nuages de points alignés les uns avec les autres, nous pouvons également calculer une distances entre eux à l'aide de la distance de Hausdorff ou ses variations. Nous proposons une nouvelle variation de la distance de Hausdorff que nous appelons *Squared Modified Hausdorff distance* basée sur la *Modified Hausdorff distance*, une variation existante de la distance de Hausdorff, que nous avons construit pour être analogue à la distance de Frobenius sous certaines conditions. Nous introduisons également le concept de distance de Hausdorff relative, qui permet de définir une distance indépendante de la taille des données.

Les méthodes présentées sont utilisées pour comparer des données venant de diatomées du lac Léman, séparés en 10 jeux de données $L_1, L_2, \ldots, L_{10}$ composés chacun d'environ 100 000 éléments auxquels nous appliquons l'algorithme de BDLPMDS pour reconstruire le nuage de points complet. Grâce à cette méthode, nous pouvons effectuer une MDS sur cette matrice de taille 1 000 000 en utilisant uniquement un nœud de calcul, alors qu'il en faut plus de 100 pour faire ce même calcul par RSVD-MDS (ou RsEVD-MDS) complète. De plus, cette méthode nécessitant uniquement les blocs diagonaux (ainsi qu'un petit nombre de points supplémentaires servant de références), cette méthode de calcul de la MDS permet d'obtenir le nuage de points recherché en utilisant uniquement une fraction de la matrice de distance, dont le calcul est le principal obstacle pour le calcul de MDS à grande échelle.

Le chapitre 4 traite de l'échantillonnage aléatoire *uniforme* pour construire une MDS réduite, c'est-à-dire un sous-ensemble de points du nuage. Comme mentionné ci-dessus, la plupart des méthodes d'échantillonnage discutées dans la littérature évaluent d'abord l'importance des entrées de la matrice avant de sélectionner aléatoirement (et éventuellement de mettre à l'échelle) un sous-ensemble de points en fonction de leurs poids respectifs. Ces méthodes d'*importance sampling* nécessitent l'accès à l'ensemble de la matrice de distance D. Leur force réside dans le fait qu'elles offrent d'excellentes garanties d'erreur *a priori*. Ces garanties dépendent du spectre de la matrice d'entrée. Bien que nous ne connaissions pas le spectre a priori, dans la pratique, les premières valeurs propres (positives) ont une forte décroissance. Cette dépendance théorique ne devrait donc pas poser de problème majeur dans la pratique. Un problème plus fondamental est que la taille de l'échantillon de référence auquel nous avons affaire peut être surestimée ou sous-estimée afin d'obtenir un résultat de la qualité que l'on souhaite obtenir. En effet, bien que cette thèse ne porte que sur le traitement d'une matrice de dissimilarité D, en pratique, la construction d'une telle matrice pour l'étude d'un échantillon de référence est une tâche difficile. Par conséquent, l'approximation de l'échantillon de référence par un échantillon réduit n'est peut-être pas la bonne question à traiter. Une autre question est de savoir si l'on peut estimer la qualité intrinsèque d'un échantillon. Cette nouvelle question est liée à l'utilisation d'estimateurs d'erreur *a posteriori*. Associés à des schémas itératifs, ces estimateurs sont préconisés par Martinsson et Tropp lorsque l'échantillonnage manque de garanties précises. Le chapitre 4 explore l'utilisation des distances considérées dans le chapitre 3 comme moyen de construire des estimateurs d'erreur *a posteriori* pour la valeur intrinsèque de l'échantillon. L'estimateur est calculé en utilisant une technique de *rééchantillonnage*.

Sur la base de ces résultats, nous avons proposé des algorithmes MDS itératifs qui utilisent un critère d'arrêt basé sur la distance entre les nuages de points rééchantillonnés à chaque itération. Nous avons proposé quatre algorithmes, chacun suivant la même étape de rééchantillonnage et de calcul de distance, différant uniquement par la manière dont ils calculent la MDS d'une itération à la suivante. Tout d'abord, nous avons proposé deux algorithmes conservateurs qui recalculent la MDS à chaque itération sans garder en mémoire le résultat de l'itération précédente. Nous avons également proposé deux algorithmes progressifs basés sur la LMDS et la BDLPMDS, qui visent à construire l'itération suivante en étendant le résultat de

l'itération précédente. Ces deux approches montrent des résultats prometteurs.

La technique de *landmarks* utilisée dans les chapitres 3 et 4 nécessite le calcul de certains éléments hors blocs diagonaux en plus des grands blocs diagonaux. Le chapitre 5 étudie la possibilité de le faire sans aucun élément hors (bloc)-diagonale dans le cas où les nuages de points à comparer sont très proches. Cette proximité est possible lorsque l'on considère des nuages de points extraits de manière indépendante et identiquement distribuée à partir d'un plus grand échantillon. Nous montrons que lorsque cette condition est respectée, les nuages de points représentent la même population et la MDS capture correctement les nuages de points.

Contents

Introduction

Points in large dimension spaces are often expected to live on an unknown small dimensional manifold, and the question is how to reveal it. Many methods have been proposed to do so such as Sammon mapping, curvilinear component analysis, stochastic neighbour embedding and t-SNE, isomap, Laplacian eigenmaps, just to present the diversity of available methods (here, nonlinear), with a global survey [84]. Those methods reach limits in time for, say, 10,000 items or more. An old classical method is the multidimensional scaling (MDS) [109, 132, 125], reviewed in [29]. It aims at translating information about the pairwise distances among a set of m items into a configuration of m points mapped into a Cartesian space.

From a high-level algebraic point of view, it consists of processing a dissimilarity matrix $D \in \mathbb{R}^{m \times m}$ to obtain a matrix $X \in \mathbb{R}^{m \times k_{MDS}}$, as illustrated in Figure 1. The matrix X can be interpreted as a collection of m points (the rows of X) in a space $\mathbb{R}^{k_{MDS}}$ of dimension k_{MDS}, and is often referred to as the *coordinate* matrix. Figure 2 shows an example for $m = 4$ items mapped into a space of dimension $k_{MDS} = 2$.

As shown in Figure 3, from a high-level (simplified) computational point of view, MDS first computes an *inner-product* matrix G from the dissimilarity matrix D. The inner-product matrix is often referred to as the *kernel* matrix or *Gram* matrix in the literature. We will follow the latter terminology, hence the notation G in this thesis. The second step then essentially consists in the symmetric eigenvalue value decomposition (sEVD) [102, 56] of G to compute the coordinate matrix X. MDS will be further introduced in Section 1.1 and we restrict our discussion to computational aspect in the rest of this introduction.

When dealing with large datasets, processing the second step with a standard, deterministic sEVD on the whole Gram matrix may be out of reach due to memory or time to solution constraints. For these reasons, large-scale MDS have often been processed with heuristic approaches, which may yield good results in practice, though no guarantee can be provided on their quality. Randomized algorithms have changed the game, allowing robust MDS to be performed at large scale. A first class of application of randomized methods to MDS has been *Nyström* approximation. In the context of MDS, it has been introduced under the name of Landmark MDS (LMDS) by Silva and Tenenbaum [118, 31]. It consists of performing a partial MDS on a principal submatrix and interpolate the remaining unknowns. A second class of

Figure 1 – High-level algebraic view of the MDS.

$$D = \begin{pmatrix} 0 & 1.1 & 5.3 & 5.01 \\ 1.1 & 0 & 4.97 & 4.49 \\ 5.3 & 4.97 & 0 & 0.95 \\ 5.01 & 4.49 & 0.95 & 0 \end{pmatrix}$$

$$X = \begin{pmatrix} -2.43 & 0.51 \\ -2.51 & -0.47 \\ 2.66 & 0.43 \\ 2.28 & -0.46 \end{pmatrix}$$

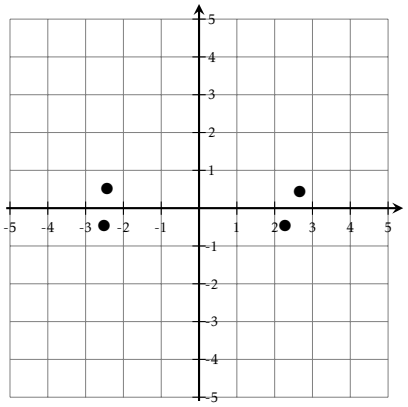


Figure 2 – MDS on $m = 4$ items mapped into a space of dimension $k_{MDS} = 2$. The input dissimilarity matrix $D \in \mathbb{R}^{4\times4}$ is translated into a coordinate matrix $X \in \mathbb{R}^{4\times2}$ representing the $m = 4$ items in $k_{MDS} = 2$ dimensions.

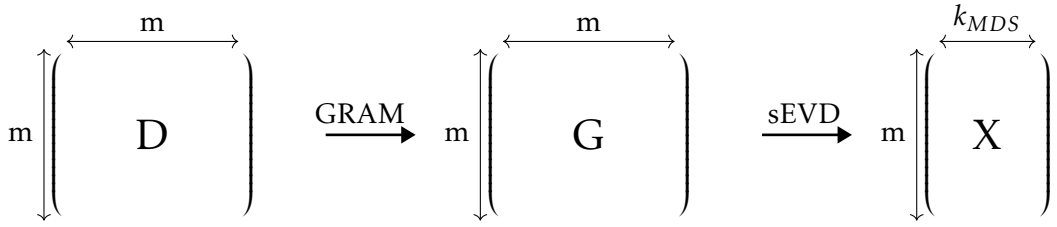
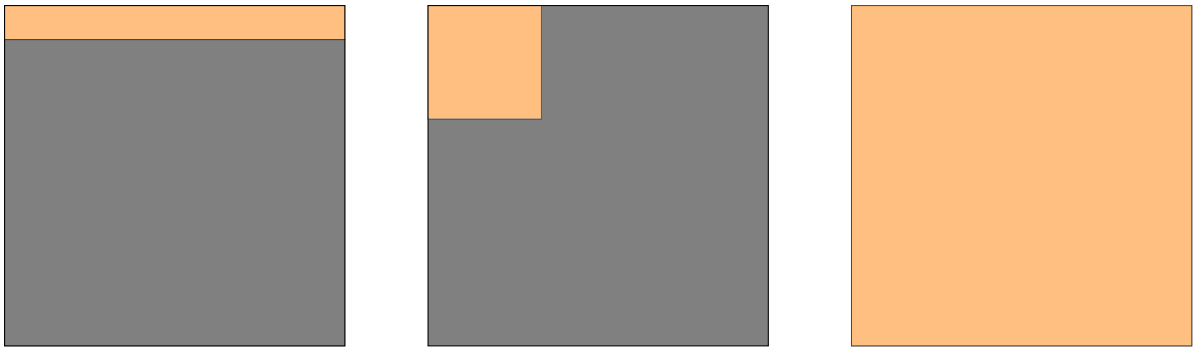


Figure 3 – High-level (simplified) view of the main MDS steps.



(a) Nyström (LMDS) (b) Uniform sampling (c) Importance sampling and linear embedding

Figure 4 – MDS high-level data access pattern (up to a symmetric permutation) to the dissimilarity matrix D (in grey) for the main classes of randomized algorithms. Only orange blocks need to be accessed.

application of randomized algorithms to MDS has been the *sampling* approximation. It was introduced in the MDS ecosystem under the name of Pivot MDS by Brandes and Pich [22] and further developed in the thesis of Klimenta [78]. It consists of performing a partial MDS on a (possibly scaled) principal submatrix. A third class of application of randomized methods to MDS has been the *linear embedding* (which have also been referenced as random projections in the 2000s and 2010s). The most widely known randomized linear embedding algorithm is often referred to as the randomized singular value decomposition (RSVD) [64, 32]. It consists in performing a singular value decompostion (SVD) [17, 75] via probabilistic, fast approach, ensuring the quality of the solution via randomized linear embeddings. Its usage within the MDS (RSVD-MDS) was studied by Blanchard et al. [20, 21, 19] and Paradis [99]. Large datasets could be processed while preserving the numerical robustness [99] of the standard sEVD-MDS.

We refer the reader to [93] for an overview of these three classes (Nyström, sampling and linear embedding) of randomized algorithms in the general context of matrix computation. In the particular context of this thesis, (1) LMDS (thus a Nystrom approximation) will be introduced in Section 3.7.1; (2) MDS based on randomized sampling methods will be discussed in chapters 4 and 5; (3) MDS based on randomized linear embedding will be introduced in Section 1.4. For now, we present only a high-level picture of the methods by their data access pattern to the dissimilarity matrix D. As illustrated in Figure 4a, (1) LMDS only needs to access to a thin block row of D. This block row can be of dimension $(k_{MDS}+1)\times m$ in principle and typically $2k_{MDS}\times m$ in practice. (2) The most basic sampling method blindly selects a random subset of the items. In the MDS case, it translates to access only a principal submatrix of D as illustrated in Figure 4b. Such sampling methods are referred to as *uniform sampling*. However, randomized sampling methods most often discussed in the literature first evaluate the importance of the items before randomly selecting (and possibly scaling) a subset of them depending on their respective importance. Such sampling methods are referred to as *importance sampling*. This preliminary step offers better *a priori* guarantees [93] than uniform sampling. However for evaluating the importance of the items, these methods need to access the whole distance matrix D, as illustrated in Figure 4c. (3) Randomized linear embedding methods also need to access the whole distance matrix D, following the pattern in Figure 4c again. Indeed, a linear embedding of the whole Gram matrix G is performed by multiplying it with a random (typically) Gaussian matrix Ω (see $Y = G\Omega$ matrix multiplication step in Section 1.4.1). Since G must be fully known, so must D.

Apart from randomized algorithms, another class of scalable MDS has also been developed

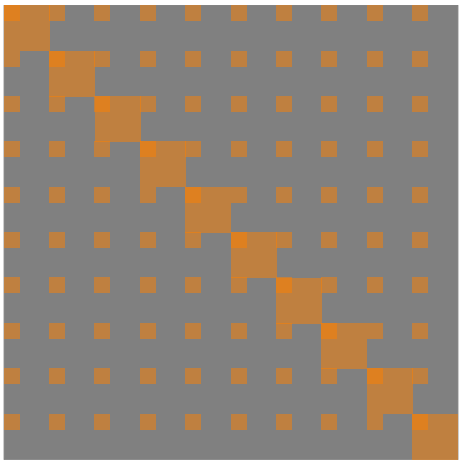


Figure 5 – MDS high-level data access pattern (up to a symmetric permutation) to the dissimilarity matrix D (in grey) for *divide-and-conquer* algorithms (simplified, non hierarchical view). Only orange blocks need to be accessed.

based on a divide-and-conquer paradigm [130, 76]. The idea is to perform independent MDS on overlapping principal submatrices. Figure 5 illustrates the typical data access pattern of such schemes. Extra-diagonal blocks allow diagonal blocks to overlap with each other. The overlap is used to align the respective MDS by Procrustes analysis [117]. The method was introduced under the name of FastMDS by Yang et al. [130]. Ketpreechasawat independently introduced a closely related method under the name of hierarchical landmark charting [76]. Lee and Choi recast them as ensemble learning methods under the name of Landmark MDS Ensemble (LMDSE) [85].

Among these scalable algorithms, both importance sampling and linear embedding still require access to the entire input dissimilarity matrix D, as illustrated in Figure 4c. Processing large scale MDS with these algorithms therefore requires special care in their design. Agullo et al. have recently proposed a high-performance computing (HPC) MDS based on linear embedding [7]. From a numerical point of view, their MDS corresponds to the RSVD-MDS of Blanchard et al. [20, 21, 19]. From an HPC perspective, it consists of a task-based design of the entire RSVD-MDS algorithm, implemented in an efficient software stack including state-of-the-art numerical solvers, runtime systems and communication layers. The outcome is the ability to efficiently apply robust MDS to large datasets on modern supercomputers. This thesis can be seen as a continuation of this work, which we further introduce as a background in Chapter 1 [1].

The first part of this thesis, presented in Chapter 2, revisits the numerical [20, 21, 19, 99] and HPC [7] design of related work on MDS based on linear embedding summarized in Chapter 1. We open this chapter by assessing the impact of the choice of randomized linear embedding technique used for performing the sEVD on the numerical behaviour of the MDS. On the one hand, the standard RSVD used in [20, 21, 19] may (slightly) break the implicitly assumed symmetry of the MDS but requires only a single projection. On the other hand, the standard randomized sEVD (RsEVD) used in [99] preserves the symmetry but requires two projections. An experimental study conducted on a small-scale biodiversity dataset confirms that both approaches may be relevant. This motivates us to propose a HPC design of RsEVD in addition

1. Although I have been involved in this work [7] and that the present thesis builds on top of it, it shall *not* be considered as *a contribution* of the thesis. Indeed, it is the result of much larger project involving five research teams. On the contrary, this thesis has benefited from the dynamics of that project. Therefore, we consider it as a background to the thesis.

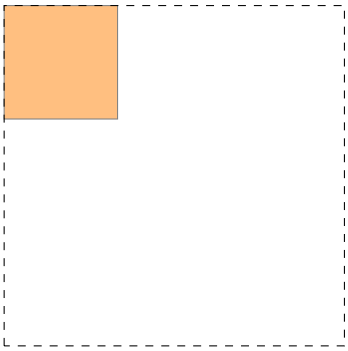


Figure 6 – MDS high-level data access pattern (up to a symmetric permutation) to the dissimilarity matrix D for *uniform sampling without reference sample*. Only orange blocks need to be accessed. There is no reference sample (in grey in the previous pictures).

to the available RSVD algorithm described in background. We are then able to perform HPC MDS using either RSVD and RsEVD variants of the randomized linear embedding algorithm. Both variants are dominated by matrix-matrix multiplications (MM). Their performance and memory consumption are thus keys for that of the overall MDS based on such randomized linear embedding algorithms. We thus incorporate a recently proposed task-based general MM (GEMM) as well as a new task-based symmetric MM (SYMM). We assess the resulting MDS on a large-scale biodiversity dataset corresponding to diatoms from Lake Geneva collected at monthly intervals. We also compare this improved MDS with the starting point of this thesis [7] reviewed in Chapter 1.

The second part of this thesis deals with the comparison of point clouds, either as an end in itself (chapters 3 and 5) or as a means to design a new iterative randomized sampling algorithm (Chapter 4).

Chapter 3 is specifically aimed at comparing point clouds from large scale MDS such as those obtained in the first part of the thesis. The main issue when comparing point clouds from MDS is that they can be arbitrarily translated, rotated, and reflected. It is thus related to the well known problem of Procrustes analysis [117]. The ingredients are thus similar to those of the divide-and-conquer paradigm [130, 76] discussed above, but they are used for the purpose of comparison rather than to build the overall union point cloud. We show that a Procrustes analysis can be performed efficiently with only a few common landmarks, significantly reducing the amount of the input matrix to be computed, consistently with the data access pattern from Figure 5. Once the point clouds are aligned with each other, we also compute their respective distances using Hausdorff distance [70] or variations thereof. The methods discussed are used to compare the diatoms from Lake Geneva collected at different monthly intervals.

Chapter 4 deals with randomized *uniform sampling* to construct a reduced MDS, i.e. a subset of the points in the cloud. As mentioned above, most randomized sampling methods discussed in the literature first assess the importance of the items before randomly selecting (and possibly scaling) a subset of them according to their respective importance [93]. These *importance sampling* methods require access to the entire distance matrix D of the reference sample, as illustrated in Figure 4c. Their strength is that they provide excellent *a priori* error guarantees [93]. These guarantees depend on the spectrum of the data matrix. Although we do not know the spectrum a priori, in practice the first (positive) eigenvalues have a strong decay, as shown in Section 1.8. This theoretical dependence should therefore not be a major problem in practice. A more fundamental problem is that the size of the reference sample with which we are dealing may be overestimated or underestimated in order to obtain an output of the quality we are intending to achieve. Indeed, while this thesis deals only with the processing of

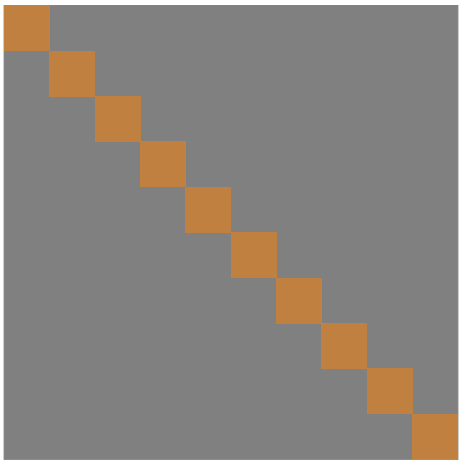


Figure 7 – MDS high-level data access pattern (up to a symmetric permutation) to the dissimilarity matrix D (in grey) studied in Chapter 5. Only orange blocks need to be accessed.

an input dissimilarity matrix D, in practice, the construction of such a matrix for the study of biodiversity is a daunting task (further discussed in Section 1.2). Therefore, approximating the original reference sample with a reduced sample may not be the good question to address. An alternative question is whether we can estimate the intrinsic quality of a sample? Figure 6 is a schematic view of the question: instead of approximating a reference sample as in Figure 4b, the aim is to estimate an unknown whole population (represented by dashes in Figure 6). This latter question is related to the use of *a posteriori* error estimators. Together with iterative schemes, such estimators are advocated by Martinsson and Tropp when sampling lacks precise guarantees [93, Section 9.7]. Chapter 4 explores the use of the distances considered in Chapter 3 as a means of constructing *a posteriori* error estimators for the intrinsic value of a sample. The estimator is computed using a basic *resampling* [72, Chapter 5, 133] technique.

The landmark technique used in chapters 3 and 4 requires the computation of some off-diagonal elements in addition to large diagonal blocks. Chapter 5 investigates whether it is possible to do this without any off-(block-)diagonal elements at all, as illustrated in Figure 7.

Chapter 1

Background

This chapter presents the main background material on top of which this thesis is based. We begin the chapter with an introduction to MDS in Section 1.1, as it is the cornerstone of this thesis. The presentation of MDS is followed in Section 1.2 by a brief introduction to metabarcoding. MDS has many possible applications, and while we present this thesis through the use of metabarcoding data, the methods discussed in this thesis are not limited to the study of biodiversity and are applicable to other areas as well.

MDS relies on an sEVD, which we introduce in Section 1.3, followed by a presentation of the SVD, as it is equivalent to sEVD in the symmetric case, and both have been used in the literature to perform MDS. The main computational step of MDS is the sEVD (or SVD). It has been shown [20, 21, 19, 99] that this step can be performed using randomized linear embedding with either an RSVD or an RsEVD. Following [20, 21, 19, 7], we adopt an RSVD formalism and present such a RSVD-MDS in Section 1.4.

As mentioned in the general introduction, randomized linear embedding in general and RSVD in particular require access to the entire input dissimilarity matrix (see Figure 4c, page 3). Processing large scale RSVD-MDS therefore requires special care in their design. Agullo et al. have recently proposed a high-performance computing (HPC) RSVD-MDS [7], following the numerical design of Blanchard et al. [20, 21, 19]. From an HPC perspective, it consists of a task-based design of the entire RSVD-MDS algorithm, implemented in an efficient software stack including state-of-the-art numerical solvers, runtime systems and communication layers. Section 1.5 introduces task-based programming. Section 1.6 then presents the task-based RSVD-MDS of [7], this thesis can be seen as a continuation of that work. Section 1.7 presents related work on distributed-memory MDS. Section 1.8 presents the datasets used for the experiments of this thesis.

This background chapter is based on the presentation of [7].

1.1 Multidimensional Scaling (MDS)

1.1.1 A brief historical digression

Before presenting the method itself, we propose a brief and partial early historical digression on the original reason behind the term *scaling* in MDS, as it might otherwise be ambiguous in an HPC context. Representing items onto a *linear continuum* (a one-dimensional space in mod-

ern terminology or simply a *scale*) may be loosely related to measure theory, and thus to some extent dates back to Ancient Greece when Archimedes tried to calculate the area of a circle. This idea has been pushed very far in the early twentieth century, in particular by the psychology community who developed advanced methods to design scales onto which positioning judgements (see *e.g.* [123, 124]) or other non trivial concepts. However, if such a traditional *scaling* method (design of scale and positioning of items onto it) was very elaborated in the one dimensional case (representing the information onto a linear continuum only, *i.e.* along one axis), it remained to design a robust process in the multidimensional case (representing the information along multiple well chosen axes).

The term MDS has been ambivalent in the literature. In the present manuscript, it corresponds to "classical MDS", as pioneered by [125, 132]. In 1964, J. B. Kruskal issued a paper [81] with the same name, MDS, but another approach, even if related. The aim is to minimize on X a so-called stress function of the form $\Phi(X) = \sum_{i<j} \left(\|x_i - x_j\| - d_{ij} \right)^2$, where $(d_{ij})_{i,j}$ are dissimilarities, $(x_i)_i$ points in a Euclidean space, and X the set of points. It is not a linear algebra method, but a hard nonlinear optimisation problem. It is nowadays known as Least Square Scaling (LSS), to avoid confusion with classical MDS [29, 90, 71].

This thesis deals only with classical MDS.

1.1.2 Classical MDS

The popularization [42] in the main psychometric journal of the truncated SVD (TSVD) [116], a fundamental mathematical tool, opened the door for the robust extension of the scaling procedures mentioned in Section 1.1.1 to the multidimensional case. Built on top of the TSVD, the *muldimensional scaling* (MDS) method [109, 132] may be viewed as retrieving the most relevant possible multidimensional scale for a prescribed dimension. As it was immediately noted [42], when the matrix is (square and) symmetric (as it is the case in our context), the SVD and the sEVD coincide up to the sign of the eigenvalues. MDS may thus be equally viewed as based on TSVD or truncated sEVD (TsEVD). We follow the sEVD formalism when both approaches are equivalent. However, when randomized algorithms are employed, the approaches can differ. For instance, the RSVD-MDS of Blanchard et al. [20, 21, 19] is not numerically equivalent to the RsEVD-MDS considered by Paradis [99], as it will be detailed in the next chapter (Section 2.3).

There exists many excellent textbooks presenting MDS such as [29] or [71]. A classical and rigorous reference with many results, their demonstration and history is [90]. We refer to this literature for a thorough presentation of the MDS and now only propose a short (but, we hope, progressive) introduction to the method, following [29]. Let $M = (E, d)$ be a discrete metric space of m points endowed with a distance d (in what follows, d can be a dissimilarity too). Let $k_{\text{MDS}} \in \mathbb{N}$ be an integer. MDS [109, 132, 125] aims at building a map x:

$$i \in E \xrightarrow{\quad x \quad} x_i \in \mathbb{R}^{k_{\text{MDS}}} \tag{1.1}$$

such that the norms of the distances $\|x_i - x_j\|_2$ between points x_i and x_j in $\mathbb{R}^{k_{\text{MDS}}}$ are as close as possible to $d(i, j)$. The implicit assumption is that the distances $d(i, j)$ are known (they are the *input* data of the problem) and come from an unknown point cloud $X = (x_i)_i$ (the *output* data to be computed) in a Euclidean space [37]. The objective is to recover X and approximate it as accurately as possible for a prescribed dimension k_{MDS}. In the following, we will denote by $D = (d_{ij})_{i,j}$ the $m \times m$ input pairwise distance matrix. Assuming the Euclidean space is endowed with a given inner product $\langle ., . \rangle$, we furthermore denote by $G = (g_{ij})_{i,j}$ the $m \times m$ matrix of inner products $g_{ij} = \langle x_i, x_j \rangle$, often referred to as the Gram matrix. The solution comes from the

observation that the Gram matrix G can be built from the distance matrix D. Indeed, in a Euclidean geometry, G and D are related through the law of cosines by:

$$d_{ij}^2 = g_{ii} + g_{jj} - 2g_{ij}. \tag{1.2}$$

To obtain the g_{ij} coefficients of G from the d_{ij} coefficients of D, it remains to express the diagonal coefficients $(g_{ii})_i$ (and thus $(g_{jj})_j$) in terms of coefficients of D. To do so, we fist remark that two isometric point clouds cannot be discriminated through the distance between their points, inducing that X can be recovered up to an isometry only. In particular, without loss of generality, we may thus assume that the point cloud X is centered [1], which imposes:

$$\sum_j g_{ij} = \langle x_i, \sum_j x_j \rangle = 0, \tag{1.3}$$

i.e., the sum of each row of the Gram matrix is zero. Noting $d_{i+}^2 = \sum_j d_{ij}^2$, $d_{+j}^2 = \sum_i d_{ij}^2$ and $d_{++}^2 = \sum_{ij} d_{ij}^2$, we may then sum (1.2) over j. We obtain that $\sum_j d_{ij}^2 = \sum_j g_{ii} + \sum_j g_{jj} - 2\sum_j g_{ij}$, By definition of d_{i+}^2, by (1.3), and reminding that $\sum_j g_{jj}$ is the trace of matrix G (*i.e.* $\mathrm{Tr}(G) = \sum_j g_{jj}$), it translates into $d_{i+}^2 = ng_{ii} + \mathrm{Tr}(G)$, yielding $g_{ii} = \frac{1}{n}(d_{i+}^2 - \mathrm{Tr}(G))$. To obtain an explicit expression for g_{ii}, it remains to obtain an expression of the trace of G, which can be done by summing (1.2) over both i and j, yielding $d_{++}^2 = 2n\mathrm{Tr}(G)$. Finally, the Gram matrix G may therefore be written from the distance matrix D as follows:

$$g_{ij} = -\frac{1}{2}\left(d_{ij}^2 - \frac{1}{n}d_{i+}^2 - \frac{1}{n}d_{+j}^2 + \frac{1}{n^2}d_{++}^2\right). \tag{1.4}$$

This computation of G out of D through (1.4) is the first step of MDS (line 1 in Algorithm 1, referred to as `GRAM` in the rest of the thesis). It remains to deduce $X \in \mathbb{R}^{m \times k_{\mathrm{MDS}}}$ from G. First, we remind that, by its above definition, $G = XX^{\mathrm{T}}$. Second, as G is symmetric, it admits a unitary diagonalization so that its sEVD may be written $G = Q\Lambda Q^T$ where Λ is diagonal and Q is unitary. In the case G is semi-definite positive, Λ has only non-negative diagonal values and we may thus write $X = Q\Lambda^{1/2}$. Otherwise, a common heuristic (out of LSS) consists in keeping only the k^+ non-negative eigenvalues Λ^+ and associated eigenvectors Q^+ and considering $X \simeq Q^+\Lambda^{+1/2}$. This is in practice required to relax the assumption that D is a Euclidean distance matrix (and furthermore support the case where it is only a dissimilarity matrix, *i.e.* the triangular inequality does not need to hold). In the case we want to consider the MDS using the SVD of G instead of its sEVD, one may notice that the sEVD of G may also be expressed as $G = U|\Lambda|\mathrm{sign}(\Lambda)V^T$, where $|\Lambda|$ is the diagonal matrix whose diagonal values are the absolute values of the eigenvalues and $\mathrm{sign}(\Lambda)$ is the diagonal matrix whose diagonal values are the signs of the eigenvalues. If we note $U = Q, \Sigma = |\Lambda|$ and $V = \mathrm{sign}(\Lambda)Q$, we observe that we also have $G = U\Sigma V^T$, which is the SVD of G (see below in section 1.3.1). In this thesis, we will consider randomized variants of both sEVD and SVD. For now, we name this step `sEVD` (line 2). Because of the cost of computing a full decomposition, as discussed in the general introduction, various fast alternatives have been considered. In this background section, we will present how this step can be performed using RSVD in Section 1.4.2; we will consider the RsEVD variant in the next chapter (Section 2.2). We omit for now the details of the `CHECK` step (line 3) and will come back to it later. Once the decomposition is obtained, following the above common filtering heuristic, eigenvectors associated with negative eigenvalues are ignored and clipped off from Q (line 4, `Compute_X (1/2)`). Hence, we have $G \simeq Q^+\Lambda^+Q^{+\mathrm{T}}$, with $Q^+ \in \mathbb{R}^{m \times k^+}$ and $\Lambda^+ \in \mathbb{R}^{k^+ \times k^+}$ if k^+ eigenvalues of G are non negative. X can thus be computed as $X \simeq Q^+\Lambda^{+1/2}$ (line 5, `Compute_X`

1. We will discuss this assumption again in Chapter 3.

(2/2)). Finally, the solution of MDS is provided by returning the information associated with the part $X_{k_{\text{MDS}}}$ of the map X associated with the k_{MDS} (assuming $k_{\text{MDS}} \leq k^+$, which is the case in practice as a reasonably small k_{MDS} is most often targeted) largest eigenvalues (line 6).

Algorithm 1: MDS: $(X, \Lambda) = \text{MDS}(D, k_{\text{MDS}})$

Input: a distance matrix $D \in \mathbb{R}^{m \times m}$; a dimension $k_{\text{MDS}} \leq m$ // READ_D
Output: a set of points $X_{k_{\text{MDS}}}$, $\Lambda_{k_{\text{MDS}}}$ the largest positive eigenvalues // WRITE_X

1 Compute the Gram matrix of D: $G = \text{GRAM}(D)$ // GRAM
2 Compute the sEVD of G: $Q\Lambda Q^\intercal = G$ // sEVD
3 Check $\tau = \frac{\|\Lambda\|_F}{\|G\|_F} \overset{?}{>} \tau_{\min} = 1.0 - \epsilon$ // CHECK
4 Compute Q^+ and Λ^+ by discarding in Q all columns q such that $\lambda_q < 0$ // Compute_X (1/2)
5 Compute $X = Q^+ \Lambda^{+1/2}$ // Compute_X (2/2)
6 Keep in $X_{k_{\text{MDS}}}$ (resp. $\Lambda_{k_{\text{MDS}}}$) the k_{MDS} columns of X (resp. Λ) associated with the largest eigenvalues
7 **return** $X_{k_{\text{MDS}}}, \Lambda_{k_{\text{MDS}}}$

The map X we obtain from the MDS will often be called a point cloud. Figure 1.1 represents the heatmap (a representation of point density rather than individual points, the brighter the more points are present) projected alongside the first two axes. Because of the size of the datasets we consider in this thesis, we will prefer heatmaps to represent point clouds. The particular dataset presented here for instance comes from a distance matrix of size $30,000 \times 30,000$. This representation does not allow one to fully represent the information present in X, as projecting along the first 2 axes is equivalent to only considering the fist two columns of X. Indeed, it is equivalent to computing the MDS with $k_{\text{MDS}} = 2$. It is also possible to plot more information, for instance choosing to plot the 2^{nd} and 3^{rd} axes instead of the 1^{st} and 2^{nd}, although the representation using the first two axes is the most natural as they are always the two axes containing the most information (as the axes associated with the two largest eigenvalues). Another option would be to display the point clouds in 3 dimensions (3D). Figure 1.2 for instance presents the same point cloud as the one in Figure 1.1 using a 3D representation. We observe that adding a third dimension reveals additional information about the point cloud. However, this is still not all the information we have on this cloud, as we will usually compute the MDS with a k_{MDS} in the order of $1,000$ and it is already too complicated to plot (and possibly even more complicated to understand) a 4D point cloud. Section 1.8 will present classical scatter plots in higher dimensions. Chapter 3 will discuss in more detail about comparing point clouds by a distance between them.

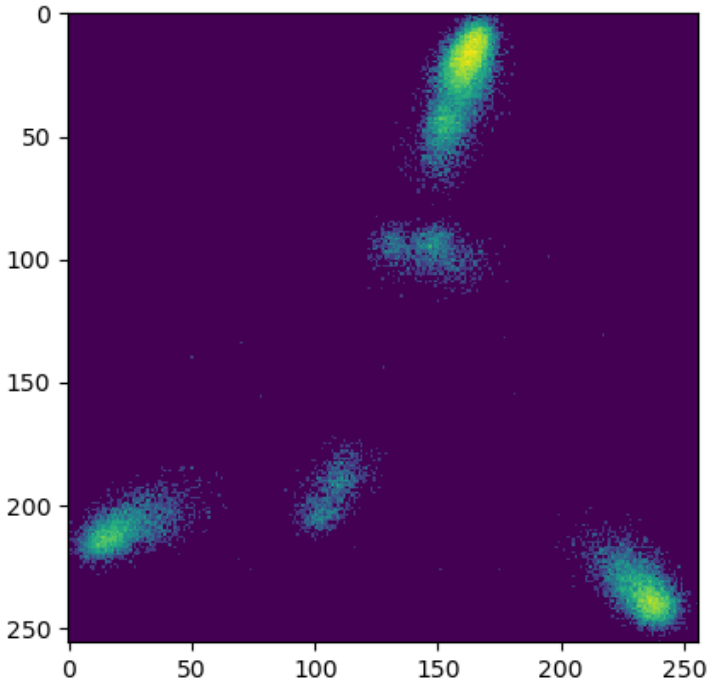


Figure 1.1 – Heatmap of a point cloud generated by MDS, represented using the two axes associated with the two largest eigenvalues. The values on the axes are related to the binning used to create the heatmap. Here the number of bins is 256 per axis.

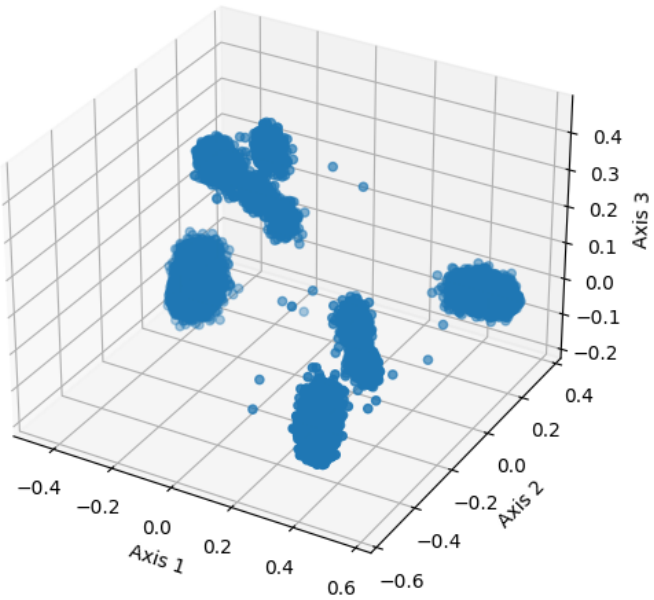


Figure 1.2 – 3D representation of a point cloud using the first 3 axes.

1.2 Metabarcoding

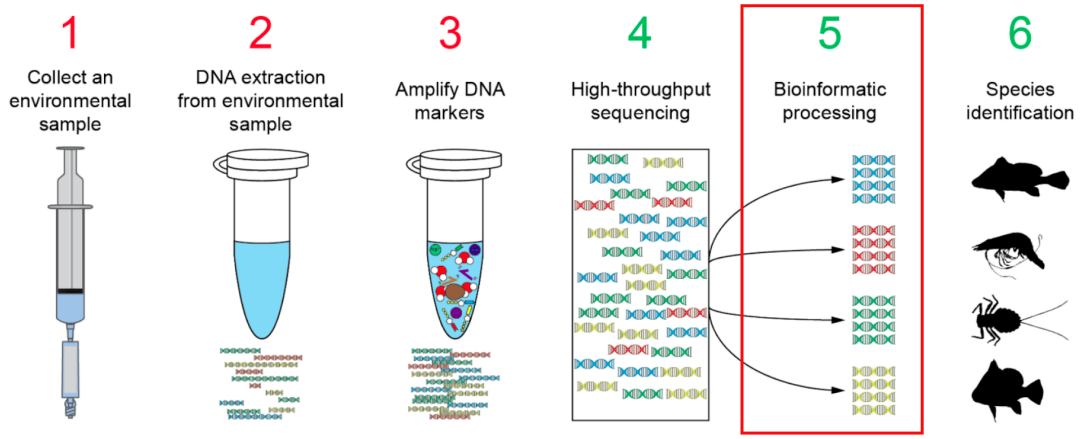


Figure 1.3 – Metabarcoding : a method of species identification based on DNA. In this thesis we only deal with step 5.

The original motivation for this thesis and all the numerical experiments conducted therein are based on biodiversity data. We now present a brief overview of the underlying technique.

Metabarcoding [107, 128, 20] is a branch of molecular biology that focuses on the identification of operational taxonomic units (OTUs, a group of individuals that are closely related at the DNA level) from samples containing large amounts of genetic material obtained by high throughput DNA sequencing. It is an extension of DNA barcoding [46, 66, 127] which involves identifying species on the basis of specific parts of their DNA, to the large amount of data produced by the latest generation of sequencers. Metabarcoding (and DNA barcoding) allows for the study of biodiversity when it is not possible to identify individuals directly or when the number of individuals is too large to be classified using other methods.

The metabarcoding workflow, illustrated in Figure 1.3 consists in the extraction of a specific portion of DNA to be studied. MDS on the distance matrix [20] is one way of studying these samples. In this case, the distance matrix used by MDS is calculated using the Smith-Waterman distance [120] between the DNA sequences. Point clouds resulting from the MDS on such samples can then be used for studying biodiversity.

This thesis only deals with step 5 of the metabarcoding workflow from Figure 1.3. Conversely, we recall that MDS has many possible applications, the methods discussed are applicable to other areas as well.

1.3 Symmetric Eigenvalue Decomposition and Singular Value Decomposition

The main computation step of MDS is the computation of the sEVD of G (line 2 in Algorithm 1). As discussed in Section 1.1.2, the sEVD and SVD are equivalent in this case. In this section we will present both of these decompositions.

1.3.1 Singular Value Decomposition (SVD)

The SVD [17, 75] of a real rectangular matrix $A \in \mathbb{R}^{m\times n}$ consists of the factorization of the form $A = U\Sigma V^T$ where $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal rectangular matrix whose diagonal entries $\sigma_i = \Sigma_{i,i}$ are non negative, ranged in decreasing order, and referred to as the singular values and $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices representing the so-called left and right singular vectors, respectively. In practice, the SVD is stored in a compact form. Assuming A is of rank $r \leq \min(m, n)$, $\Sigma \in \mathbb{R}^{r\times r}$ is stored as a diagonal square matrix and $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{n\times r}$ are stored as rectangular matrices with orthogonal columns. A major breakthrough in the numerical computation of the SVD has been proposed by Golub et al. [52, 57, 53]. We refer the reader to [56] for details and to [122, 68] for an early history. The SVD plays a central role in data analysis because it can been shown that keeping the information related to the first k singular values, referred to as a truncated SVD (TSVD) [116] at rank k, consists of the best approximation in commonly employed norms such as 2-norm among all possible rank k approximations of A.

1.3.2 Symmetric Eigenvalue Decomposition (sEVD)

The sEVD [102, 56, Ch.8] of a real symmetric matrix $A \in \mathbb{R}^{m\times m}$ consists of a factorization of the form $A = Q\Lambda Q^T$ where $\Lambda \in \mathbb{R}^{m\times m}$ is a diagonal matrix whose entries λ_i are the eigenvalues of A and $Q \in \mathbb{R}^{m\times m}$ is orthogonal and representing the eigenvectors of A. While *not* all matrices admit an EVD, all *symmetric* matrices *do* and their sEVD can be written with an orthonormal basis of eigenvectors. We already noted that the sEVD of a symmetric matrix can be related to its SVD in Section 1.1.2. There is also a connection between the SVD of a matrix A as the sEVD of A^TA. Writing $A = U\Sigma V^T$ the SVD of A, we have $A^TA = V\Sigma^T U^T U\Sigma V^T = V\Sigma^2 V^T$ [2]. Note that in this manuscript we will neither tackle nor introduce the unsymmetric eigenvalue problem. The interested reader might refer to chapter 7 of [56] for a detailed description.

1.4 Approximated SVD through randomized linear embedding

It has been shown [20, 21, 19, 99] that we can rely on a much faster algorithm than a deterministic SVD, by using randomized linear embedding [64, 32, 92]. In this algorithm, a deterministic algorithm is still required but with much lower dimensions, such that it is not critical to have a fast distributed-memory deterministic SVD to ensure the efficiency of the overall algorithm. We refer the reader to [38] for a detailed review of HPC (deterministic) SVD developments and we instead focus here on SVD based on randomized linear embedding, or, for short, randomized SVD (RSVD). We will employ it in the sEVD step of our MDS and, for this reason, we will refer to the step associated with line 2 in Algorithm 1 as `RSVD` in the rest of this chapter. Because the randomized linear embedding methods results in approximated factorisation, we assess their quality. This is why the MDS (Algorithm 1) is enriched with a `CHECK` step at line 3. This check consists in measuring the part τ of the information captured by Λ among G through their relative Frobenius norms ($\tau = \frac{\|\Lambda\|_F}{\|G\|_F}$) and verifying that it is large enough ($\tau > \tau_{\min} = 1.0 - \epsilon$, with $\epsilon = 10^{-3}$, being typically satisfying in our case). A more detailed study of the quality of this factorization will be presented in Section 2.3.

2. While this is an intuitive relation between the SVD and sEVD, the computation of the SVD through the sEVD of A^TA is less stable than a direct SVD computation.

1.4.1 Approximating the range of A

In order to perform the RSVD (Section 1.4.2) we need to approximate the range of the input matrix, following the methods presented in [92]. Let us consider $A \in \mathbb{R}^{m \times n}$. The main idea is to approximate the column space of the matrix A by only a small number of vectors through a linear combination of the columns. It requires the generation of random matrix (lines 1 in Algorithm 2, referred to as `RAND` in the following) followed by a matrix multiplication, or MM for short (line 2, referred to as `MM1` in the following). The result is then orthogonalized (line 3, `QR1` and `Q1`) to obtain a basis (Q) of an approximation of the range of A.

The procedure for approximating the range of A consists in considering Ω a random $n \times k$ matrix and performing the matrix-matrix product $Y = A\Omega$ (where Y is then a $m \times k$ matrix). There are several ways to choose Ω [64]. We here restrict ourselves to the Gaussian distribution, *i.e.* Ω is a random Gaussian matrix whose coefficients satisfy the standard normal distribution $\mathcal{N}(0,1)$. Usually, for a good precision at rank k, it is advised to select Ω as $n \times k'$ with $k' = k + s$, where s is called the oversampling. Because taking s as low as $s = 5$ or $s = 10$ is often (and in particular in this study) sufficient [64], we do not distinguish k and k' in the rest of the document. We also assume $k \ll \min(m, n)$, which is also a reasonable practical hypothesis in general and in the particular metabarcoding case study we consider here. In addition, we have $m = n$, as the RSVD is applied on an input square matrix A consisting of the Gram matrix G of the MDS.

1.4.2 Randomized SVD (RSVD)

Let $A \in \mathbb{R}^{m \times n}$ be a matrix with $m \geq n$. The number of floating point operations (flop) of the SVD is $O(mn^2)$ and it becomes unaffordable for large values of m and n. Alternatively, randomized algorithms are very efficient algorithms with bounds on errors, to compute the first singular values and vectors in a reasonable amount of time and memory [64]. One of them is the recently developed RSVD [110, 91]. Once the range of A has been approximated through a basis Q following the steps presented in Section 1.4.1, the matrix A is projected onto this space (line 4, `MM2`) and factorized via a (deterministic) SVD (line 5). After this step, the right singular vectors V are obtained whereas it remains to explicitly form the left singular vectors U (line 6, `MM3`).

Algorithm 2: randomized linear embedding SVD: $(U, \Sigma, V) \simeq \text{RSVD}(A, k)$

Input: A a $m \times n$ matrix, k a prescribed rank
Output: an approximate factorization $A \simeq U\Sigma V^T$

1 Draw a $n \times k$ Gaussian random matrix Ω // RAND
2 Form the $m \times k$ matrix $Y = A\Omega$ // MM1
3 Compute the QR decomposition of Y: $QR = Y$ // QR1, Q1
4 Form the $n \times k$ matrix $C = A^T Q$ // MM2
5 Compute the SVD of C: $U_C \Sigma V_C^T = C$ // QR-SVD
6 Form the matrix $U = QV_C$ and note $V = U_C$ // MM3
7 **return** U, Σ, V

A $m \times k$ matrix with such dimensions $k \ll m$ is often referred to as *tall and skinny* and this shape can be computationally exploited following Algorithm 3 to reduce the complexity of the SVD with respect to the original developments from [52, 57, 53]. The idea is to first compute the QR factorization (line 1 in Algorithm 3, referred to as `QR2` and `Q2` in the following) of the matrix whose SVD is sought. Doing so, it only remains to process a $k \times k$ matrix (R) with the standard (such as [53] or one of its recent derivatives [56]) SVD algorithm (line 2 in Algorithm 3, $k \times k$ `SVD`) before reconstructing the left singular vectors U (line 3, `MM4`). Such a QR-SVD approach was first proposed by Lawson and Hanson [83] and further analyzed by Chan [26]. The orig-

inal motivation was the reduction of the computational complexity. In a distributed-memory context, it is also an opportunity for ensuring a separation of concerns: ensuring only a fast $m \times k$ distributed-memory QR factorization while relying on a sequential (or shared-memory) $k \times k$ standard SVD. We will employ such a QR-SVD algorithm to process the deterministic SVD step of the RSVD algorithm; this is why line 5 in Algorithm 2 will be referred to as `QR-SVD` in the following.

Algorithm 3: QR-SVD algorithm: $(U, \Sigma, V) = \text{QR-SVD(A)}$

Input: A a $n \times k$ matrix
Output: a factorization $A \simeq U\Sigma V^T$

1 Compute the QR decomposition of A: $QR = A$ // QR2, Q2
2 Compute the SVD of the triangular matrix R: $U_R \Sigma V^T = R$ // $k \times k$ SVD
3 Form the matrix $U = QU_R$ // MM4
4 **return** U, Σ, V

We then consider the RSVD-MDS as the MDS algorithm (Algorithm 1) for which the `sEVD` (step 2) is processed with an RSVD (Algorithm 2) for which the input $m \times n$ matrix A is the $m \times m$ Gram G matrix of the MDS, in particular involving that the RSVD is processed on a square matrix ($m = n$). The internal deterministic SVD step within the RSVD (step 5 in Algorithm 2) is itself processed with a QR-SVD (Algorithm 3). We also highlight that, in addition to the prescribed rank k_{MDS} of the baseline MDS algorithm (Algorithm 1), the RSVD-MDS algorithm is also parameterized with the dimension k of the randomization and must satisfy $k_{MDS} \leq k^+ \leq k$. The dimensions m and k drive the computational load of the RSVD and RSVD-MDS algorithms. On the other hand, the k_{MDS} parameter is only used in the ultimate step of the MDS, essentially depends on what the MDS is used for (for instance, if the goal is to obtain a point cloud visualization, it corresponds to $k_{MDS} = 2$), and does not significantly impact the computational load. As a consequence, the performance study will be parameterized with m and k only.

It is to be noted that the RSVD may (slightly) break the symmetry when forming the approximate matrix QQ^TA. However, in our context, the QQ^TA approximation is likely to be an excellent approximation of A so that the symmetry is almost preserved and does not significantly impact the numerical result in most cases. We will discuss it in greater details in Section 2.3. Alternatively, a symmetry-preserving randomization technique [64, 91] might be employed, we present it the Section 2.2.

1.5 Task-based programming

Recent supercomputers are increasingly complex, composed of many multicore nodes as well as accelerators such as GPU. Writing numerical code for scientific simulation that can take advantage of the full potential of these machines is becoming increasingly complicated. Writing these codes using relatively low-level primitives may in theory allow for a more optimized code that can exploit the full potential of the hardware, but requires to develop and maintain complex codes combining multiple levels of parallel expressions such as message passing for inter-node communications, multithreading for exploiting multicore chips and vendor primitives such as `cuda` for exploiting GPUs. The other approach consists in delegating this work to a runtime system and let it handle the scheduling of computation and communication between nodes. However, using these runtime systems while adding a level of abstraction that can greatly simplify the writing of code comes with other challenges, an important one being to deliver the same performance as libraries based on lower level parallel scheme. In Chapter 2 we have chosen to rely on the sequential task flow (STF) model [4, 5, 6], where tasks are created

sequentially and their mutual dependencies are automatically inferred through an analysis of their data access mode (`READ (R)`, `WRITE (W)` or `READ-WRITE (RW)`). This model is, for example, available in OpenMP (through the `task` directive and the `depend` clause), OmpSs [41] or StarPU [10], the runtime system we use in this work. The tasks are then represented using a directed acyclic graph (DAG) where vertices are tasks and edges dependencies between them. As a consequence, the runtime system has a complete view of the application, and can schedule the task and communication appropriately, up to the point of being able to pipeline multiple operations without the need for synchronization steps. The effectiveness of this model on shared memory parallel computers has been proved by numerous works although it has been much more rarely considered in distributed-memory settings; early work on this topic is presented by [131] and [4]. More recently, an extension of the STF model was proposed by [5] that allows for portable implementations of scalable algorithms for distributed-memory computers.

The STF programming model can be used in a distributed-memory setting to manage the distribution of the task over the nodes of a cluster, see [4]. In practice, the application instructs the runtime about the distribution of data to be used over the nodes of the cluster, typically a 2D block-cyclic (2DBC) distribution, see Section 2.5. Unlike widely used dense linear algebra libraries, for which the data mapping is often hardwired in the code making the implementation of alternative data distribution tedious, this model allows for delegating this task to the runtime making the implementation of new and potentially irregular data mapping much more accessible. On each node, the runtime system automatically determines which tasks of the graph should be performed by the node: if and only if it owns the data that is written to by the task. This is deterministic, and thus all nodes automatically agree on which of them will perform each task, without having to exchange any message. The runtime is also in charge of handling communication of the data between the nodes that needs it, relying in many cases on MPI [47]. Special care has to be taken to mix communication and multithreading [35] or to take advantage of using both [126].

Writing a task-based code in the STF model is twofold. First, the data structures must be designed and declared to the runtime system, we will refer data declared to the runtime to as *data handles*. Second, the sequence of tasks operating on those data handles must be written using an `insert_task` function. The submitted task is an elementary operation accessing data with specific modes. A runtime can use the provided access modes (e.g. `Read`, `Write`) to order tasks and control concurrent execution.

We illustrate it with a general matrix matrix multiplication (GEMM). The sequential code of a matrix multiplication is commonly written as three nested loops as in Algorithm 4. This algorithm can be translated into an STF version as simply as shown in Algorithm 5. Such a task-based implementation can then run on any platform as long as the elementary operations (here the GEMM tasks inserted in line 4) are provided for the target platform. For instance, on an heterogeneous machine composed of both CPUs and GPUs, if the GEMM implementation for a CPU core and a GPU are provided, the runtime system can then dynamically decide on which unit perform each individual GEMM task. In addition, it automatically handles communications in the case of a distributed-memory platform. This versatility will be used in the next chapter to design advanced versions of the symmetric matrix multiplication (SYMM), the dominant kernel in both RSVD and RsEVD algorithms and corresponding RSVD-MDS and RsEVD-MDS respective methods.

1.6 Task-based randomized linear embedding MDS via RSVD

Algorithm 4: Sequential block GEMM.

```
1 for j = 1...N do
2     for i = 1...M do
3         for l = 1...M do
4             gemm(A_{i,l}, B_{l,j}, C_{i,j});
```

Algorithm 5: STF block GEMM.

```
1 for j = 1...N do
2     for i = 1...M do
3         for l = 1...M do
4             insert_task(gemm, A_{i,l}:R, B_{l,j}:R, C_{i,j}:RW);
```

1.6.1 Task-based RSVD

The RSVD as presented in 1.4.2 was implemented in [7] in the `fmr` library, which was used in the context of a larger software stack presented in Figure 1.4

The input matrix A of the RSVD (Algorithm 3) is the Gram matrix G of the MDS. The code in `fmr` and `mds` used to perform the RSVD and MDS respectively can rely on either BLAS/LAPACK or `chameleon` to perform the basic matrix operations needed for the MDS, namely BLAS/LAPACK for shared memory and `chameleon` in the distributed-memory case.

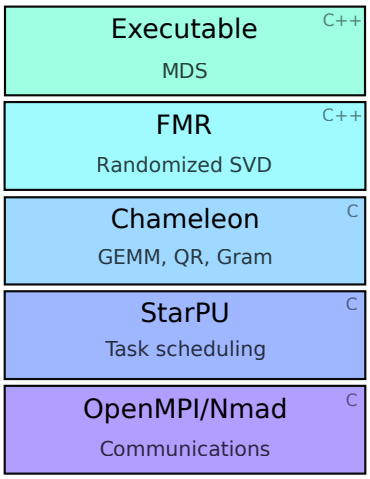


Figure 1.4 – Software stack used in the HPC computation of RSVD-MDS.

The `QR-SVD` step (Algorithm 3, called at line 5 in Algorithm 2) is implemented within `fmr` as follows. The QR decomposition (line 1 in Algorithm 3) is performed through two calls: (`QR2`) a call to `chameleon pdgeqrf` for performing the QR factorization followed by (`Q2`) a call to the `chameleon pdorgqr` for explicitly forming matrix Q. The $k \times k$ R obtained being of small dimension, it is centralized and processed with (a deterministic) LAPACK `dgesvd` SVD call ($k \times k$ `SVD`, line 2). Q and U_R being formed at this stage, their final product $U = QU_R$ (`MM4`, line 3) is once again simply performed by a standard general matrix multiplication through a `chameleon pdgemm` call.

1.6.2 Implementation of MDS using RSVD

Now, we present the task-based design of [7] within the `mds` application of the overall RSVD-MDS (Algorithm 1) discussed in 1.1.2. The input distance matrix D of the MDS is assumed to be stored in `hdf5` format. Following a tile algorithm design, the authors' task-based RSVD aims at reading it from disk and arranging it into a distributed-memory tile matrix (`READ_D` in Algorithm 1). Here the matrix is distributed according to a 1D block-cyclic pattern (1DBC) as opposed to a more classical 2D block-cyclic pattern (2DBC) as the size of the matrix D compared to the tall skinny matrices makes the 2DBC less efficient.

The first numerical step of the MDS is the computation of the Gram matrix G from the distance matrix (`GRAM`, line 1 in Algorithm 1). From a data access point of view, `GRAM` is a reduction-like algorithm and is mainly communication-bound. A per-column reduction (the per-row reduction is avoided by symmetry) is performed to compute all the d^2_{+i} (and d^2_{i+} by symmetry) while a complete reduction is performed to compute d^2_{++}. This process is illustrated in Figure 1.5. In the context of the Gram matrix computation, note that after the distributed row reduction (step 3 in Figure 1.5), all the d^2_{i+} values are computed in small vectors of the same size as the tiles. The reduction is then pursued up to the distributed column reduction (step 6 in Figure 1.5) in order to obtain the d^2_{++} value. Finally, an embarrassingly parallel update of each tile of the matrix is submitted. Each task computes the final g_{ij} values with the respective vectors from the Step 3 and the final value from the Step 6.

The second numerical step (`RSVD`, line 2 in Algorithm 1) is the SVD, effectively computed with an RSVD in the considered RSVD-MDS algorithm. The `mds` application has been modified to call the `fmr` task-based RSVD routine presented earlier in 1.4.2. Σ being diagonal, its Frobenius norm is immediate to compute and the third step (`CHECK`, line 3 in Algorithm 1) therefore essentially consists in computing the Frobenius norm of G. The last two steps of the MDS consist in computing matrix X (`Compute_X`) through $X = U^+\Sigma^{+1/2}$ (line 5 in Algorithm 1) based on the filtered (line 4 in Algorithm 1) output (U^+, Σ^+) of the (R)SVD and scaling each column i of U^+ to compute $X \simeq U^+\Sigma^{+1/2}$.

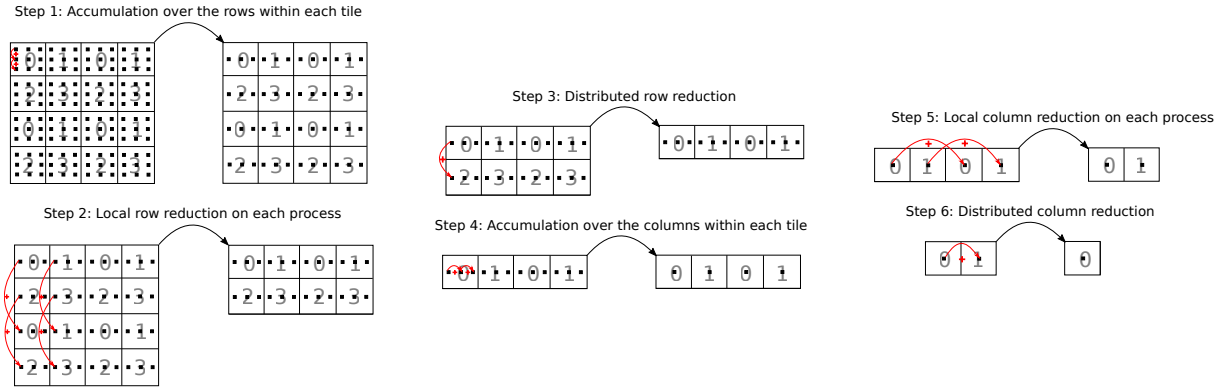


Figure 1.5 – Task-based `GRAM` tile algorithm.

1.6.3 Complexity and key performance steps of the RSVD-MDS

The flop count of the RSVD (Algorithm 2) is dominated by the `MM1` and `MM2` matrix multiplications. The RSVD satisfies $\text{RSVD}(m,n,k) \sim_{m,n\to\infty} 4mnk$. In particular, we have $\text{RSVD}(m,n,k) = \mathcal{O}(mn)$ as $m,n \to \infty$. The RSVD step (called with $n = m$) dominating the RSVD-MDS (Algorithm 1), the flop count of the latter satisfies $\text{RSVD-MDS}(m,k) = \mathcal{O}(m^2)$ as $m \to \infty$. In a nutshell, the `MM1` and `MM2` general matrix multiplication steps are the dominant numerical operations of both the RSVD and the overall RSVD-MDS algorithms, when considering practical dimensions

($k \ll m$).

Both QR factorization (`QR1` and `QR2`) and subsequent orthogonal generation (`Q1` and `Q2`) steps have a lower computational complexity than the dominant matrix multiplication step `MM1`. However, their tall and skinny shape had long made them challenging to process efficiently in parallel. A new numerical scheme [33, 34] has been proposed about fifteen years ago to reduce the number of communications when processing such matrices. It has then been demonstrated [63] that the original tile QR factorization [24] can cope with this scheme.

The other numerical steps do not represent as high challenges and we therefore do not discuss them further. It remains however to explain how to arrange the sequence of calls in a distributed framework, which is often a hard challenge to make without synchronization. In [7] the authors rely on the STF programming model explained in 1.5. It allows for writing a sequential task-based algorithm and let the runtime system infer (and handle) the dependencies between them. As a consequence, no synchronization is needed between each above discussed step. As a consequence, the whole RSVD and RSVD-MDS algorithms are fully pipelined up to the actual numerical dependencies of the tasks.

1.7 Related work on distributed-memory MDS

To the best of our knowledge, no distributed-memory classical MDS had been proposed before [7]. However, multiple propositions have been discussed for designing distributed-memory LSS [135, 11, 104, 28, 12]. They all rely on a full MPI [47] parallelization scheme. Zilinskas and Zilinskas designed the parallelization of a genetic LSS algorithm, assessed on up to 24 cores [135]. Pawlizeck and Dzwinel considered a heuristic based on particle dynamics simulation to find the minimum of the stress function. They validated their approach up to 144 cores [104]. The last three LSS parallelization schemes are based on a non trivial variant of the gradient descent algorithm. The method is named Scaling by MAjorizing a COmplicated Function (SMACOF) [86]. At each iteration of the SMACOF algorithm, the dominant part is a matrix-matrix product to update the points. The parallel design proposed by Bae was initially assessed up to 8 cores [11], and extended up to 256 cores [12, 28], obtaining an efficiency of 70% for dataset of size $100,000 \times 100,000$.

Regarding the RSVD itself, it had been parallelized for shared-memory [87] and GPU-accelerated [73, 89] single-node machines but, to the best of our knowledge, not for distributed-memory machines before [7].

1.8 About the datasets used in this thesis

In this section, we present the datasets used for the experiments of this thesis. They range from very small $1,502 \times 1,502$ sizes that were mainly used for numerical validation all the way up to a dataset of size $1,043,192$ that we used extensively to assess the scalability of the proposed approaches.

1.8.1 Atlas Guyane

The Atlas Guyane dataset (see Figure 1.6) is a 1502-by-1502 dissimilarity matrix obtained from sequences of tropical trees in French Guiana [49]. We use this dataset as a small-scale dataset, which is convenient to perform quick experiments when doing development. It is also an interesting dataset as the magnitude of the eigenvalues decays slowly (see Figure 1.7) and it

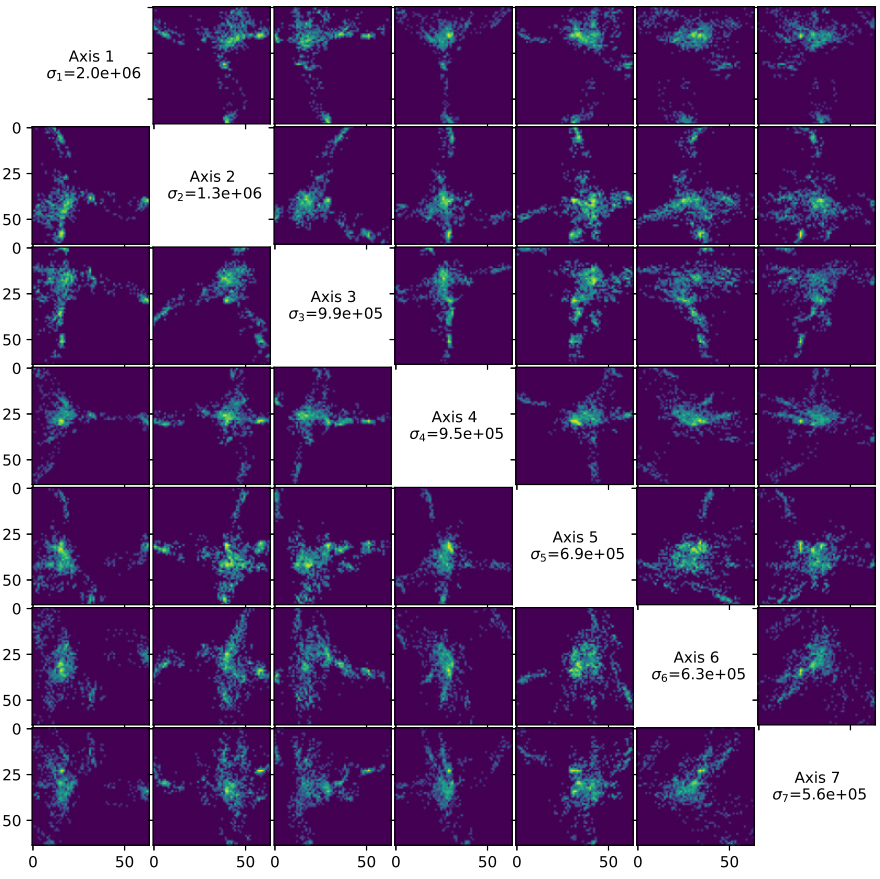


Figure 1.6 – Heatmap of the Atlas guyane point cloud, obtained from a full MDS of the dataset. Each heatmap corresponds to the representation of the two dimensions in the diagonal block. For instance, the third heatmap of the first column represents the first axis along the x-axis and the fourth axis along the y-axis.

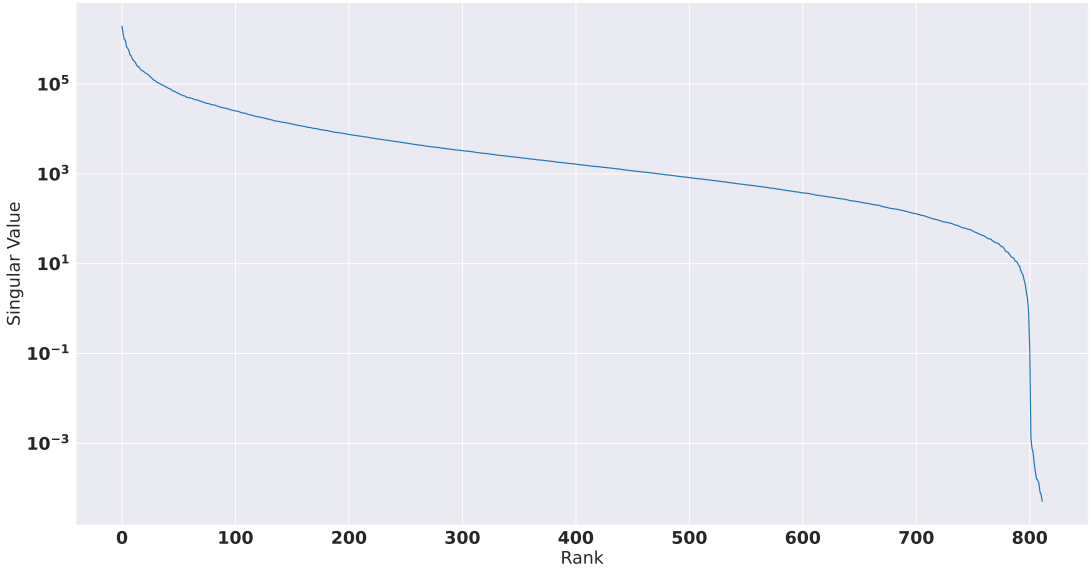


Figure 1.7 – Singular values of the Atlas guyane dataset, obtained from full SVD. Values associated with a negative eigenvalue have been filtered out as they are not part of the axes kept for building the associated point cloud.

may thus be viewed as numerically full-rank. From this perspective, it is very different from the other datasets. For more information related to this dataset, see [25, 1].

1.8.2 Long Reads

The Long Reads samples are a set of four samples of bacteria named Long Reads **A** through **D**, each composed of 30,000 entries. These datasets have been randomly sampled from a larger sample of 400,000 entries. As a consequence, they are supposed to represent the same information. We will further discuss it in Chapter 5 with the aim to assess whether it is possible to confirm this hypothesis without accessing any off-diagonal block at all (having in mind Figure 7 from the general introduction on page 6). Note that we actually do have access to all the off-diagonal, which allows us to compute a reference overall reliable point cloud. This overall dataset is referred to as Long Reads ABCD. Figures 1.8 and 1.10 respectively show the heatmap of Long Reads A and the decay of its positive eigenvalues. Figure 1.9 shows the heatmaps of the four Long Reads samples **A** through **D**.

1.8.3 10V-RbcL (Malabar project)

The dataset 10V-RbcL is from the Malabar project [9]. It contains a number of samples extracted at different places, time of the year and tide. It is a sample of size 23,214. Figures 1.11 and 1.12 respectively show the corresponding heatmap and decay of the positive eigenvalues.

1.8.4 Diatoms from Lake Geneva (S5 dataset split into L1, . . . , L10)

In this section, we present the S5 dataset. It is the larger one assessed in this thesis [48]. It results from the sampling of diatoms from Lake Geneva at ten monthly intervals. The ten samples are named L1, . . . , L10. Together they form the S5 dataset (also referred to as the full L1-L10 dataset). Each sample consists of approximately 100,000 entries, as further detailed in Table 1.1. In total, S5 (i.e. the full L1-L10) is a matrix of size 1,043,192 (726GB). This matrix

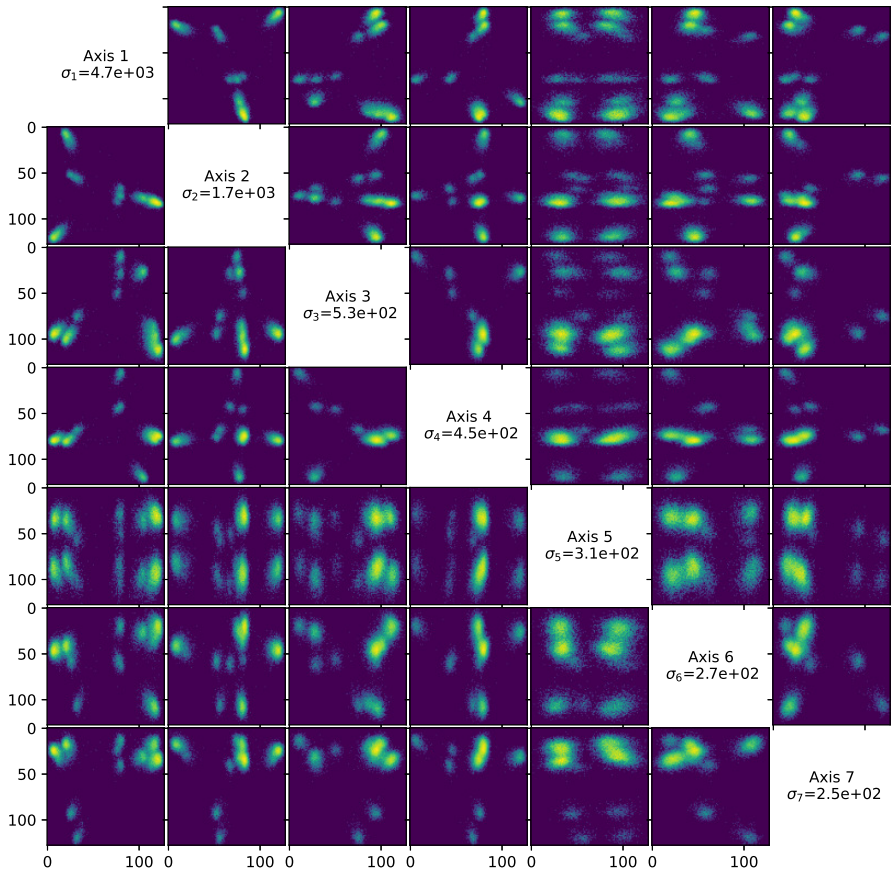


Figure 1.8 – Heatmap of the Long Reads A point cloud, obtained from a RSVD-MDS of the dataset at rank 5000. Each heatmap corresponds to the representation of the two dimensions in the diagonal block. For instance, the third heatmap of the first column represents the first axis along the x-axis and the fourth axis along the y-axis.

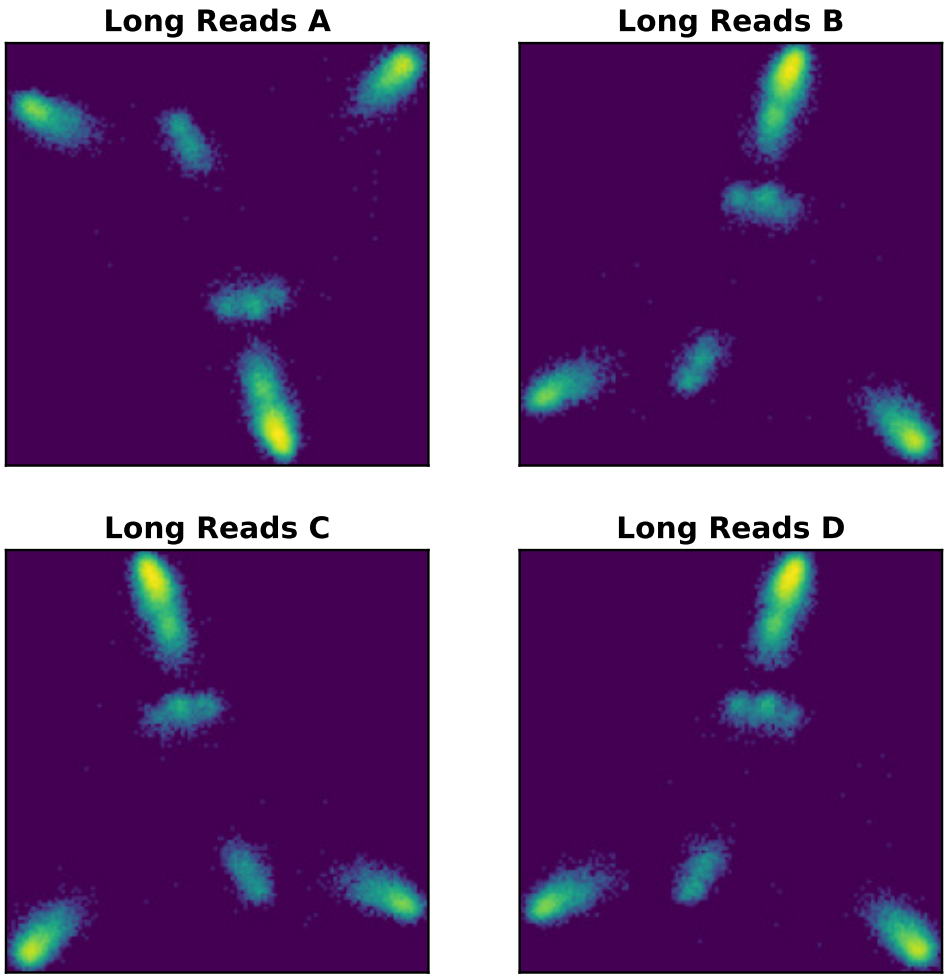


Figure 1.9 – Heatmaps of the four Long Reads Samples, each obtained from a RSVD-MDS of the respective datasets Long Reads A, B, C and D at rank 5000.

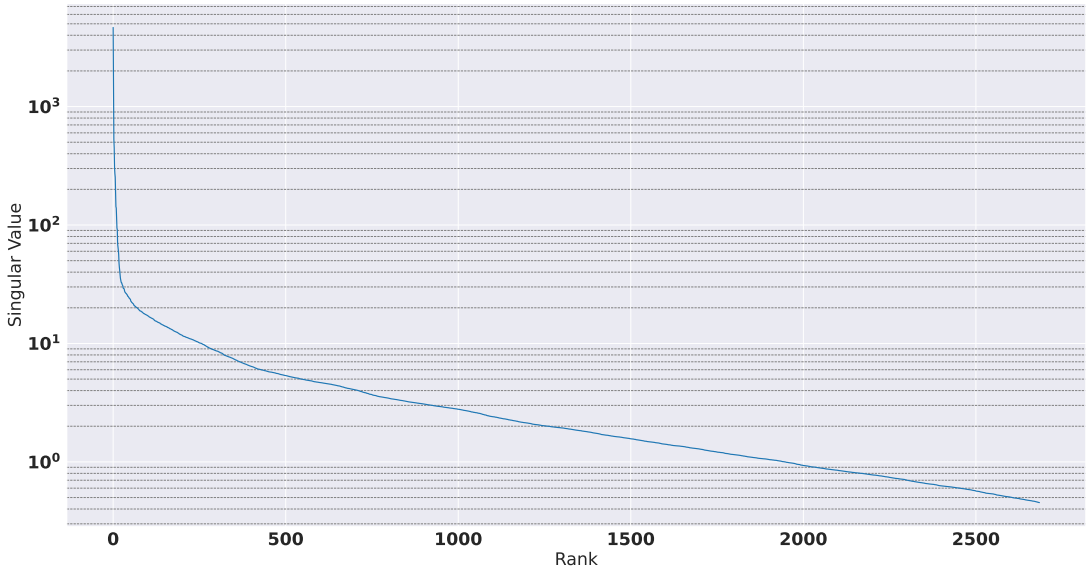


Figure 1.10 – Singular values of the Long Reads A dataset, obtained from RSVD-MDS of rank 5000. Values associated with a negative eigenvalue have been filtered out as they are not part of the axes kept for building the associated point cloud.

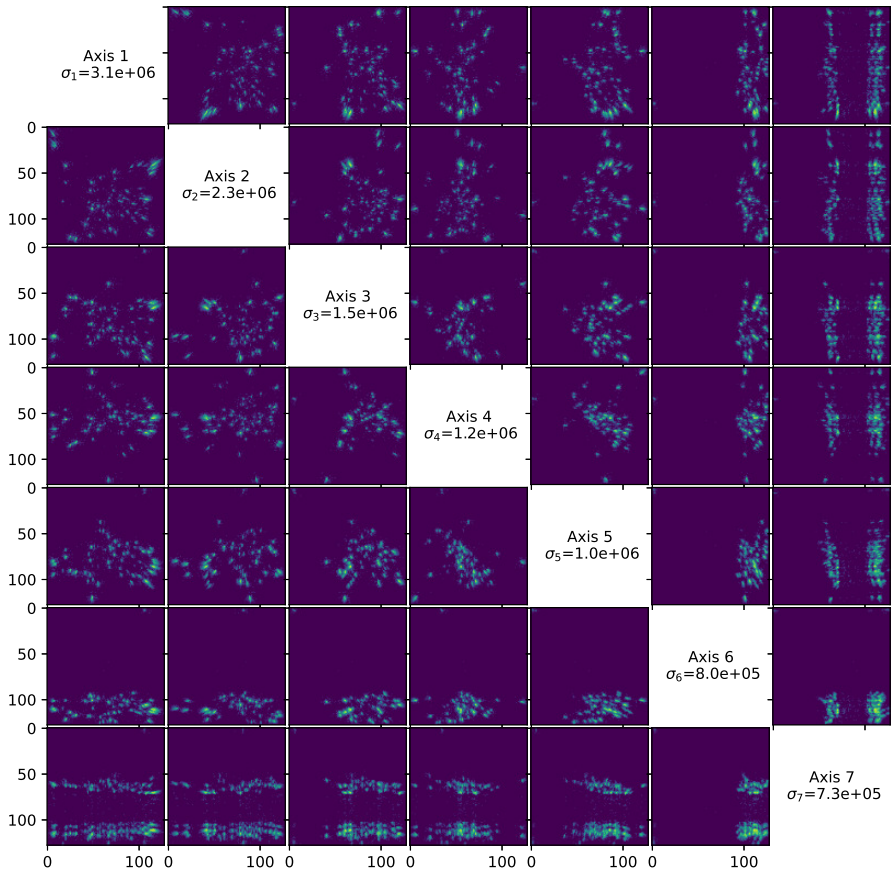


Figure 1.11 – Heatmap of the 10V-RbcL point cloud, obtained from a RSVD-MDS of the dataset at rank 500. Each heatmap corresponds to the representation of the two dimensions in the diagonal block. For instance, the third heatmap of the first column represents the first axis along the x-axis and the fourth axis along the y-axis.

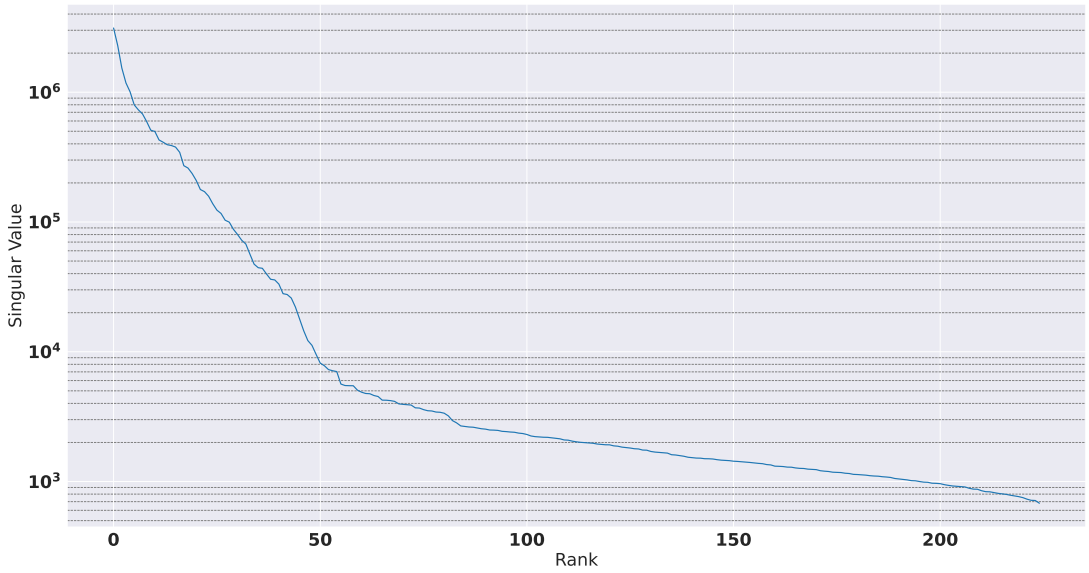


Figure 1.12 – Singular values of the 10V-RbcL dataset, obtained from RSVD-MDS of rank 500. Values associated with a negative eigenvalue have been filtered out as they are not part of the axes kept for building the associated point cloud.

was used in [7] in order to demonstrate the ability of the RSVD-MDS algorithm to scale to very large datasets. This is also the dataset used in [20]. We use it in the thesis to evaluate the scalability of our algorithms. We can consider either a single L_i sample or the union of a subset of L_i samples as shown in Table 1.2. In addition, this dataset is used to evaluate the robustness of the comparison algorithms of Chapter 3. The singular values obtained using RSVD-MDS are shown in Figure 1.14. We also present the heatmaps in Figure 1.13, following the same per-axis representation to show the first few dimensions of the full **S5** point cloud. Finally, Figure 1.15 shows the heatmaps for the first two dimensions of each L_i (obtained separately) as well as that of the full sample S5. These results have been obtained using the RSVD-MDS of [7] and will serve as a reference for the remainder of the thesis.

Table 1.1 – Samples names and matrix size from Lake Geneva dataset.

Sample name	size
L1	72,083
L2	98,492
L3	72,897
L4	136,450
L5	75,218
L6	99,594
L7	124,367
L8	115,607
L9	81,983
L10	166,501

Table 1.2 – Samples names, composition and size.

Sample name	composition	size
S1	*L6*	99,594
S2	*L2-L3-L6*	270,983
S3	*L1-L3-L5-L7-L9*	426,548
S4	*L2-L4-L6-L8-L10*	616,644
S5	*All*	1,043,192

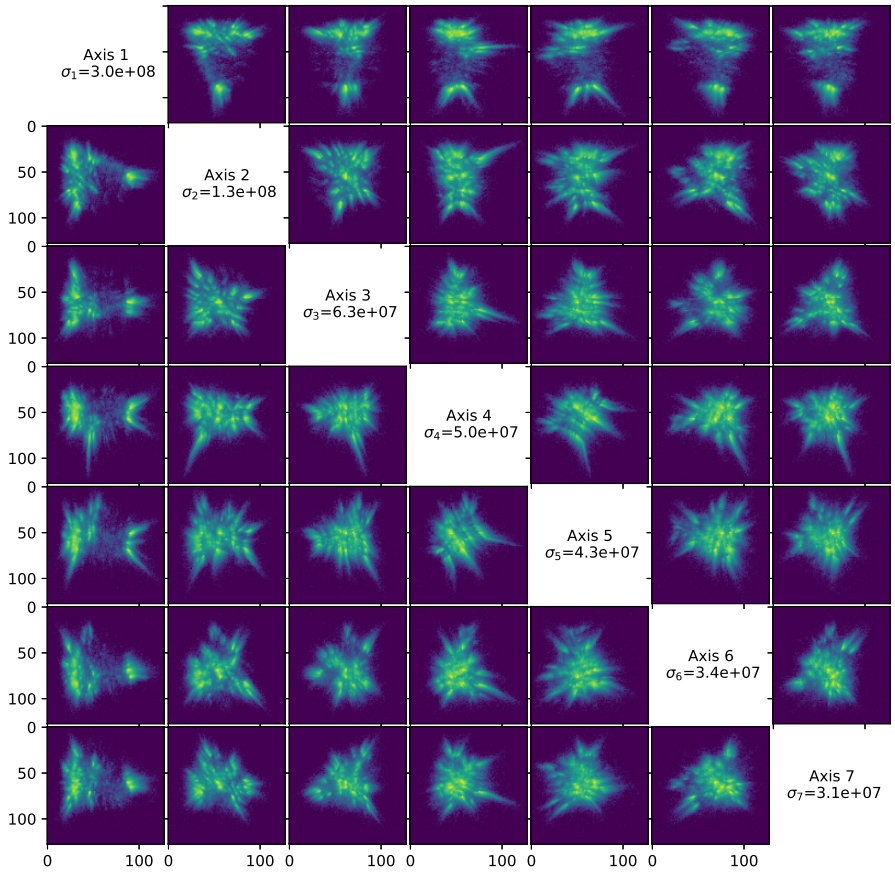


Figure 1.13 – Heatmap of the full L1-L10 point cloud (sample S5), obtained with an RSVD-MDS of the dataset at rank 1000. Each heatmap corresponds to the representation of the two dimensions in the diagonal block. For instance, the third heatmap of the first column represents the first axis along the x-axis and the fourth axis along the y-axis.

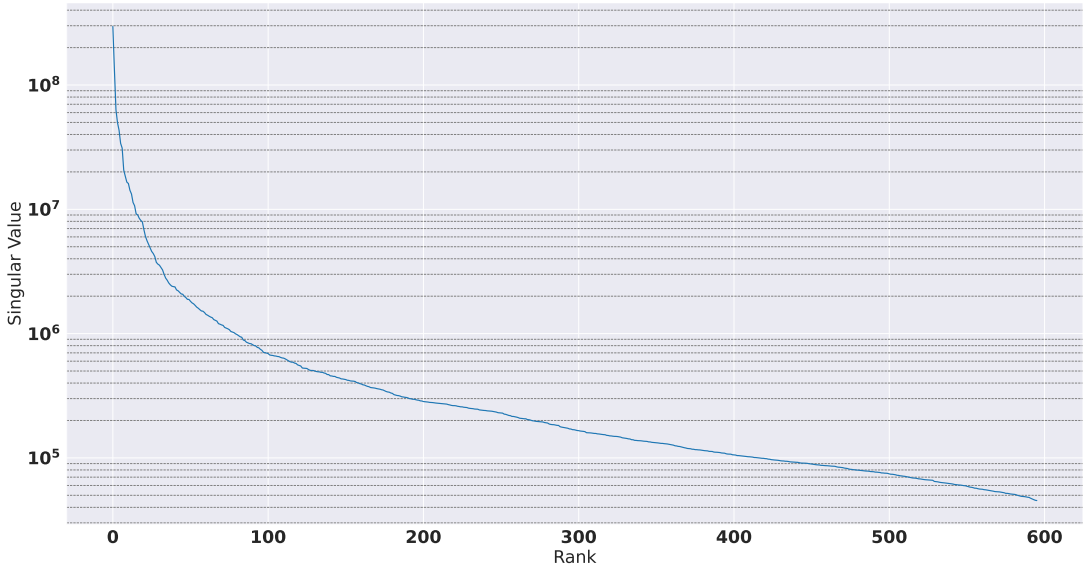


Figure 1.14 – Singular values of the L1-L10 dataset (sample S5), obtained with an RSVD-MDS at rank 1000. Values associated with a negative eigenvalue have been filtered out as they are not part of the axes kept for building the associated point cloud.

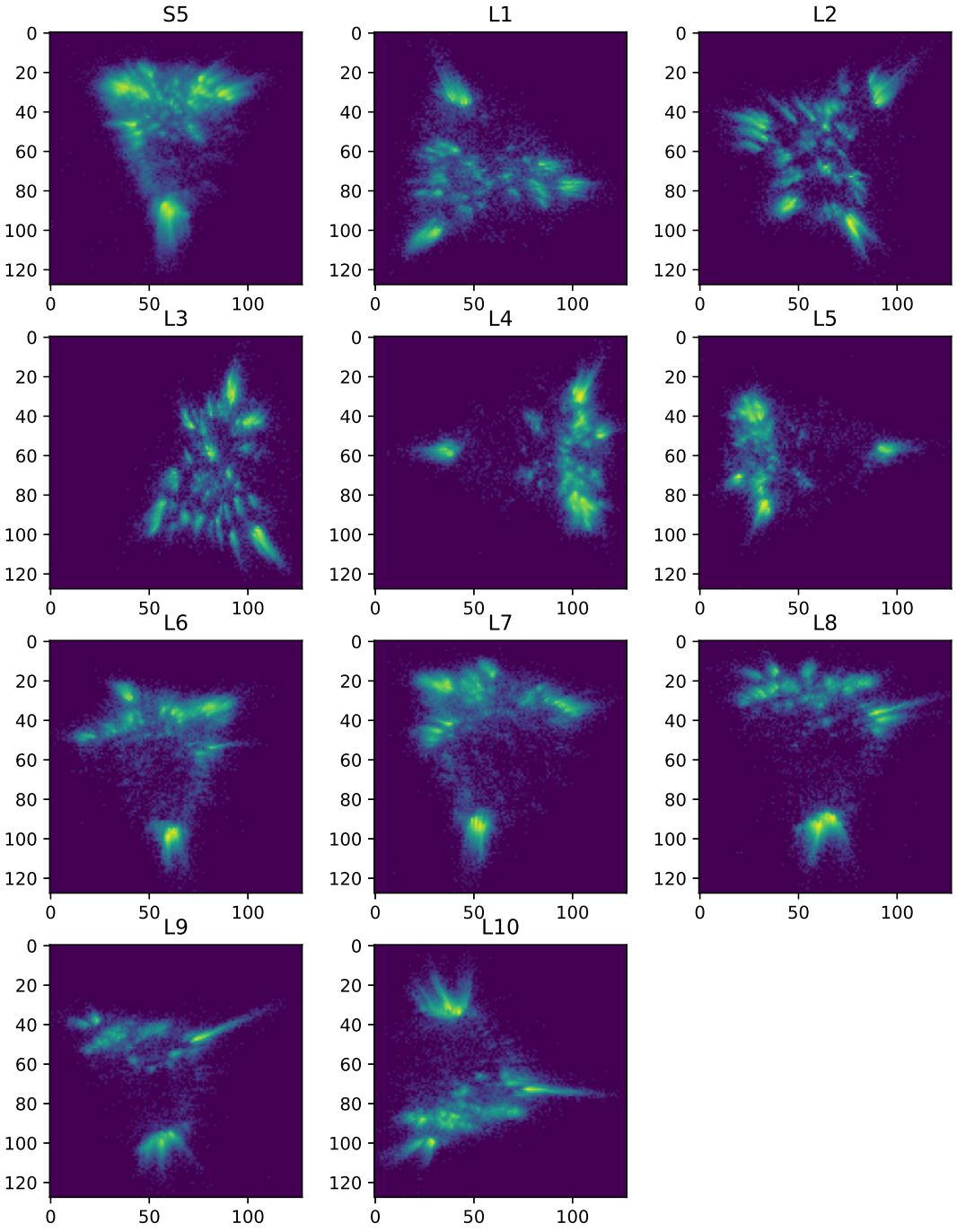


Figure 1.15 – Heatmaps of the full L1-L10 dataset (sample S5) obtained with an RSVD-MDS at rank 1000 on the top left, alongside the heatmaps for every other L_i samples also computed from RSVD-MDS of rank 1000.

Part I

Robust high-performance MDS

Chapter 2

Task-based randomized linear embedding MDS

2.1 Introduction

This chapter revisits the numerical [20, 21, 19, 99] and HPC [7] design of related work on MDS based on linear embedding discussed in Chapter 1. Whereas Chapter 1 presented an RSVD-MDS [20, 21, 19, 99, 7], Section 2.2 introduces an RsEVD-MDS variant. Instead of being computed with an RSVD, the `sEVD` (step 2 in Algorithm 1) is computed through an RsEVD. Section 2.3 discusses the impact of the choice of the randomized linear embedding variant on the numerical behaviour of the MDS. On the one hand, the standard RSVD used in [20, 21, 19] may (slightly) break the implicitly assumed symmetry of the MDS but requires only a single projection. On the other hand, the standard randomized sEVD (RsEVD) used in [99] preserves the symmetry but requires two projections.

Both RSVD-MDS and RsEVD-MDS are dominated by matrix-matrix multiplications (MM). It corresponds to steps `MM1` and `MM2` in both algorithms 2 and 6. The Gram matrix G is symmetric by construction, as it is based on a centered dissimilarity matrix. As such, both `MM1` and `MM2` can be performed using either GEMM or SYMM. GEMM is the general matrix multiplication and does not take into account any specificity of the matrices while SYMM is its counterpart when one of the input matrix is symmetric. In order to maximize performance, both matrix products may be performed with a GEMM; however this implies that the matrix G, initially stored in a symmetric format, has to be converted to full format, thus doubling the initial memory footprint. The performance and memory consumption of MM are thus keys for that of the overall MDS based on such randomized linear embedding algorithms. We motivate the improvement of distributed-memory matrix multiplication in Section 2.4. We review state-of-the-art MM for distributed-memory machines in Section 2.5. We furthermore design a new task-based symmetric MM (SYMM) in Section 2.6. We assess the resulting MDS on the large-scale biodiversity dataset corresponding to diatoms from Lake Geneva in Section 2.8. We also compare this improved MDS with the starting point of this thesis [7] reviewed in Chapter 1.

2.2 Randomized sEVD (RsEVD)

The sEVD of a matrix $A \in \mathbb{R}^{m \times m}$ has a complexity of $O(m^3)$ and is the dominating step of the MDS. Using an sEVD becomes unaffordable for large values of m, this is why, similarly to the RSVD presented in Section 1.4.2, we can take advantage of randomized algorithms to compute an approximate sEVD. The concept, described in [92] echoes the one of the RSVD: we start by approximating the column space of the matrix A by only a small number of vectors through a linear combination of the columns and then we orthogonalize them to obtain a basis (Q_Y) of an approximation of the range of A. The `RAND`, `MM1` and `QR1` operations are the ones presented in Section 1.4.1 and are identical between the RSVD (Algorithm 2) and RsEVD (Algorithm 6). Then, the matrix A is projected onto this space, from both sides to compute the product $C = Q_Y^T A Q_Y$, in algorithm 6 `MM2` and `MM3`. This projection on both sides preserves the symmetry of the input matrix. In this projection, only `MM2` is costly, as `MM3` is the product of a $k \times m$ matrix with an $m \times k$ one and its cost is therefore low compared to the one of `MM2`. We then compute a full sEVD of the $k \times k$ matrix C (`sEVD`) before forming the matrix $Q = Q_Y Q_C$ (`MM4`). Unlike with the RSVD, projecting on both sides preserves the symmetry. We will further discuss the symmetry issue in Section 2.3.

Algorithm 6: randomized linear embedding sEVD algorithm: $(Q, \Lambda) \simeq$ RsEVD(A)

Input: A a $m \times m$ matrix, k a prescribed rank
Output: an approximate factorization $A \simeq Q \Lambda Q^T$

1 Draw a $m \times k$ Gaussian random matrix Ω // RAND
2 Form the $m \times k$ matrix $Y = A\Omega$ // MM1
3 Compute the QR decomposition of Y: $Q_Y R_Y = Y$ // QR1, Q1
4 Form the $m \times k$ matrix $C = AQ_Y$ // MM2
5 Form the $k \times k$ matrix $C = Q_Y^T C$ // MM3
6 Compute the EVD of C: $Q_C \Lambda Q_C^T = C$ // sEVD
7 Form the matrix $Q = Q_Y Q_C$ // MM4
8 **return** Q, Λ

We then consider the RsEVD-MDS as the MDS algorithm (Algorithm 1) for which the `sEVD` (step 2) is processed with an RsEVD (Algorithm 6) for which the input matrix A is the $m \times m$ Gram G matrix of the MDS.

This method does not affect the symmetry of the input matrix. However having to project on both sides of the matrix, the error due to the approximation is thus doubled when compared with the error of the RSVD. As already mentioned above, this trade-off between symmetry and precision will be further discussed in Section 2.3.

2.3 Numerical behaviour of RSVD-MDS and RsEVD-MDS

This section discusses the impact of the choice of the randomized linear embedding variant on the numerical behaviour of the MDS. On the one hand, the standard RSVD used in [20, 21, 19] may (slightly) break the implicitly assumed symmetry of the MDS but requires only a single projection. On the other hand, the standard randomized sEVD (RsEVD) used in [99] preserves the symmetry but requires two projections. There is therefore a numerical trade-off.

As mentioned in Section 1.1.2, the computation of G out of D through (1.4) (see p. 9) is the first step of MDS (line 1 in Algorithm 1). The coordinate matrix $X \in \mathbb{R}^{m \times k_{\text{MDS}}}$ is then retrieved as $G = XX^{\intercal}$. As G is symmetric, it admits a unitary diagonalization so that its sEVD may be

written $G = Q\Lambda Q^T$ where Λ is diagonal and Q is unitary. In the case G is semi-definite positive, Λ has only non-negative diagonal values and we may thus write $X = Q\Lambda^{1/2}$. When the matrix is (square and) symmetric as it is the case for the Gram matrix G, the SVD ($G = U\Sigma V^T$) and the sEVD ($G = Q\Lambda Q^T$) coincide up to the sign of the eigenvalues. Indeed, the sEVD of G may also be expressed as $G = U|\Lambda|\text{sign}(\Lambda)V^T$, where $|\Lambda|$ is the diagonal matrix whose diagonal values are the absolute values of the eigenvalues and $\text{sign}(\Lambda)$ is the diagonal matrix whose diagonal values are the signs of the eigenvalues. If we note $U = Q, \Sigma = |\Lambda|$ and $V = \text{sign}(\Lambda)Q$, we observe that we also have $G = U\Sigma V^T$, which is the SVD of G. In the case G is semi-definite positive, we may thus also write $X = U\Sigma^{1/2}$. MDS may thus be equally viewed as based on sEVD ($X = Q\Lambda^{1/2}$) or SVD ($X = U\Sigma^{1/2}$) [1]

However, when randomized algorithms are employed, the approaches can differ. In particular the RSVD-MDS of Blanchard et al. [20, 21, 19] is not numerically equivalent to the RsEVD-MDS considered by Paradis [99]. Section 2.3.1 explains this trade-off while Section 2.3.2 presents a numerical study.

2.3.1 Numerical trade-off of RSVD-MDS and RsEVD-MDS

On the one hand, the standard RSVD used in [20, 21, 19] requires only a single projection (when building QQ^TG) but may (slightly) break the implicitly assumed symmetry of the MDS. Indeed, the RSVD-MDS is based on the approximation $G \approx QQ^TG$ and builds the SVD of QQ^TG. Even if G is symmetric, QQ^TG can be unsymmetric. We can still perform an SVD $QQ^TG = U\Sigma V^T$ of QQ^TG but there is then no guarantee that $X = U\Sigma^{\frac{1}{2}}$ satisfies $XX^{\intercal} = G$ nor even $XX^{\intercal} = QQ^TG$. Indeed, because QQ^TG can be unsymmetric, we cannot assume that $V = U$ anymore. Is this fatal? The hope is that QQ^TG remains *almost* symmetric because it is close to the symmetric matrix G (since $QQ^TG \approx G$). We propose to measure the loss of symmetry as follows. Considering a square matrix A, we may decompose it as the sum $A = A_{sym} + A_{skew}$ of its symmetric part $A_{sym} = \frac{A+A^T}{2}$ and of its skew-symmetric part $A_{skew} = \frac{A-A^T}{2}$. We can then define the skew-symmetry ratio $\chi(A)$ of A as follows:

$$\chi(A) = \frac{\|A_{skew}\|}{\|A\|} \tag{2.1}$$

In the RSVD case, we may expect that the skew-symmetry ratio $\chi(QQ^TG)$ is low as QQ^TG is close to the symmetric matrix G (since $QQ^TG \approx G$).

On the other hand, the possible drawback of the RsEVD-MDS [99] is that it requires two projections (when building QQ^TGQQ^T). As a consequence, it may lead to a lower accuracy than the RSVD for approximating G [64, Section 5.3]. However, contrary to the RSVD case, the approximation $G \approx QQ^TGQQ^T$ of G remains symmetric. We can thus perform an sEVD $QQ^TGQQ^T = Q\Lambda Q^T$ of QQ^TGQQ^T and write $X = Q\Lambda^{1/2}$ as the coordinate matrix associated with the approximated QQ^TGQQ^T Gram matrix.

As a summary, *a first source of error* is the approximation $\widetilde{G}$ of G, $\widetilde{G}$ being equal to QQ^TG and QQ^TGQQ^T for the RSVD and RsEVD, respectively. We denote $e_r(\widetilde{G})$ the associated *relative approximation error*:

$$e_r(\widetilde{G}) = \frac{\|G - \widetilde{G}\|}{\|G\|} \tag{2.2}$$

Besides depending on the randomization algorithm, it depends on the rank k of the Gaussian random matrix Ω. The metric may also be employed to assess the reference approximation error $e_r(\widetilde{G} = Q_k\Sigma_kQ_k^T)$ of the (deterministic) truncated sEVD (TsEVD) $G \approx \widetilde{G} = Q_k\Sigma_kQ_k^T$ of G

1. We refer Section 1.1.2 for the handling of negative eigenvalues.

at rank k. For a given rank k, we expect that the relative approximation error $e_r(\widetilde{G} = QQ^TG)$ associated with the RSVD to be lower (and thus closer to the one of the TsEVD) than the error $e_r(\widetilde{G} = QQ^TGQQ^T)$ associated with the RsEVD as the former performs only a single projection whereas the latter performs two projections. However, the RSVD-MDS (but not the RsEVD) may be affected by *a second source of error* as QQ^TG can be unsymmetric and we cannot therefore assume that $V = U$ anymore in its SVD decomposition $QQ^TG = U\Sigma V^T$. As a consequence, the coordinate matrix X (built through $X = U\Sigma^{1/2}$) does not satisfy $XX^T = G$ anymore. The subsequent *cumulative relative error* for the RSVD can be assessed with $e_r(\widetilde{G} = XX^T)$, i.e. $e_r(\widetilde{G} = U\Sigma U^T)$.

In practice, the MDS algorithm we rely on (Algorithm 1) filters out the negative eigenvalues and associated eigenvectors (line 4 of Algorithm 1). Denoting Λ^+ the positive eigenvalues and Q^+ the corresponding eigenvectors as in Section 1.1.2, we define the *relative effective error* as:

$$e_r^+(X^+) = \frac{\|G^+ - X^+X^{+T}\|}{\|G^+\|} \tag{2.3}$$

where $G^+ = Q^+\Lambda^+Q^{+T}$ is computed via a full deterministic reference sEVD and X^+ is the effective output of the considered MDS variant (RSVD-MDS or RsEVD-MDS).

2.3.2 Numerical study

We conducted a numerical study on two datasets. The first dataset is the 1502-by-1502 **Atlas Guyane** from Section 1.8.1 (see page 19). The distance matrix associated with this dataset is of full numerical rank (see Section 1.8.1), which shall emphasize the numerical trade-off discussed above. The second dataset is an artificially generated low-rank matrix. We used the Frobenius norm to compute the skew-symmetry ratios and relative errors associated with equations (2.1), (2.2), and (2.3), respectively. The results obtained with Atlas Guyane and the artificial datasets are discussed in sections 2.3.2.1 and 2.3.2.2, respectively.

2.3.2.1 Atlas Guyane data set

Figure 2.1 shows the positive eigenvalues associated with **Atlas Guyane** depending on the numerical method used for computing the sEVD. The reference sEVD is shown in black. Colored plots show the linear embedding variants using a Gaussian random matrix Ω of rank k equal to 100, 200, 300 and 400 in red, blue, green, yellow, respectively. The RSVD and RsEVD variants are represented using solid and dashed lines, respectively. We observe that the RSVD better captures the spectral behaviour than the RsEVD. This is expected as the RSVD better approximates G than the RsEVD due to the fact that it performs only a single projection ($G \approx QQ^TG$) whereas the RsEVD performs two projections ($G \approx QQ^TGQQ^T$).

Figure 2.2 [2] furthermore shows that the approximated relative error $e_r(\widetilde{G})$ associated with Equation (2.2) due to the approximation $\widetilde{G}$ of G is better for the RSVD (for which $\widetilde{G} = QQ^TG$, in blue) than the RsEVD (for which $\widetilde{G} = QQ^TGQQ^T$, in red). However, as explained in Section 2.3.1, there is a trade-off. Indeed, the RSVD-MDS algorithm is based on the assumption that $U = V$ in the SVD decomposition $QQ^TG = U\Sigma V^T$ of QQ^TG. As there is no guarantee that QQ^TG remains symmetric, the better approximation of G through the RSVD ($G \approx QQ^TG$) than that through the RsEVD ($G \approx QQ^TGQQ^T$) does not guarantee that it translates into a better approximation of X. We recall that this additional source of error due to the loss of symmetry of QQ^TG induces that the coordinate matrix X (built through $X = U\Sigma^{1/2}$) does not satisfy $XX^T = G$ anymore. We observe that the subsequent *cumulative relative error* ($e_r(\widetilde{G} = XX^T) =$

2. We provide an extended version of Figure 2.2 in Appendix A.

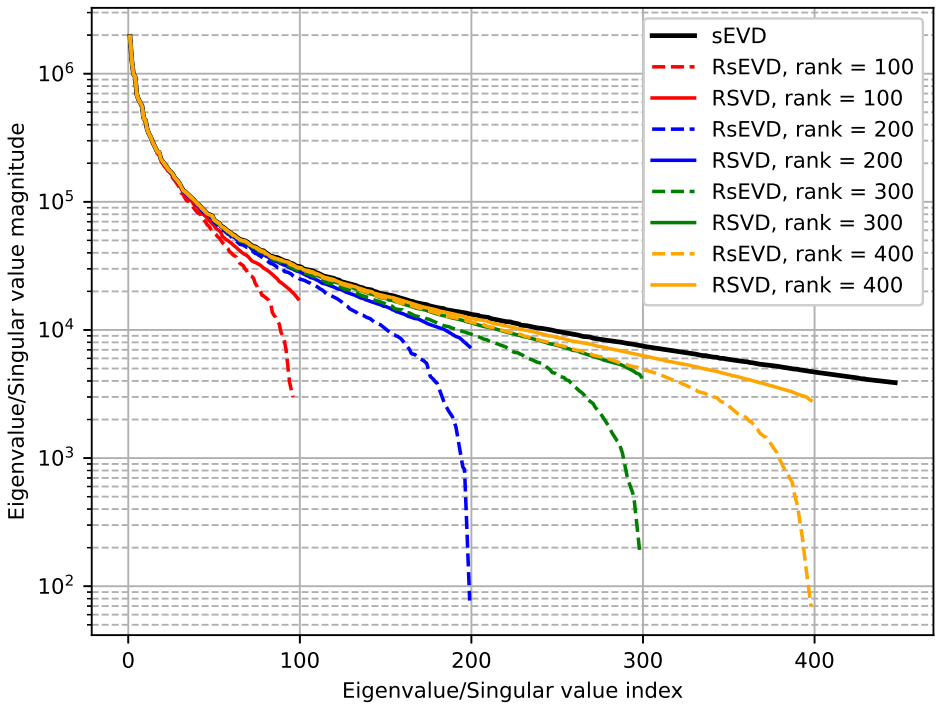


Figure 2.1 – Eigenvalues of dataset Atlas Guyane obtained using different variants of randomized linear embedding and compared to the actual eigenvalues obtained with full sEVD (in black). Each color represents a different rank for the randomized algorithms, with the solid lines being the RSVD and the dashed lines the RsEVD.

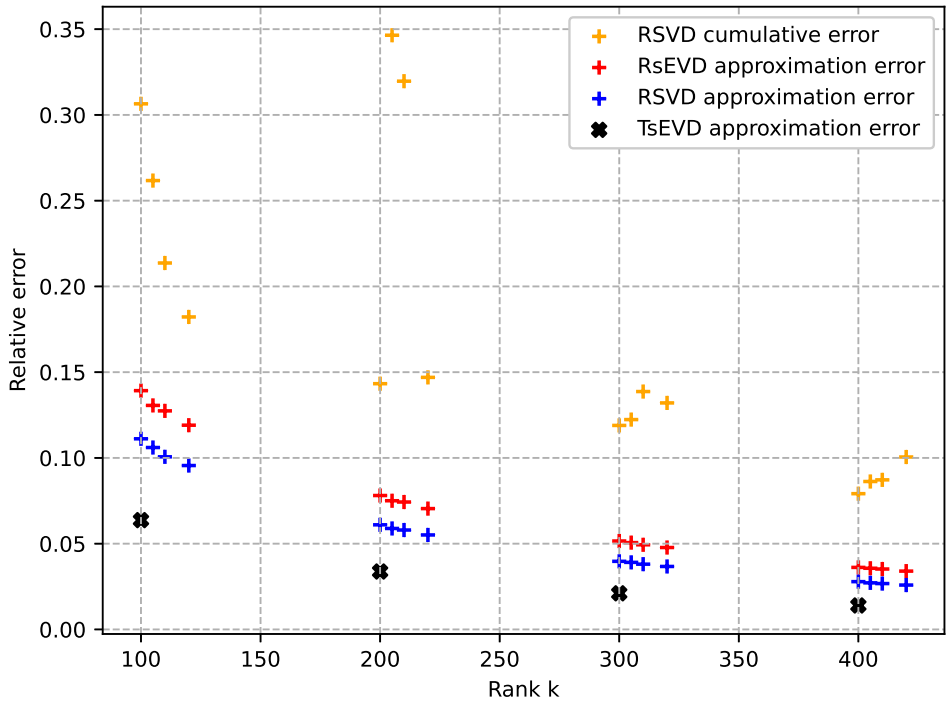


Figure 2.2 – Relative error $e_r(\widetilde{G})$ associated with Equation (2.2) due to the approximation $\widetilde{G}$ of G for the Atlas Guyane dataset. As defined in Section 2.3.1, the approximated Gram matrix $\widetilde{G}$ is equal to $Q_k \Sigma_k Q_k^T$, $QQ^T G$, $QQ^T GQQ^T$ and $U\Sigma U^T$, for the TsEVD approximation, RSVD approximation, RsEVD approximation and RSVD cumulative errors, respectively.

$e_r(\widetilde{G} = U\Sigma U^T)$, in yellow) of the RSVD gets higher than the RsEVD approximated relative error. As the RsEVD preserves the symmetry and therefore does not have such an additional source error, the RsEVD overall yields a better solution (in red) than the RSVD (in yellow) for this Atlas Guyane test case.

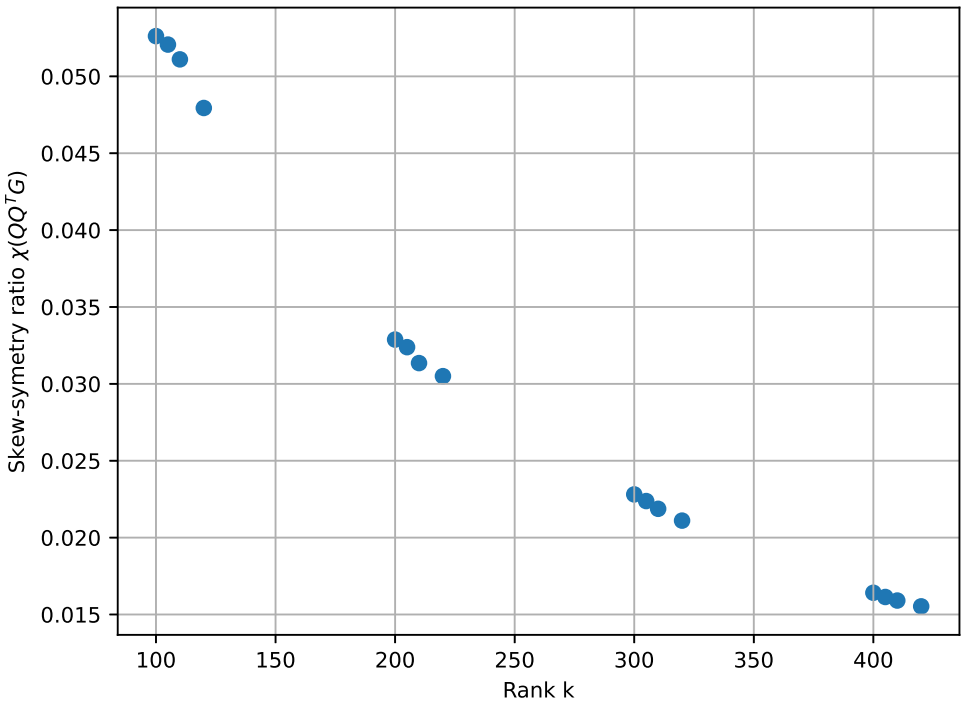


Figure 2.3 – Skew-symmetry ratio $\chi(QQ^TG)$ associated with Equation (2.1) induced by the RSVD approximation of the dataset Atlas Guyane.

Figure 2.3 shows the skew-symmetry ratio $\chi(QQ^TG)$ associated with Equation (2.1) induced by the unsymmetric RSVD approximation. We observe that the lower the rank of the projection, the higher the skew-symmetry ratio. These numerical results are consistent with the observed difference between the RSVD cumulative error represented in yellow in Figure 2.2 and the RSVD approximation error represented in blue in 2.2. They confirm the expected correlation between the skew-symmetry ratio and the second source of error for the RSVD discussed in Section 2.3.1.

As recalled in Section 2.3.1, the MDS algorithm we rely on (Algorithm 1) filters out the negative eigenvalues and associated eigenvectors (line 4 of Algorithm 1). The relative effective error $e_r^+(X^+)$ associated with the effective output point cloud X^+ and defined in (2.3) is displayed in Figure 2.4. These results confirm that the RsEVD-MDS also yields a more accurate solution than RSVD-MDS in practice on this Atlas Guyane test case for a given rank k. Figure 2.5 shows both point clouds ($k_{MDS} = 2$ in this visualisation) resulting from RsEVD (bottom left) and RSVD (bottom right) at rank $k = 10$ (rank of Ω). The point cloud obtained by a full deterministic sEVD is also provided as a reference (top).

2.3.2.2 Artificial dataset

The second dataset is an artificially generated low-rank matrix. It consists of a $1000-by-1000$ matrix of rank $r = 250$, with eigenvalues decaying linearly from 10000 for the largest one to

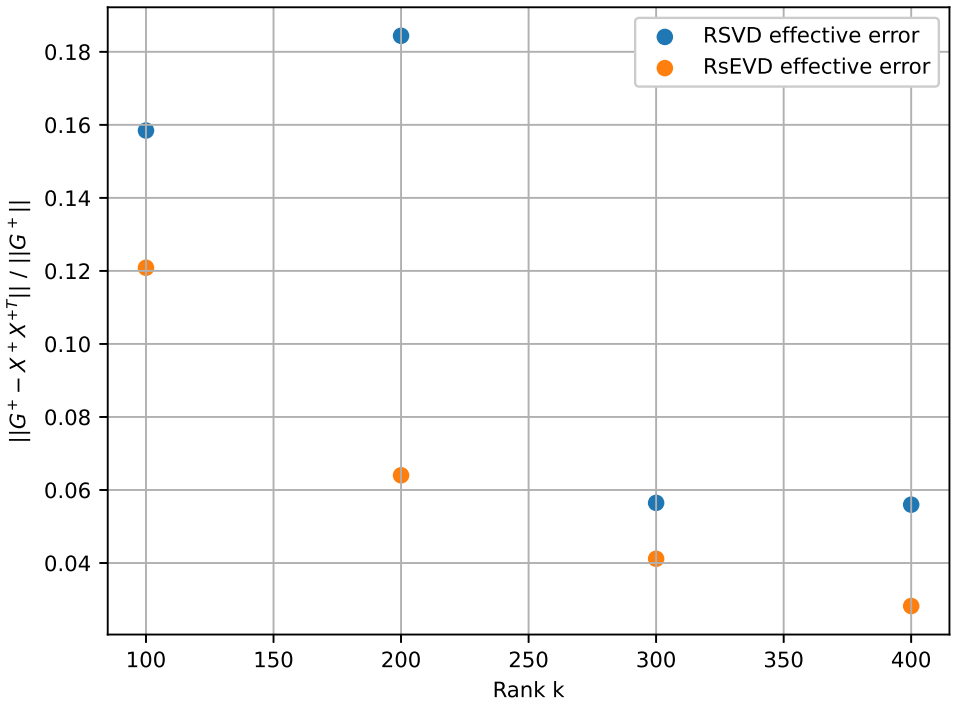


Figure 2.4 – Relative effective error $e_r^+(X^+)$ associated with Equation (2.3) induced by the RSVD and RsEVD algorithms for the dataset Atlas Guyane.

500 for the 250^th one, and all others being exactly 0, as shown in Figure 2.6 (sEVD reference plot in black). The figure also shows that both RSVD (solid) and RsEVD (dashed) capture the spectral behaviour well.

Figure 2.7 confirms that, when the rank is not well enough captured ($k = 100$ and $k = 200$), the approximated relative error $e_r(\widetilde{G})$ associated with Equation (2.2) due to the approximation $\widetilde{G}$ of G is better for the RSVD (for which $\widetilde{G} = QQ^TG$, in blue) than the RsEVD (for which $\widetilde{G} = QQ^TGQQ^T$, in red). However, as it was the case for the Atlas Guyane test case, once again, the additional source of error of the RSVD due to the loss of symmetry of QQ^TG induces that the subsequent *cumulative relative error* ($e_r(\widetilde{G} = XX^T) = e_r(\widetilde{G} = U\Sigma U^T)$, in yellow) of the RSVD gets higher than the RsEVD approximated relative error. As the RsEVD does not have such an additional source error, the RsEVD overall yields a better solution (in red) than the RSVD (in yellow) as it was the case for the Atlas Guyane test case. Figure 2.8 shows the skew-symmetry ratio $\chi(QQ^TG)$ associated with Equation (2.1) induced by the unsymmetric RSVD approximation and once again confirms the expected correlation between the skew-symmetry ratio and the second source of error for the RSVD discussed in Section 2.3.1.

On the other hand, when the rank is well captured ($k = 300$ and $k = 400$), both RSVD and RsEVD are excellent approximations. Indeed, the projection steps ($G \longrightarrow QQ^TG$ and $G \longrightarrow QQ^TGQQ^T$ for the RSVD and RsEVD, respectively) reduce to a numerical invariant for such low-rank matrices. As a consequence, the first source of error vanishes. In addition, precisely because the RSVD projection step is an invariant, its skew-symmetry ratio is the one of the symmetric matrix G, hence zero ($\chi(QQ^TG) = \chi(G) = 0$). The second source of error of the RSVD therefore vanishes too.

Figure 2.9 shows that this behaviour translates to the relative effective error $e_r^+(X^+)$ associated

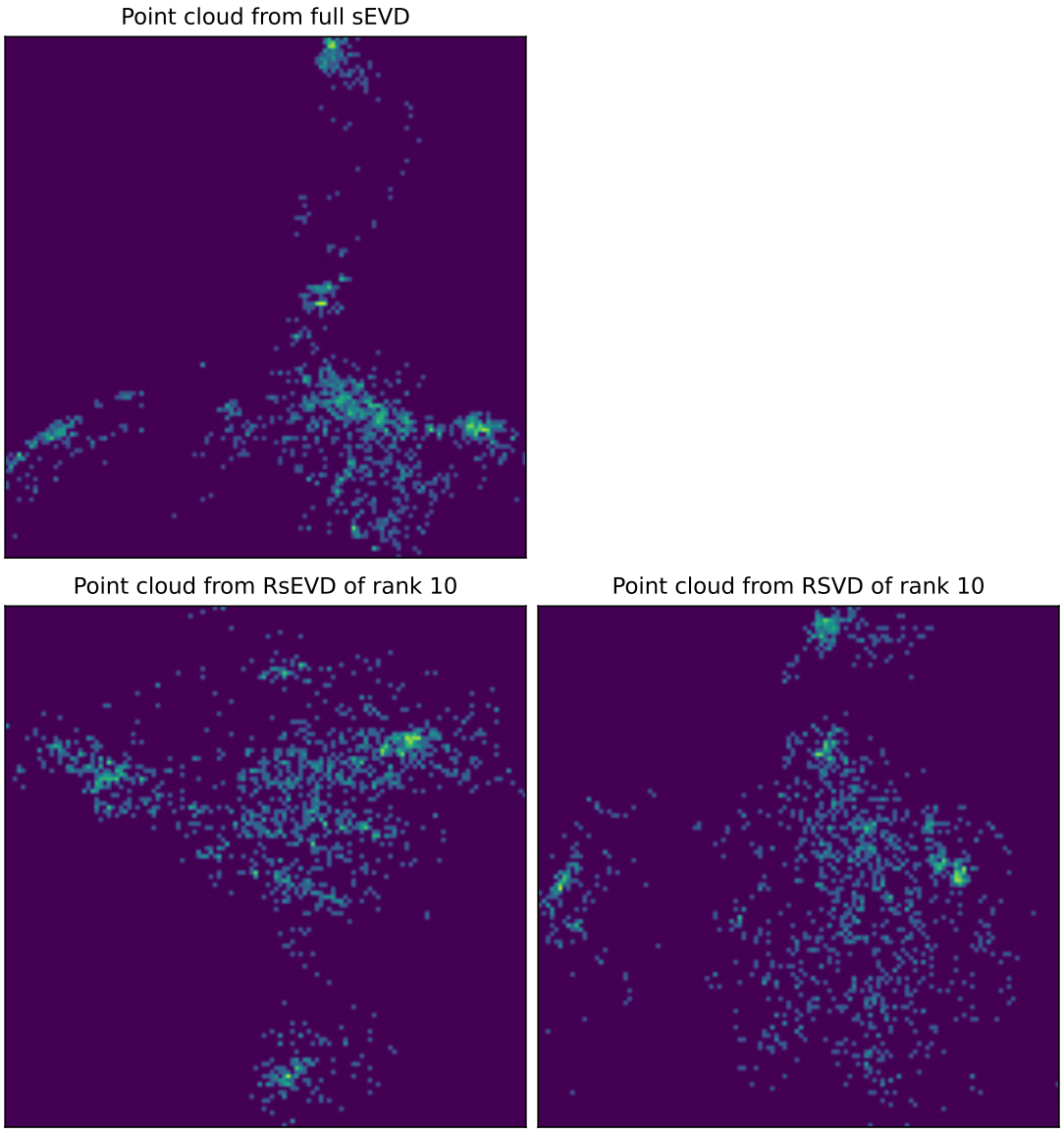


Figure 2.5 – Point clouds obtained for dataset Atlas Guyane using either full sEVD (top), RsEVD of rank 10 (bottom left) and RSVD of rank 10 (bottom right).

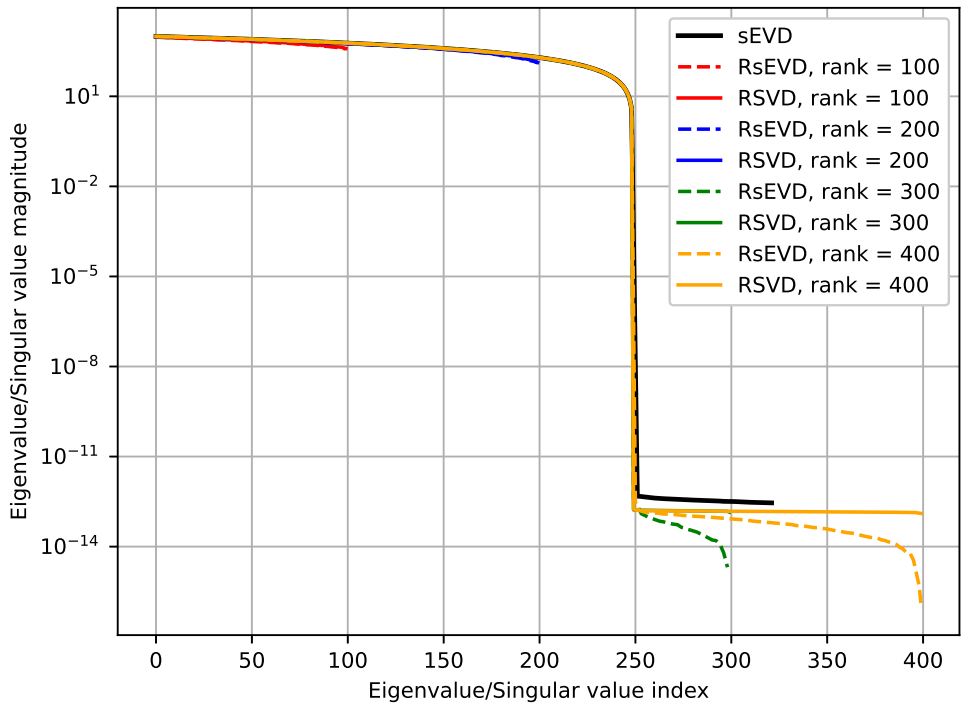


Figure 2.6 – Eigenvalues of an artifially generated dataset of size 1,000 and rank 250 obtained using different variants of randomized linear embedding and compared to the actual eigenvalues obtained with full sEVD (in black). Each color represents a different rank for the randomized algorithms, with the solid lines being the RSVD and the dashed lines the RsEVD.

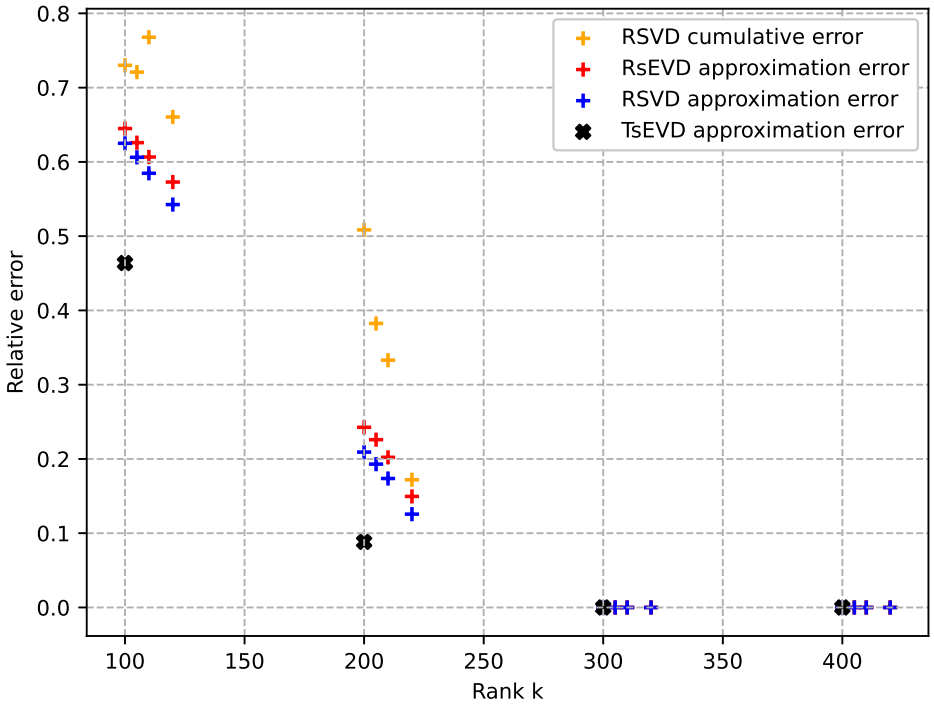


Figure 2.7 – Relative error $e_r(\widetilde{G})$ associated with Equation (2.2) due to the approximation $\widetilde{G}$ of G for the artificially generated low-rank dataset. As defined in Section 2.3.1, the approximated Gram matrix $\widetilde{G}$ is equal to $Q_k \Sigma_k Q_k^T$, $QQ^T G$, $QQ^T GQQ^T$ and $U\Sigma U^T$, for the TsEVD approximation, RSVD approximation, RsEVD approximation and RSVD cumulative errors, respectively.

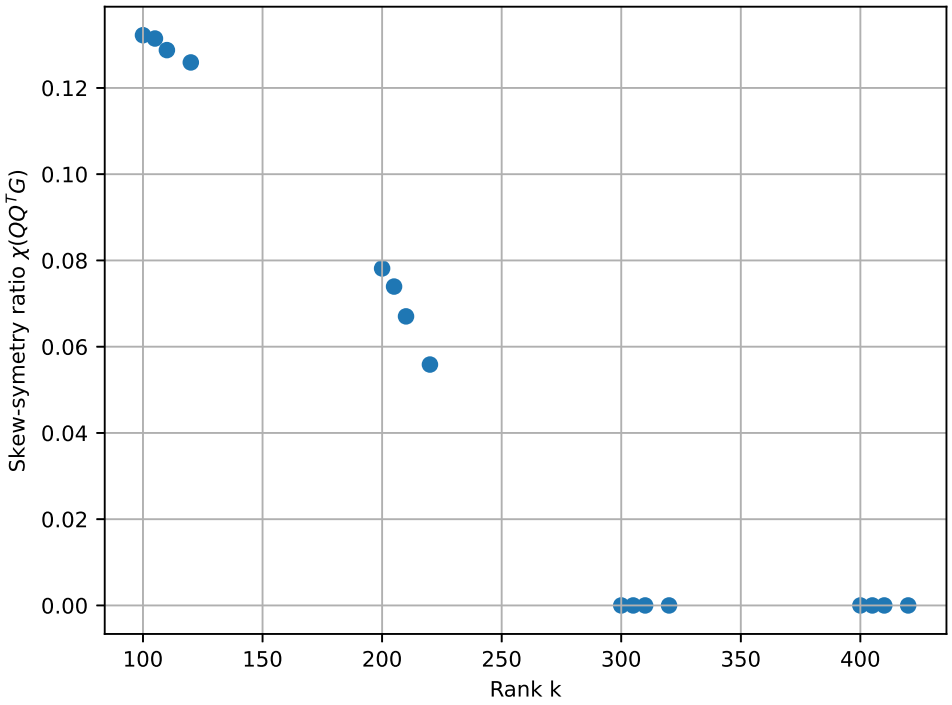


Figure 2.8 – Skew-symmetry ratio $\chi(QQ^TG)$ associated with Equation (2.1) induced by the RSVD approximation of the articially generated low-rank dataset.

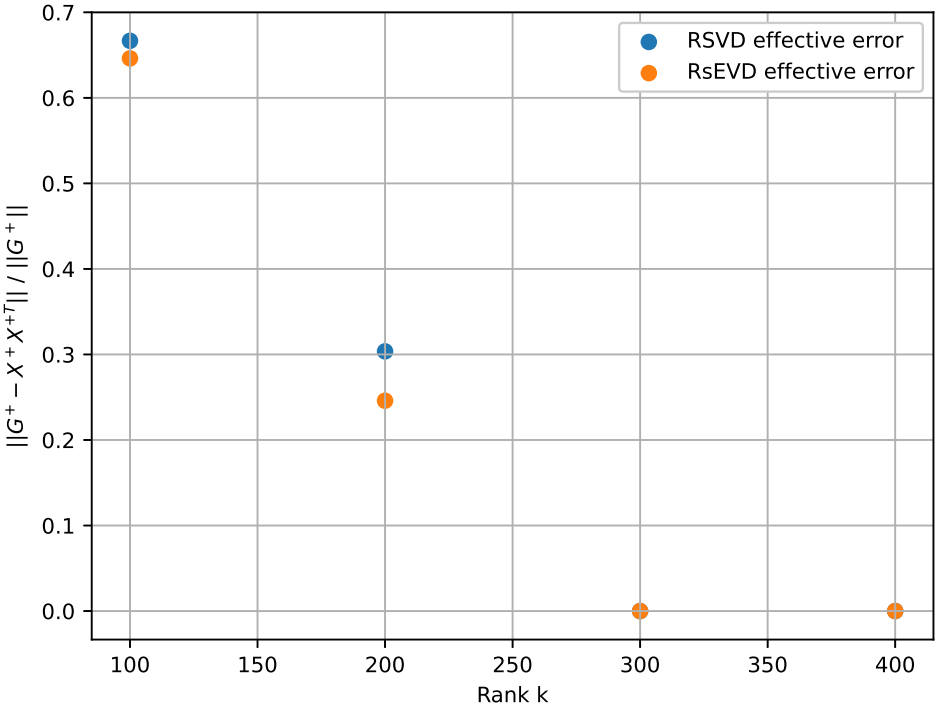


Figure 2.9 – Relative effective error $e_r^+(X^+)$ associated with Equation (2.3) induced by the RSVD and RsEVD algorithms for the artificially generated low-rank dataset.

with the effective output point cloud X^+ defined in (2.3).

2.3.3 Conclusion

This study has shown that both RSVD-MDS and RsEVD-MDS achieve an excellent numerical performance when the numerical rank is well captured. When it is not well captured, there is a numerical trade-off. On the one hand, the RSVD approximates G better than the RsEVD due to the fact that it performs only a single projection ($G \approx QQ^TG$) whereas the RsEVD performs two projections ($G \approx QQ^TGQQ^T$). On the other hand, the RSVD-MDS can be further penalised by the loss of symmetry. The study has revealed that in practice this additional penalty is not dramatic as long as the approximated Gram matrix QQ^TG remains relatively close to G, as in this case its skew-symmetry ratio remains low.

As a consequence, in a general-purpose MDS library, it may be worthwhile to include both the RSVD-MDS and RsEVD-MDS variants. While we had an RSVD-MDS (presented in the background material in Chapter 1), we have implemented an HPC version of the RsEVD-MDS variant. From a numerical point of view, it corresponds to the variant discussed in Section 2.2. From an HPC point of view, we have followed the task-based design from Section 1.6. We omit the details.

Both RSVD-MDS and RsEVD-MDS are dominated by matrix-matrix multiplications (MM). It corresponds to steps `MM1` and `MM2` in both algorithms 2 and 6. The performance and memory consumption of MM are thus keys for that of the overall MDS based on such randomized linear embedding algorithms. We motivate the improvement of distributed-memory matrix multiplication in Section 2.4. We review state-of-the-art MM for distributed-memory machines in Section 2.5. We furthermore design a new task-based symmetric MM (SYMM) in Section 2.6. Its performance is assessed in 2.7.

2.4 Motivation for improving the distributed-memory matrix multiplication

Matrix multiplication with a symmetric input matrix is a crucial mathematical kernel. As we have already mentioned, it is the most dominant kernel for randomized linear embedding algorithms [64, 32, 110, 91]. This is why we study it in this thesis [3].

In our case where the involved matrices are dense, the operation is commonly referred to as the symmetric matrix-matrix (SYMM) product and consists in computing $C \leftarrow \alpha AB + \beta C$, or $C \leftarrow \alpha BA + \beta C$, where α and β are scalars, A is a symmetric m-by-m dense matrix, and B and C are m-by-n dense matrices, or n-by-m dense matrices, respectively.

While its general (GEMM) counterpart – *i.e.*, not assuming A is symmetric (nor even square) – has been the focal point of many meticulous studies [2, 51, 115, 82], relatively little attention has been devoted to handle the specific features of SYMM in a distributed-memory context. As a consequence, its implementation in reference codes such as ScaLAPACK [18] or Elemental [108] follows the same parallel design as GEMM, consisting in a 2D block-cyclic (2D BC) data distribution. This thesis focuses on the case where $m \gg n$. We show that, though it may seem counter intuitive, the arithmetic intensity (AI) – sometimes also referred to as the operational

3. It is also a central kernel for other important numerical algorithms. For instance, it is the dominant operation in solving symmetric linear systems with multiple right-hand sides [98, 101, 112, 97, 43, 114, 119, 27, 61] or related eigenvalue problems [54, 55, 30, 111] and we refer the reader to [113] and references therein for more details on "block" and "augmented" Krylov subspace methods. As a consequence, the discussion on MM performance might be of interest out of the scope of MDS as well.

intensity – of reference implementations of SYMM – following a 2D BC data distribution – is lower than that of GEMM. We then show that alternative data distributions may improve the AI of SYMM up to achieving an equally high AI as GEMM while maintaining its memory advantage of storing only half of the dominant m-by-m matrix. We also show that with the same amount of memory as GEMM, using an extension inspired by the 2.5D algorithms [2, 121, 115] (also referred to as 3D in the literature) allows SYMM to achieve a higher AI than GEMM. In order to assess whether the higher AI may translate into a higher performance, the resulting algorithms have all been implemented in the Chameleon dense linear algebra library [3] following the task-based design proposed in [5]. This work only considers classical algorithms with a cubic computational complexity; Strassen-like variants [13] that can lead to reduced computation and communication volume are out of the scope of this study.

In the remainder of this chapter, we make the following contributions:

— we review several data distribution schemes for the A-stationary SYMM operation and analyze their communication volume;
— we propose an original adaptation of the TBS algorithm from [15] to a distributed-memory setting;
— we propose a scalable task-based implementation of SYMM independent of data distributions; and we experimentally assess the performance gained by lowering the communication volume;
— we conclude this chapter by showing performance results of our HPC implementation of RsEVD-MDS and RSVD-MDS, using SYMM to perform the `MM1` and `MM2` matrix-products.

2.5 Related work on distributed-memory matrix multiplication

The general matrix matrix multiplication (GEMM) – `i.e.` not assuming A is symmetric (nor even square) – has been the focal point of many meticulous studies [2, 51, 115, 82]. On the other hand, relatively little attention has been devoted to handling the specificity of SYMM in a distributed-memory context. As a consequence, its implementation in reference codes such as ScaLAPACK [18] or Elemental [108] follows the same parallel design as GEMM, relying on a 2D BC data distribution, we will present this data distribution in Section 2.6.1.2.

Outside of the matrix multiplication context, Beaumont et al. recently proposed to exploit the symmetry of matrices to enhance the distributed-memory Cholesky factorization of dense symmetric positive definite matrices [14]. The main idea is to rely on an alternative data distribution referred to as symmetric block cyclic (SBC). Still in the context of the Cholesky factorization, but in a sequential out-of-core setting, Beaumont et al. proposed in another study a variant referred to as Triangular Block Syrk (TBS) [15], and provided sharp bounds showing that TBS achieves the lowest possible I/O volume for this operation. However, since it does not readily apply to a distributed-memory context in the case of a Cholesky factorization, TBS was only considered in a sequential setting.

2.6 Task-based scalable SYMM

We show that in the case of interest of the randomized linear embedding algorithms ($m \gg n$), the state-of-the-art 2D BC SYMM achieves a lower AI than 2D BC GEMM, theoretically confirming the empirical observations reported in [7]. We also revisit SBC [14] in the case of the distributed-memory matrix multiplication, and proposes Triangular Block Cyclic (TBC), a distributed-memory adaptation of the ideas behind TBS [15]. Both the theoretical analysis

Scheme	P	S	Q/(mn)	AI
1. $\mathcal{G}$ 2DBC(p,q)	pq	$\frac{m^2}{P}$	$(p+q-2)$	$\frac{m}{\sqrt{P}} = \sqrt{S}$
2. $\mathcal{S}$ 2DBC(p,q)	pq	$\frac{m^2}{2P}$	$2(p+q-2)$	$\frac{m}{2\sqrt{P}} = \sqrt{S/2}$
3. $\mathcal{S}$ SBC(r)	$r^2/2$	$\frac{m^2}{2P}$	$2(r-1)$	$\frac{m}{\sqrt{2P}} = \sqrt{S}$
4. $\mathcal{S}$ TBC(c)	$c(c+1)$	$\frac{m^2}{2P}$	$2c$	$\frac{m}{\sqrt{P}} = \sqrt{2S}$

Table 2.1 – Discussed GEMM (denoted $\mathcal{G}$ in the table) and SYMM (denoted $\mathcal{S}$) A-stationary schemes together with their communication volume Q and AI. P denotes the number of nodes, S the storage per node. The communication volume Q is expressed as a factor of mn (which is the common size of matrices B and C).

(Section 2.6.1) and the experimental evaluation (Section 2.7) show that these new data distributions significantly improve above the state-of-the-art 2D BC SYMM, eventually allowing us to solve the original memory / performance discrepancy for both the randomized linear embedding algorithms and the resulting MDS.

In Section 2.6.1, we analyze the AI of reference distributed 2D BC GEMM and SYMM algorithms as well as that of our proposals for new SYMM data distributions. We present the implementation of our algorithms in Section 2.6.2 and assess the resulting communication volume and performance in Section 2.7. We show the impact on randomized linear embedding algorithms and the resulting MDS in Section 2.8.

2.6.1 Data distribution for SYMM

In this section, we present different data distributions for the $C \leftarrow \alpha AB + \beta C$ matrix product, in increasing order of complexity and AI. These results are summarized in Table 2.1. Here we assume $m \gg n$, i.e., the matrix A is much larger than both B and C, which is typically the case when dealing with the randomized linear embedding algorithms we consider for the MDS. In this case $A-stationary$ schemes are the best approaches to minimize communication volume. In an $A-stationary$ algorithm, the computations are performed on the node that owns the corresponding block (also referred to as tile) of A. In such an algorithm, the blocks of B are broadcast to the nodes that require them, and we denote Q^B the corresponding quantity of data transferred. Several nodes compute updates for a given block of C, and these updates are then reduced to the corresponding node. The communication volume for these reduce operations is denoted Q^C. The total communication volume for the multiplication is $Q = Q^B + Q^C$.

We start by the easiest case: if the complete matrix A is stored, the best solution is the 2D BC distribution (line 1 in Table 2.1 and Section 2.6.1.2), and we analyze its communication volume. We then specialize to the case where only half of matrix A is stored (because of symmetry). We use the same analysis to show that the communication volume of the 2D BC distribution (line 2) is twice larger than in the previous case. We then describe two symmetric distributions: first SBC [14] (line 3, and Section2.6.1.3), whose communication volume is lower by a factor of $\sqrt{2}$, then TBC (line 4, and Section 2.6.1.4), adapted from [15], whose communication volume is lower by another factor of $\sqrt{2}$. In total, SYMM with the TBC distribution achieves the same communication volume as 2D BC when the whole matrix is stored, thus saving a factor of 2 on storage.

Focusing on Table 2.1, the number of floating point operations does not depend on the allocation, it is $2m^2n$ in all cases: one multiplication and one addition for each product computed. Hence, the AI is inversely proportional to the communication volume Q, and varies like $\frac{m}{\sqrt{P}}$ for all 2D distributions (see left part of the AI column). However, we can also express AI as a func-

tion of the memory size of one node, denoted as S; this allows one to measure how efficient an algorithm is at using the values stored in memory. For all 2D distributions studied here, $\text{AI} = \Theta\left(\sqrt{S}\right)$; the efficiency of an algorithm is measured by how large the constant is, shown on the right part of the AI column.

2.6.1.1 Generalities

Since A is large, the best solution is to use an A-stationary algorithm: the computations are performed on the node that owns the corresponding block (also referred to as tile or submatrix) of A. In such an algorithm, the blocks of B are broadcast to the nodes that require them, and we denote Q^B the corresponding quantity of data transferred. Several nodes compute updates for a given block of C, and these updates are then reduced to the corresponding node. The communication volume for these reduce operations is denoted Q^C. The total communication volume for the multiplication is $Q = Q^B + Q^C$.

With this total communication volume Q, we can also compute the Arithmetic Intensity (AI), defined as

$$\text{AI} = \text{flop}/Q, \tag{2.4}$$

where flop is the total number of floating point operations. The number of floating point operations does not depend on the allocation, it is $2m^2n$ in all cases: one multiplication and one addition for each product computed. Hence, the AI is inversely proportional to the communication volume Q, and varies like $\frac{m}{\sqrt{P}}$ for all 2D distributions (see left part of the AI column in Table 2.1). However, we can also express AI as a function of the memory size of one node, denoted as S; this allows one to measure how efficient an algorithm is at using the values stored in memory. For all 2D distributions studied here, $\text{AI} = \Theta\left(\sqrt{S}\right)$; the efficiency of an algorithm is measured by how large the constant is, shown on the right part of the AI column in Table 2.1.

2.6.1.2 2D Block-Cyclic distribution

We first consider the situation where the whole matrix A is stored, and distributed among the nodes in a 2D BC (p, q) fashion. This situation is depicted on the left of Figure 2.10.

Figure 2.10 – Communications incurred with an A-stationary matrix multiplication, with a 2D BC $(2, 4)$ distribution. **Left:** storing the whole matrix A. One block of B follows the red path and is sent to $p - 1 = 1$ nodes. A result computed on a row of A is involved in a reduction operation on q nodes (blue path), resulting in $q - 1 = 3$ messages sent. **Right:** storing the lower half of A. Both types of communication now involve $p + q - 1 = 5$ nodes, resulting in 4 messages sent. Parts of the matrix where fewer nodes are involved are highlighted in grey.

When Considering a given column of matrix A, the corresponding values are owned by a set of p nodes. Each of these nodes must receive all values in the corresponding row of B, which is owned by another set of nodes. The best possible case is that the second set is included in the first one: in that case, each value of B must be sent to $p-1$ nodes, and this incurs a communication volume of $n(p-1)$. Since there are m columns in A, in total we get $Q^B = mn(p-1)$ (in the worst case, the set of nodes that own the blocks of B is disjoint from the set of nodes that own the blocks of A, and we get $Q^B = mnp$).

Similarly, consider a given row of matrix A. Since the nodes that own this row need to perform one reduction per column of C to send the total to the owner of the corresponding block in C, the total communication volume for the blocks of C is $Q^C = mn(q-1)$ in the best case, and $Q^C = mnq$ in the worst case (when the owner of a block in C never belongs to the corresponding set of nodes in the row of A).

The best case can be achieved if C is distributed with the same (p,q) 2D distribution, and B is distributed with the transpose (q,p) distribution.

In total, the communication volume is $Q^{p,q}_{\text{GEMM}} = mn(p+q-2)$. In practice, we often choose $p \simeq q \simeq \sqrt{P}$, so that $Q^{\text{2DBC}}_{\text{GEMM}} \simeq 2mn(\sqrt{P}-1)$. Asymptotically, the AI is $\text{AI}^{\text{2DBC}}_{\text{GEMM}} \simeq \frac{2m^2n}{2mn\sqrt{P}} = \frac{m}{\sqrt{P}}$, with a memory usage $S = \frac{m^2}{P}$, which yields $\text{AI}^{\text{2DBC}}_{\text{GEMM}} \simeq \sqrt{S}$. This result is summarized in Table 2.1, line 1.

In the case where we only store the lower part of a symmetric matrix, the 2D BC distribution is only applied to the lower half of the matrix, the upper tiles are not being stored at all. The result is depicted on the right of Figure 2.10. We can apply the same kind of reasoning as for the previous case. However, now the set of nodes that own a given column of A can be of size [4] up to $p+q-1$: the p nodes that own the (truncated) column, plus the q nodes that own the (truncated) row that completes the column (the total is $p+q-1$ because one node belongs to both the row and the column). Again, the best case is when the set of nodes that own the blocks of B is included in these $p+q$ nodes, and this yields a communication volume $Q^B = mn(p+q-2)$. Similarly, we get $Q^C = mn(p+q-2)$.

In total, $Q^{p,q}_{\text{SYMM}} = 2mn(p+q-2)$. As we can see, the communication volume is twice as large as in the previous case, and can be written as $Q^{\text{2DBC}}_{\text{SYMM}} \simeq 4mn(\sqrt{P}-1)$. Asymptotically, the AI is $\text{AI}^{\text{2DBC}}_{\text{SYMM}} \simeq \frac{2m^2n}{4mn\sqrt{P}} = \frac{m}{2\sqrt{P}}$, with a memory usage $S = \frac{m^2}{2P}$, which yields $\text{AI}^{\text{2DBC}}_{\text{SYMM}} \simeq \sqrt{S/2}$. This result is summarized in Table 2.1, line 2. As we can see, the AI is twice smaller compared to the previous case, but since the memory usage is also smaller by a factor of 2, the AI expressed as a function of S is only lower by a factor of $\sqrt{2}$.

In short, this means that if the memory of the nodes is the limiting factor, storing half the matrix allows to use half as many nodes, which partially offsets the overhead in terms of communication volume.

2.6.1.3 Symmetric Block Cyclic distribution

In order to reduce the amount of communication, we need to make sure that the nodes that own a truncated column of A are the same as the nodes that own the corresponding truncated row. The Symmetric Block Cyclic distribution (SBC) has been proposed in the context of the Symmetric Rank-k update (SYRK) and the Cholesky factorization [14], where a similar issue appeared. We describe here the basic version of SBC, defined for an even integer $r > 2$ (see

4. The first q (respectively the last p) columns of A involve a slightly smaller number of nodes, because not all nodes appear in the truncated row (respectively column). The corresponding zones are highlighted in grey on the right of Figure 2.10. However, since we are interested in large matrices A where $m \gg p,q$, we decide to neglect this effect.

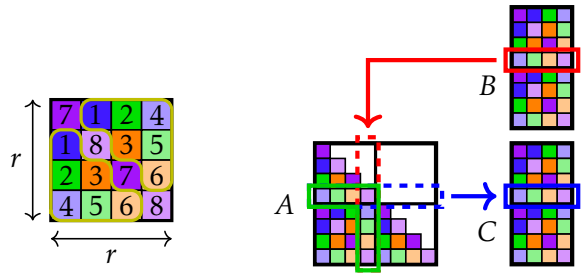


Figure 2.11 – Symmetric Block Cyclic distribution. **Left:** the pattern with $r = 4$, using $P = 8$ nodes. The symmetrical lower and upper parts are highlighted. **Right:** communications induced when using SBC. Communications related to a row of matrices B and C both involve r nodes, resulting in $r - 1 = 3$ messages sent.

Figure 2.11). It consists of a symmetric $r \times r$ pattern with $P = r^2/2$ nodes: $\frac{r(r-1)}{2}$ nodes are organized arbitrarily in one half of the pattern, and symmetrically on the other half. The remaining $\frac{r}{2}$ nodes are each placed on two locations in the diagonal.

For any i, the row i and the corresponding column i of the pattern contain the same set of r nodes. We can compute the amount of data transferred involved by using this distribution in an $A-stationary$ SYMM operations: each block of B is sent to a set of r nodes, and r nodes are involved in each reduction operation for a given block of C. If we again consider the best case, we get that $Q^B = Q^C = mn(r-1)$, which yields $Q = 2mn(r-1)$. Since $r = \sqrt{2P}$, we can write this as $Q^{\text{SBC}}_{\text{SYMM}} = 2mn(\sqrt{2P} - 1)$: this improves over the 2D BC distribution by a factor of $\sqrt{2}$. Since the memory usage is the same, the AI is also improved by a factor of $\sqrt{2}$, which gives $\text{AI}^{\text{SBC}}_{\text{SYMM}} \simeq \sqrt{S}$: SBC obtains the same AI as the 2D BC distribution when storing the whole matrix. This result is summarized in Table 2.1, line 3.

2.6.1.4 Triangular Block Cyclic distribution

Another recent work proposed a Triangular Block approach for the SYRK operation [15], which achieves provably the lowest possible quantity of data transferred. This work was presented in the context of sequential out-of-core computations, but we propose here a way to transform it into an allocation for distributed nodes.

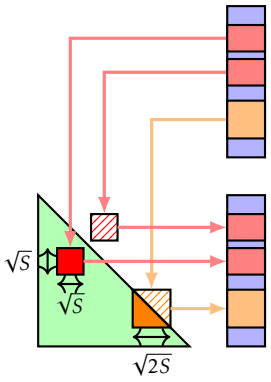


Figure 2.12 – Triangles along the diagonal use fewer communications: both the red square and the orange triangle contain S elements. However, the corresponding operations for the square require $2\sqrt{S}$ rows of B, and only $\sqrt{2S}$ rows for the triangle.

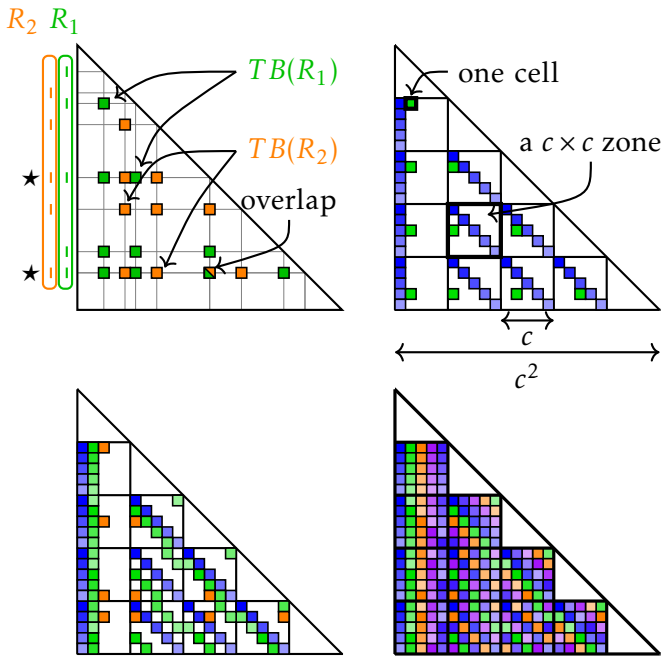


Figure 2.13 – **Top Left:** two triangle blocks. An overlap happens when two triangle blocks have two rows in common ($\star$). **Top Right:** first column of the triangle-blocks in the TBC pattern. Gaps must be introduced for the triangle-block in the next column to avoid overlapping. **Bottom Left:** the gaps in the next column need to be larger, with a "wrap around" when reaching the bottom of the zone. **Bottom Right:** complete pattern for all the zones.

We remind that with the SBC distribution, each node is assigned $S = \frac{m^2}{2P} = \frac{m^2}{r^2}$ blocks, and needs to receive 2 rows of matrix B for each repetition of the pattern. The number of required rows of matrix B is thus $h = \frac{2m}{r} = 2\sqrt{S}$. This is similar to assigning a square part of the matrix to a node, as shown on Figure 2.12: if a node is responsible for the red square of side $\sqrt{S}$, it needs to receive $\sqrt{S}$ rows of B to perform the operations in the lower half, and another $\sqrt{S}$ rows to perform the operations in the upper half.

The idea of the TBS algorithm [15] stems from the observation that, thanks to the symmetry of matrix A, triangular parts along the diagonal of A are involved in operations that require even fewer communications than square parts. Indeed, as shown with the orange triangle on Figure 2.12, if a node owns a triangle containing S elements along the diagonal of A (its side length is thus $\sqrt{2S}$), it only needs to receive $\sqrt{2S}$ rows from matrix B since the operations on the lower and upper halves require the same rows of B. Similarly, this node only needs to participate in reduction operations on $\sqrt{2S}$ rows for matrix C.

The TBS algorithm defines a solution where these favorable properties are extended to blocks away from the diagonal by ensuring that each node is assigned a set of blocks which can be gathered into a diagonal triangle using a symmetric permutation.

This leads to the notion of *triangle-blocks*, defined, for a given set R of row indices, as the set of blocks of the matrix A that a node can own while only requiring rows of matrix B indexed by R. Formally, the triangle-block associated with R is $TB(R) = \{(i, j) \in R^2 | i > j\}$. Figure 2.13 (left) shows examples of two triangle blocks. This notion generalizes the "triangle along the diagonal", since a triangle-block with $|R| = h$ contains $\sim \frac{h^2}{2}$ blocks of A, and the corresponding operations involve only h rows of matrices B and C. A triangle-block can indeed be seen as a triangle along the diagonal, up to reordering of the rows and columns of the matrix.

The key contribution of [15] is a method that makes it possible to partition almost all of the matrix in disjoint triangle-blocks. This requires to assign a set of rows R_p to each node, so that any two sets R_p and $R_{p'}$ have at most one row in common. Indeed, as can be seen on the top-left of Figure 2.13, two triangle blocks overlap if they share two row indices. This implies that the two corresponding nodes will receive the data necessary to perform an operation, however only one of them will actually perform it; communicating that data to the other node was not useful. Finding a distribution with no overlap ensures that we minimize the communication volume.

To apply these ideas in a distributed-memory setting, we propose to build a pattern where each triangle-block is assigned to a different node. Since this pattern is symmetric, for simplicity we only describe its lower half. We fix a prime integer $c > 2$, and build a symmetric pattern of size $c^2 \times c^2$, divided in $c \times c$ square *zones*, each containing c^2 cells. We partition the square zones among triangle-blocks, and the main idea is that each triangle-block has one cell in each zone. The top right of Figure 2.13 shows how the first c triangle-blocks are organized. The next triangle-block is also shown, and we can see that a gap must be introduced at each new row to ensure that it does not overlap with the previous blocks. The bottom left of Figure 2.13 shows the next step: the other blocks of the second column can be assigned with the same gaps. However, the next block in the third column needs to have larger gaps at each row to ensure that no overlap happens. With such large gaps, the last row would be outside of the zone, so the actual row is chosen modulo c: this effectively "wraps around" at the boundary of the zone. The final partition with all the triangle-blocks is shown at the bottom right of this figure.

With this partitioning, each node receives $\frac{c(c-1)}{2}$ cells from the lower half of the pattern, but the cells in the triangular zones along the diagonal remain unassigned. The choice of the pattern size ensures that if we exclude the diagonal cells, each of these triangular zones also contain $\frac{c(c-1)}{2}$ cells. We can thus assign them to c additional nodes, which describes the entire lower half of the pattern. By replicating this symmetrically, we get a square pattern where only the non-diagonal cells remain unassigned. A more precise description of the pattern is described in Algorithm 7, and the resulting pattern for $c = 3$ is provided in the left of Figure 2.14. The iteration (i, j) of the loop in line 4 assigns the triangle-block which contains the cell of coordinates (i, j) of the top-most zone (which is the cell $(i + c, j)$ of the pattern). The idea behind line 5 is that each node should access one row in each zone: the value uc indicates the index of the first row on the u-th zone, and the value $i + (u - 1)j \mod c$ is the index of the row within this zone. We can see in this formula that there is a gap of size j between one row and the next (thus for two successive values of u), and that there is a modulo operation to perform the "wrap around", as described in the successive diagrams of Figure 2.13. The results from [15] (in particular Lemma 5.5), together with the condition that c is prime, ensure that the sets of rows assigned to different nodes overlap exactly once, so the sets of cells assigned to the nodes are disjoint, and each row contains exactly $c + 1$ nodes.

Algorithm 7: TBC(c) pattern, on $P = c(c + 1)$ nodes.

▷ // Assign triangular zones (red nodes)
1 **for** $i \in \{0, \ldots, c-1\}$ **do**
2 $\quad R \leftarrow \{i \cdot c + u \,|\, 0 \leq u \leq c-1\}$;
3 $\quad$ Assign cells in $\{(x, y) \in R^2 | x \neq y\}$ to a new node;
▷ // Assign non-triangular zones (remaining nodes)
4 **for** $(i, j) \in \{0, \ldots, c-1\}^2$ **do**
5 $\quad R \leftarrow \{uc + (i + (u-1)j \mod c) | 1 \leq u \leq c-1\}$;
6 $\quad$ Assign cells in $\{(x, y) \in (R \cup \{j\})^2 | x \neq y\}$ to a new node;

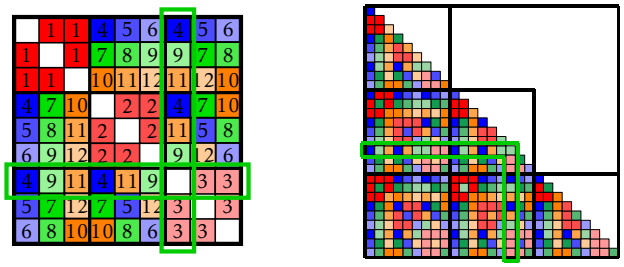


Figure 2.14 – Triangular Block Cyclic distribution. **Left:** the pattern with $c = 3$, using $P = 12$ nodes, with no allocation on the diagonal blocks. **Right:** allocation of this pattern on a 27×27 matrix, where the diagonal blocks of the pattern are filled with the greedy algorithm. As shown in the highlighted part, each communication (related to matrix B or C) involves 4 different nodes (nodes 3, 4, 9, 11 in this example). For comparison, in the (3, 4) 2D BC distribution, each communication involves 6 nodes.

This procedure results in a symmetric pattern of size $c^2 \times c^2$, in which the diagonal cells are not allocated. However, there are $c(c+1)$ nodes in total, and only c^2 diagonal cells. Each of these diagonal cells can be allocated to any node already present on the row without increasing the communication volume.

To obtain the final allocation, we replicate this incomplete pattern over the matrix A, and apply a greedy algorithm to allocate the remaining blocks : for each unassigned block, we pick the node with the lowest number of assigned blocks among all the nodes present in the row (and thus in the column, by symmetry of the pattern). The resulting allocation is thus not exactly a *cyclic* allocation, but it can nonetheless be computed very quickly. An example is provided on the right of Figure 2.14.

This pattern uses a total number of nodes $P = c(c+1)$, and each row and column of matrix A is allocated to a set of $c+1$ nodes. The communication volume can be evaluated just like previously: each communication related to a row of matrix B or C involves $c+1$ nodes, and we obtain $Q^B = Q^C = mnc$ in the best case. Thus, $Q = 2mnc \simeq 2mn\sqrt{P}$. This corresponds to another improvement by a factor of $\sqrt{2}$ over SBC, and asymptotically the same communication volume as with the 2D BC distribution when storing the whole matrix. The AI is also improved by a factor of $\sqrt{2}$: $\text{AI}^{\text{TBC}}_{\text{SYMM}} \simeq \sqrt{2S}$. This is the best of both worlds: the reduced memory storage gained by storing only half the matrix, and the reduced communication volume. This result is summarized in Table 2.1, line 4.

2.6.2 Implementation

Fully-featured distributed-memory dense linear algebra libraries such as ScaLAPACK [18] or Elemental [108] implement SYMM with a 2D BC data distribution. Such a regular data distribution makes it possible to easily set up MPI communicators along rows and columns and ensure collective communications. On the contrary, if we want to consider irregular data distributions, like those discussed in Section 2.6.1, it may be challenging to implement the corresponding code directly through the MPI interface. We therefore aim at designing a SYMM routine completely independent of the proposed mappings so that we can then effortlessly implement any non-trivial mapping. Task-based programming allows for such a separation of concerns [67, 5]. In addition, we want to assess whether this is feasible with a code which is easy to write, read and maintain. To this end, we decided, more specifically, to rely on the STF model [41, 10] introduced in Section 1.5, which allows for writing a parallel code which

Algorithm 8: Sequential block SYMM.

```
1 for j = 1...N do
2     for i = 1...M do
3         for l = 1...M do
4             op ← if i = l then symm else gemm;
5             block_A ← if i ≤ l then A_{i,l} else A^T_{l,i};
6             op(block_A, B_{l,j}, C_{i,j});
```

Algorithm 9: STF block SYMM.

```
1 for j = 1...N do
2     for i = 1...M do
3         for l = 1...M do
4             op ← if i = l then symm else gemm;
5             blk_A ← if i ≤ l then A_{i,l} else A^T_{l,i};
6             insert_task(op, blk_A:R, B_{l,j}:R, C_{i,j}:RW);
```

resembles a sequential code one would naturally write. Finally, we do not want to trade off performance with elegance, and we thus rely on the latest developments for the scalability of the STF model [5] to design appropriate communication patterns while preserving the compactness of the expression. Without loss of generality, for the sake of conciseness, we restrict the presentation to the $C \leftarrow AB+C$ case, where A is symmetric and only its lower part explicitly stored. We also assume blocks of size b-by-b, so that A is a M-by-M block matrix ($m = M*b$) and B and C are M-by-N block matrices ($n = N*b$). The sequential algorithm of the SYMM consists of three nested loops where the two innermost loops perform a matrix - block-column product ($C_{*,j} \leftarrow C_{*,j} + AB_{*,j}$), while the outer loop goes through all N block-columns. Algorithms 8 and 9 show the equivalent SYMM block and STF implementation to Algorithms 4 and 5 presented in Section 1.5 for GEMM.

The first stage of the STF algorithm presented in Algorithm 9 (not reported in the pseudo-code) is to register the data to operate on to the runtime system. In our case, the data are the $A_{i,l}$, $B_{l,j}$, and $C_{i,j}$ blocks ($1 \leq j \leq N$, $1 \leq i \leq M$, $1 \leq l \leq M$). When a data is registered, the runtime system is informed of which node owns it. This information can be later retrieved through a rank (`rk()`) function. With this simple paradigm, we are able to instruct the runtime system how to set up the data mappings discussed in Section 2.6.1. Other than that, we may observe that the STF pseudo-code is very similar to a sequential one. However, instead of being directly executed, GEMM and SYMM operations on blocks are *inserted* as *tasks* through the `insert_task` primitive. The aim of such an insertion is to make the call asynchronous. Data dependencies between tasks are inferred by the runtime system thanks to the data access modes. The basic access modes being Read (R), Write (W), and Read Write (RW), the most straightforward implementation would be to associate a R access mode to blocks of A and B and a RW mode access to blocks of C. Such an approach, following the programming models of [131, 4], would be a valid STF code for distributed-memory machines. Indeed, the runtime system can not only infer the dependencies between tasks but it may also trigger the appropriate communications by moving around the data causing the dependencies. Indeed, in this model [131, 4], tasks are executed onto MPI ranks that own data accessed in RW mode ($C_{i,j}$, in our case); the other data are automatically moved there by the runtime system before executing the corresponding task.

This baseline STF algorithm would however prevent us from implementing the algorithms

Algorithm 10: Scalable STF block A-stationary SYMM.

```
1 for j = 1...N do
2     for i = 1...M do
3         for l = 1...M do
4             op ← if i = l then symm else gemm;
5             block_A ← if i ≤ l then A_{i,l} else A^T_{l,i};
6             rank ← if i ≤ l then rk(A_{i,l}) else rk(A_{l,i});
7             insert_task(op, block_A:R, B_{l,j}:R, C_{i,j}:DIST_REDUX, ON_RANK=rank);
```

from Section 2.6.1. Indeed, first, performing a task on the MPI ranks owning data accessed in RW access mode implies that the C matrix stays in place and A and B are transferred through the network. The analyses from Section 2.6.1 assume that, because the matrix A is the largest one, it is instead preferable to keep A in place and only move around blocks of B and C. To implement such a *A-stationary* scheme, we must be able to explicitly instruct the runtime system where to perform tasks. Algorithm 10 does so through the `ON_RANK` (l. 7) directive which is used to define the rank where to execute a task which, in our case, is the owner of the corresponding block of A (l. 6); this implies that the corresponding blocks of B (and C) will be transparently sent to (and from) that same node by the runtime system prior to the task execution. A second extension is required to achieve the AI of the algorithms devised in Section 2.6.1. Indeed, after a task has been computed on the chosen rank, the programming model requires that a data accessed in W or RW mode is sent back to the owner rank [131, 4]. In the SYMM case, $C_{i,j}$ being accessed in RW mode, it must therefore be sent back to the MPI rank that owns it unless it is the same rank as the one where the task is executed. It must be noted that, with such a A-stationary mapping, all the tasks contributing to a $C_{i,j}$ block can be executed by different nodes. With $C_{i,j}$ accessed in RW mode, a single copy exists that will be sent back and forth between the contributing nodes and all the corresponding tasks will be executed sequentially [131, 4]. To overcome this limitation, the `DIST_REDUX` mode [5] (l. 8) can be used: in this case nodes will compute, locally, contributions to $C_{i,j}$ which are stored in a temporary buffer allocated upon executing a task that requires it. All these contributions can be computed concurrently and, in a second step, reduced in the $C_{i,j}$ block. The reduction is transparently detected by the runtime and performed through a binary tree – this is a reasonable choice given the typical number of contributors and the size of the message [5]. Once a contributor has sent its contribution, the corresponding memory buffer is freed. The reduction operation allows for further parallelism and more accurately matches the A-stationary product described in the literature [51]. Altogether, both these extensions allow us to reach our main goal of designing a SYMM routine completely independent of the mappings so that we can then effortlessly implement any non-trivial mapping and achieve the expected associated AI.

2.7 Performance of task-based scalable SYMM

The code designed in Section 2.6.2 allows us to assess all the mappings discussed in Section 2.6.1. We study their impact on the AI and performance. We have implemented Algorithm 10 on top of the `starpu` [10] task-based runtime system and the `newmadeleine` communication back-end, which, combined, support the dynamic detection of collective communications [36].

In an applicative setting, where no synchronization is required between filling and computing the matrices, blocks of B are transferred through broadcasts transparently: the runtime detects

them through dependencies in the DAG. In our benchmarking setting, artificial tasks are added to mimic the dependencies that allow a similar detection.

We conducted our study in double precision on the Skylake partition of the Très Grand Centre de Calcul (TGCC) computer. It has an Infiniband EDR interconnect. Each node has 192 GB of DDR4 memory and is composed of two 24 cores Intel Skylake 8168 @ 2.7 GHz processors (48 cores per node). Intel MKL v. 19.0.5.281 provides the implementation of GEMM and SYMM (single-core) tasks. All codes have been assessed with a block size b equal to 256, 512 and 1024, preliminary experiments having shown that these values allow for a good efficiency. Each configuration has been executed five times and we retrieve the median performance. GEMM and SYMM algorithms are executed with $(p = 8, q = 7)$ on 56 nodes for 2D BC distributions. 2D SBC ($r = 11$) and 2D TBC ($c = 7$) SYMM are executed on 55 and 56 nodes, respectively. 2.5D SBC ($s = 2$,$r = 8$) and 2.5D TBC ($s = 2$,$c = 5$) SYMM are executed on 56 and 60 nodes. The matrix size (m) of A varies while the number of columns of B and C is constant ($n = 8,192$).

The top plot of Figure 2.15 presents the AI of the STF algorithms discussed in Section 2.6.1 as defined in Equation 2.4: AI = flop/Q. The total volume of communication Q is retrieved by StarPU. The results show that the expected theoretical ratios of AI from Section 2.6.1 are successfully achieved in practice.

All in all, TBC SYMM AI roughly matches that of 2D BC GEMM (transparent dashed red, square).

The bottom of Figure 2.15 presents the resulting per-node performance. The first observation is that the AI gains of SBC and TBC do yield compelling performance benefits on lower size matrices where the AI is not sufficient to ensure a good overlapping between communications and computations. The proposed STF design with SBC and TBC SYMM achieves a performance roughly comparable with 2D BC GEMM, while requiring to store only half of the dominant matrix. The second main observation is that the AI advantage of 2.5D SBC and TBC does not consistently translate into performance improvement. While 2.5D symmetric distributions perform well on small problems, they do not outperform the 2D case on larger ones. This is consistent with the literature [5]. TBC is more impacted by this performance discrepancy despite having an higher AI than GEMM. A preliminary analysis suggests that this is due to contention in the network that happens because, unlike in BC, in TBC any rank participates in multiple broadcast communications.

Figure 2.16 presents the comparison of the GEMM and SYMM performance of our STF approach with state-of-the-art distributed-memory dense linear algebra libraries proposing a A-stationary implementation of SYMM, namely ScaLAPACK [18] (yellow) and Elemental [108] (black). We also report the GEMM performance of SLATE [50], a potential successor to ScaLAPACK for which A-stationary SYMM was not available. The first observation is the important gap between SYMM and GEMM performance of both ScaLAPACK and Elemental libraries. These results confirm the empirical observation that SYMM state-of-the-art codes achieve a lower performance than their GEMM counterpart. We recall that both these libraries implement SYMM with a 2D BC data distribution. The second main observation is that the STF algorithms proposed in Section 2.6.2 significantly improve over the A-stationary ScaLAPACK and Elemental SYMM reference implementations. This illustrates the strength of the programming model for designing efficient communication schemes with a high-level expression.

2.8 Performance of MDS with randomized linear embedding

In Section 2.7, we showed the performance of our task-based design of SYMM and the improvement it makes compared to the reference SYMM codes. Now we are going to present

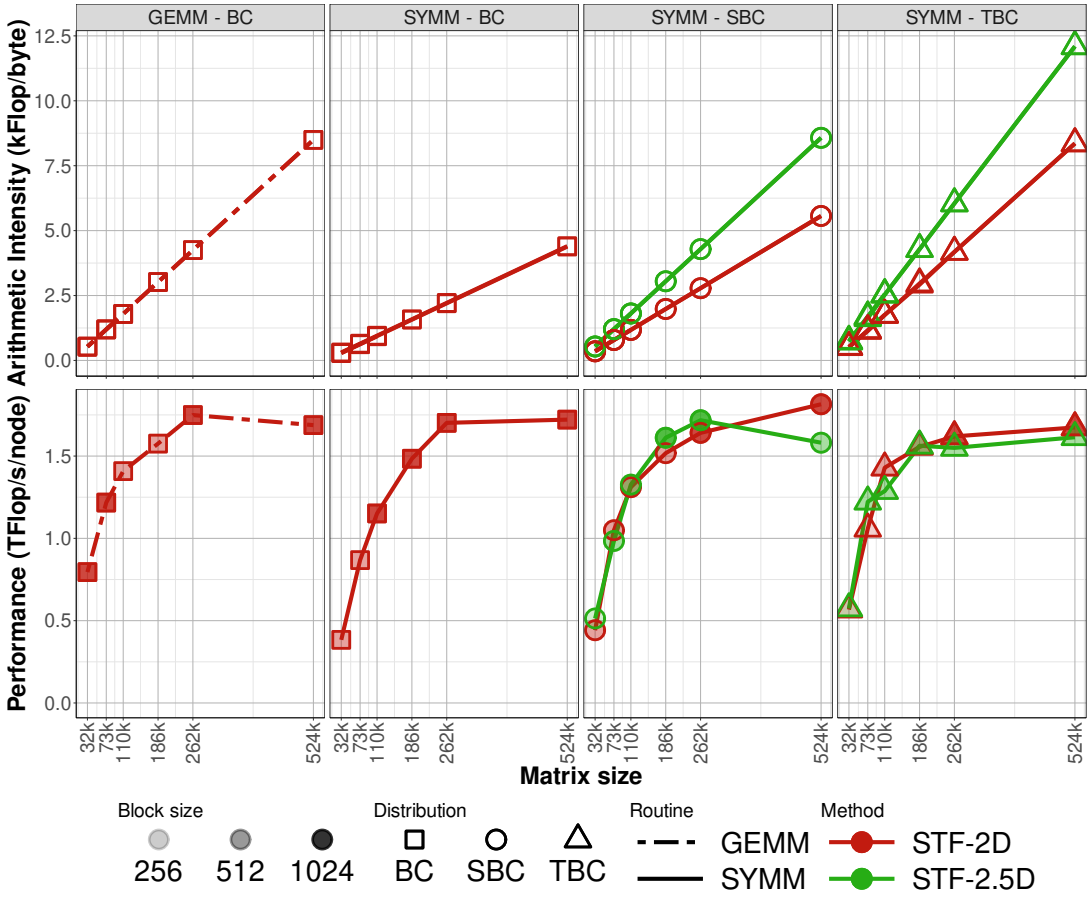


Figure 2.15 – AI (top) and per-node performance (bottom) of the STF algorithm of section 2.6.2 with the various distributions of section 2.6.1.

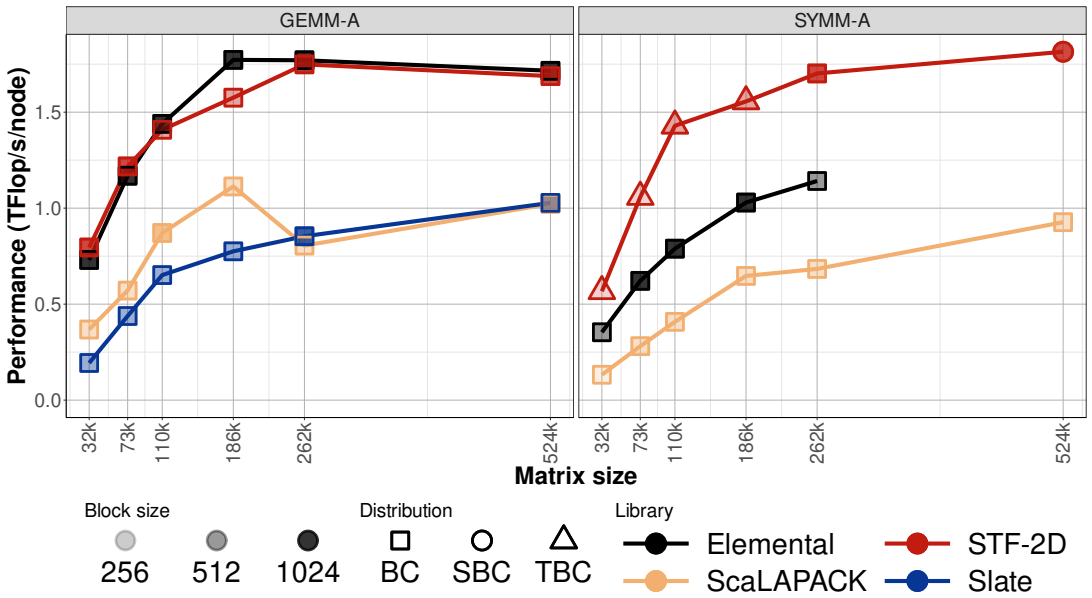


Figure 2.16 – Per-node GEMM (left) and SYMM (right) performance of the proposed STF design [5] compared with state-of-the-art libraries.

performance results of the entire MDS using randomized linear embedding, as it was the initial motivation behind the work we did on SYMM.

For this Section, all results have been obtained on the Jean Zay supercomputer. Jean Zay is a HPE SGI 8600 machine with an Omni-Path 100 Gb/s interconnect. Each node has 192 GB of memory and is composed of two 20 cores Cascade Lake 6248 @ 2.5 GHz processors (40 cores per node). Intel MKL v. 19.0.4 provides the implementation of single-core kernels.

2.8.1 Performance of RSVD-MDS

In [7], it was necessary to trade off performance, with 2D BC GEMM, with memory, with 2D BC SYMM, during the dominant steps (denoted MM1 and MM2 steps in Sections 1.4.2 and 2.2) of the randomized linear embeddings algorithms. The central question raised in Section 2.6 was whether we could store only half of the symmetric matrix through SYMM while achieving a performance on par with that of GEMM.

Table 2.2 – Samples names, matrix size m, number of nodes P (and of MPI processes), parameters (p,q) for 2D BC mappings, and parameter c for TBC mapping.

Sample name	m	P	(p,q)	c
S1	99,594	1	(1,1)	1
S2	270,983	6	(3,2)	2
S3	426,548	30	(6,5)	5
S4	616,644	56	(8,7)	7
S5	1,043,192	132	(12,11)	11

As discussed in Section 1.1.2, MDS computes a so-called Gram matrix G from an input matrix representing dissimilarities between pairs of items. In the context of our metabarcoding target application, items are diatoms collected in Lake Geneva and dissimilarities between them are their genetic distances. The dataset used as input for the MDS, fully described in [7], is a $10^6 \times 10^6$ matrix of genetic distances between sequences. We may consider either part of the data (*S1*, *S2*, *S3*, *S4*), leading to a matrix of reduced dimension, of the whole data set (*S5*). Table 2.2 presents the matrix size (m) and parallel setup associated with each sample. The whole MDS algorithm consists of the computation of the Gram matrix G followed by an RSVD (which includes `MM1` and `MM2` steps). All the tests of this section are performed in single precision and a number $n = 1,000$ of columns for B and C, consistently with [7].

Figure 2.17 illustrates the impact on performance of the present study. The original code from [7] had been designed following the programming model of [131, 4] in which task mapping was inferred from the data mapping of the RW blocks. This means that it relied on a C-stationary scheme, in which case the optimum GEMM mapping is 1D BC (denoted (0) in Figure 2.17) as it is both an A- and a C- stationary variant. The application of the new programming model from [5] allows us to employ a 2D BC A-stationary variant (l. 1 of the table and denoted (1) in Figure 2.17 after this line number). The significant improvement shows the interest of the new programming model when dealing with a task-based approach. Figure 2.17 furthermore shows that it is possible to store only half of the symmetric matrix through a TBC SYMM (l. 4 in Table 2.1 and denoted (4) in Figure 2.17) while achieving a performance competitive with (2D BC) GEMM, which positively answers the question that originally motivated this work. As this observation applies to both `MM1` and `MM2` matrix multiplication steps and since they altogether dominate the RSVD algorithm, this directly translates into a significant improvement for the entire RSVD.

As recalled above, MDS requires to compute the Gram matrix G before applying the RSVD

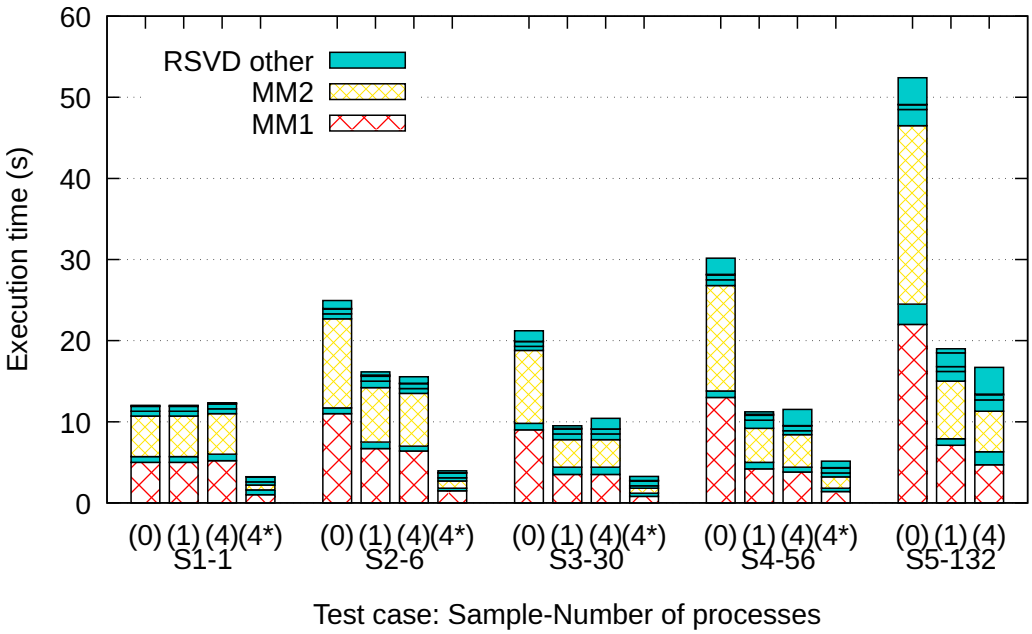


Figure 2.17 – Execution time (s) of the RSVD with $n = 1,000$. `MM1` and `MM2` are assessed with methods (1) 2D BC GEMM and (4) TBC SYMM, numbered and named after Table 2.1, in the CPU-only case. Method (0) furthermore denotes 1D BC GEMM. Method (4*) denotes TBC SYMM with GPUs turned on. Execution of (4*) on 132 nodes was not possible because of a *Quality of Service* (QoS) limitation on Jean Zay supercomputer (no more than 512 GPUs per job). Five test cases are assessed, ranging from *S1* on 1 node (denoted *S1*-1) to *S5* on 132 nodes (denoted *S5*-132).

itself. We also redesigned this step, which is mainly a reduction, using the `DIST_REDUX` access mode introduced in [5]. We have not reported detailed figures on the matter, however, the execution time of the entire RSVD-MDS algorithm (Gram computation and RSVD altogether) on the whole dataset (*S5*) using 132 nodes (5,280 CPU cores) has been reduced from 70 seconds, with the original code of [7] (denoted (0) here), to 25 seconds, with TBC SYMM (denoted (4) here) together with the new Gram step design, while using about half the memory.

We complete the study with the illustration of the capability of task-based codes to exploit heterogeneous architectures. Without any change in the code (other than providing the CUDA cuBLAS kernels of single-GPU kernels), the runtime system may execute tasks on CPU or GPU [10]. A subset of Jean Zay nodes have the exact same characteristics as described above in CPU-only case but are furthermore enhanced with four NVIDIA Tesla V100 SXM2 GPUs (32 GB). CUDA v. 10.1.2 is used. Bars denoted (4*) [5] in Figure 2.17 correspond to the execution of the RSVD with GPUs enabled, relying on TBC SYMM for the matrix multiplication. The results show a considerable improvement over the CPU-only case in spite of the relatively low number of columns ($n = 1,000$ only) of B and C, a typical set up for the application.

2.8.2 Comparison of the performance of RsEVD-MDS and RSVD-MDS

In Section 2.8.2 we wanted to build upon the results of [7] with RSVD-MDS and we showed that the model we presented in Section 2.7 did in fact allow for significant performance improvement to the SYMM operation, and that it can be used in place of GEMM for computing

5. We do not present GPU results for the largest test case as the quality of service rules of the machine did not allow for the use of that many GPU.

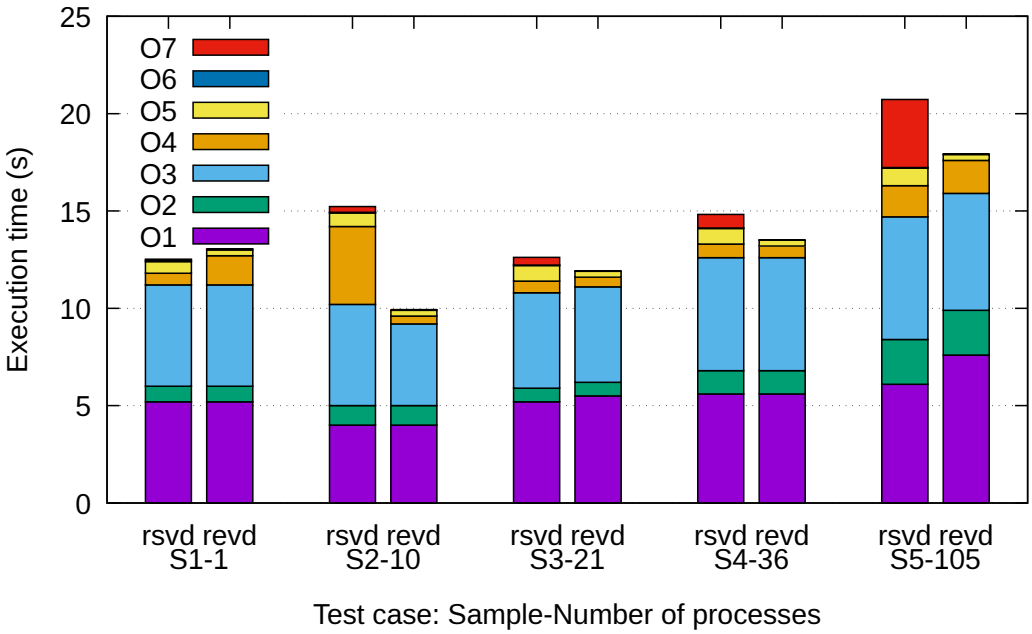


Figure 2.18 – Execution time (s) of RSVD and RsEVD with $n = 1,000$ using a SBC distribution.

Table 2.3 – Equivalent operations between RsEVD and RSVD.

Operation	RSVD	RsEVD	Nomenclature
$O1$	$Y = G\Omega$	$Y = G\Omega$	MM1
$O2$	$Y = Q_Y R_Y$	$Y = Q_Y R_Y$	QR1,Q1
$O3$	$C = GQ_Y$	$C = GQ_Y$	MM2
$O4$	$C = Q_C R_C$	$C = Q_Y^T C$	QR2,Q2 - MM3
$O5$	$SVD(R_C)$	$EVD(C)$	SVD - EVD
$O6$	$U = Q_Y V_{R_C}$	$U = Q_Y U_C$	MM3 - MM4
$O7$	$V^T = U_{R_C}^T Q_C^T$	-	MM4

MM1 and MM2 in the context of both the RSVD and the RsEVD. We showed in Section 2.3 that using the RSVD to compute the MDS on a matrix on which we do not know the rank could lead to numerical inaccuracies because of a possible loss of symmetry. As such, we might want to use the RsEVD instead as this method is not subject to the same problem. In this section, we propose to look at the performance of the two methods in the distributed case, to show that we can use the RsEVD instead of the RSVD without loss of performance. In Figure 2.18, we show the performance of both these methods using the same datasets presented in Table 2.2. The different operations 01 to 07 are presented in Table 2.3. The results obtained in Figure 2.18 were done using a SBC distribution. It shows that we achieve similar performance using either methods, although the RsEVD has a slight edge in the larger cases, in part due to the fact that the RSVD had to perform an additional operation (07). We can also see that as expected, the majority of the computing time still comes from the matrix products MM1 and MM2.

Finally, we present the actual results of this RsEVD-MDS approach. Figure 2.19 presents the heatmaps we computed of the S5 sample. For reference, the ones obtained using the RSVD-MDS of [7] are presented in Figure 1.13 (page 26). A visual study seems to indicate that these point clouds are the same. This would mean that even though the analysis we made in Section 2.3 indicated that the RSVD could lead to an error during the reconstruction of the point cloud, it does not necessarily mean that the RSVD necessarily gives wrong results. In this section, we limit ourselves to this visual examination of the point clouds. Further comparison of these point clouds is discussed in Chapter 3 and more precisely Section 3.8 were we are able to confirm in a more rigorous way whether or not these point clouds are the Same. Figure 2.20 presents the singular values obtained with RsEVD (in red) to the ones from RSVD (in blue). We can see as expected from [64] as well as our experiments from Section 2.3 that the singular values coincide up to a point (around rank 200) at which point the values obtained from RsEVD starts plummeting.

2.9 Conclusion

We refer the reader to Section 2.3.3 for the conclusion on the numerical study of the randomized linear embedding algorithms and now focus on the conclusions for our improvement of distributed-memory matrix multiplication.

We experimentally confirmed that reference distributed-memory libraries achieve a lower performance with SYMM than with GEMM. We showed that an efficient design of the communication schemes can significantly alleviate this gap. Still, we showed that part of the gap is explained by a lower AI of 2D BC SYMM compared to 2D GEMM (by a factor of 2). We considered two alternative data distributions, SBC and TBC. SBC is a direct adaptation to the matrix multiplication case of a study of the Cholesky decomposition [14]. TBC is a distributed-memory (and even a parallel) adaptation of the ideas behind TBS [15], a sequential out-of-core algorithm. We proved that SBC and TBC improve the AI of SYMM by a factor of $\sqrt{2}$ and 2, respectively, thus in particular equaling that of 2D BC GEMM for the latter one. In the case where we allow SYMM to store an amount of memory equivalent to a full matrix as 2D BC GEMM does, we furthermore showed that 2.5D TBC with $s = 2$ slices achieves a higher AI than 2D BC GEMM by a factor of $\sqrt{2}$. Our experimental study showed that the improvement of the AI translates into a compelling performance enhancement, up to the point of roughly matching GEMM performance. However, the highest AI does not always translate into the best performance.

The resulting code [6] has been integrated in the randomized linear embedding algorithms used in the MDS framework of Section 1.6 in place of the two dense matrix multiplications rep-

6. Source code and instructions available at https://doi.org/10.5281/zenodo.7657176

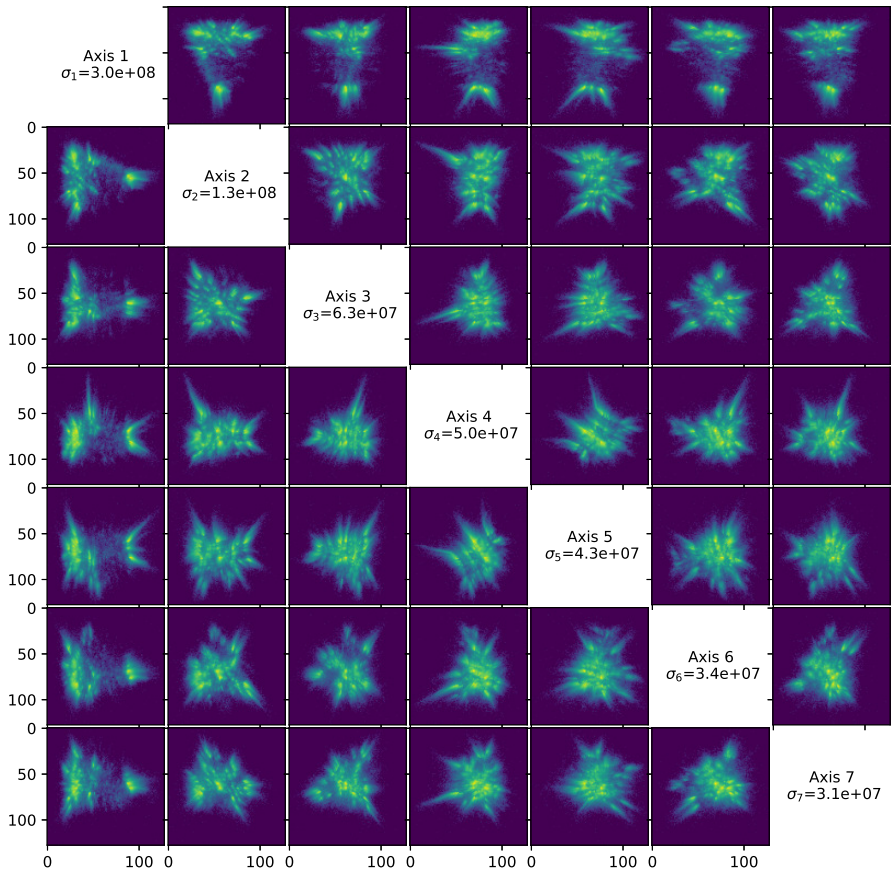


Figure 2.19 – Heatmap of the full L1-L10 point cloud (sample S5), obtained from a RsEVD-MDS of the dataset at rank 1000. Each heatmap corresponds to the representation of the two dimensions in the diagonal block. For instance, the third heatmap of the first column represents the first axis along the x-axis and the fourth axis along the y-axis.

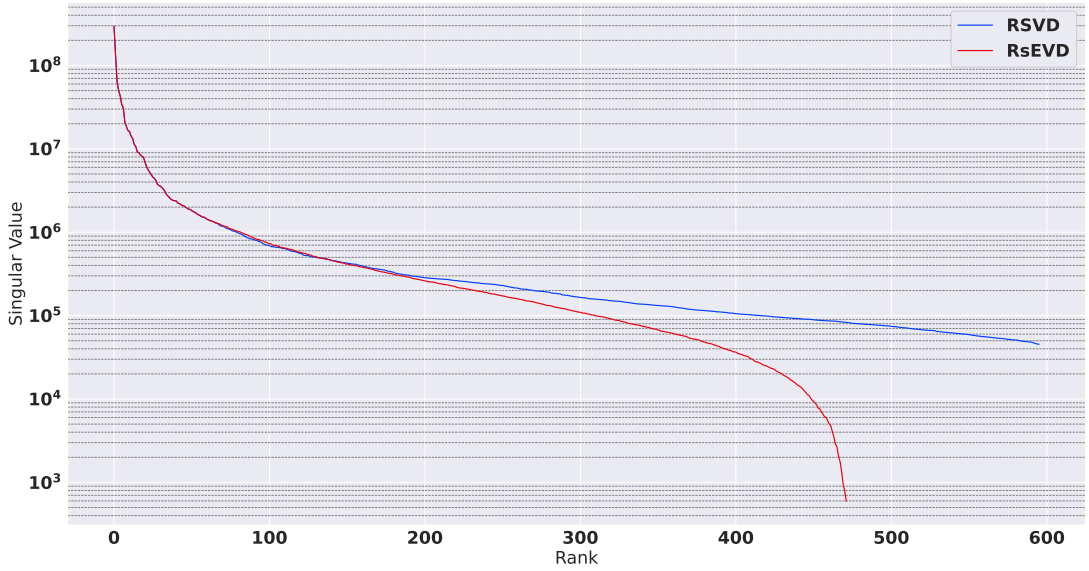


Figure 2.20 – Singular values of the L1-L10 dataset (sample S5), obtained from RsEVD-MDS (in red) of rank 1000 compared to the singular values found using RSVD-MDS (in blue) that have already been presented in Figure 1.14. Values associated with a negative eigenvalue have been filtered out as they are not part of the axes kept for building the associated point cloud.

resenting the main computational step of the algorithm. While one had to trade-off [7] between performance, with GEMM, or memory, with SYMM, we showed that, altogether, the proposed STF design with the new TBC distribution now achieves a performance competitive with GEMM. This study also showed that algorithms involving very irregular data and task distributions can now be implemented with a code easy to write, read and maintain thanks to the latest developments on the scalability of the STF model [5], while ensuring a competitive performance.

As a result of this work, we now have access to a versatile high-performance MDS library that provides an implementation of both RSVD-MDS and RsEVD-MDS in a distributed-memory context. These algorithms are based on a state-of-the-art STF implementation of SYMM which makes no compromise between performance and memory footprint.

Part II

MDS on a reduced sample of the input distance matrix

Chapter 3

Comparison of point clouds resulting from MDS

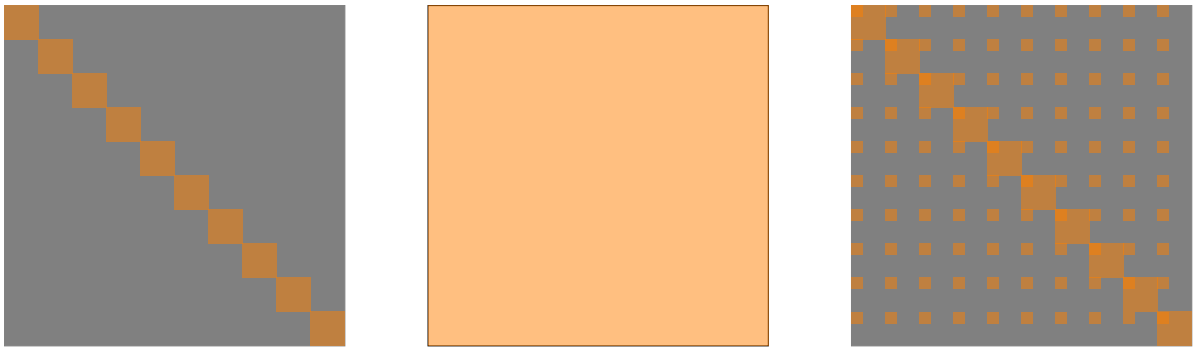
3.1 Introduction

In this chapter we will explore the comparison of point clouds resulting from MDS. Our aim is to apply these methods to the S5 ($L_1,\ldots,L_{10}$) dataset from Section 1.8.4 (p. 21) in order to compare the diatoms from Lake Geneva collected at different monthly intervals. A first possibility is to compute each L_i point cloud independently. This is how they had been computed in Figure 1.15 (p. 28). At the other end of the spectrum, another possibility is to rely on an HPC software stack such as discussed in the first part of this thesis to compute the full sample $S5$ ($L_1,\ldots,L_{10}$) and extract the L_i. Do these methods produce similar results?

The main problem with comparing point clouds from MDS is that MDS can be arbitrarily translated, rotated, and reflected. This is related to the well known problem of Procrustes analysis [117]. A first thought might thus be that the point clouds shown in 1.15 (p. 28) obtained by independent computation have both already been centered and therefore only remain to be rotated (and/or reflected). A quick look at the figure might even reassure us in this respect. Because the human brain can easily rotate (and/or reflect) a 2D image, "human" post-processing might be satisfying whereas "numerical" post-processing might be overkill. Correct?

We invite the reader to play this rotation/reflection game with the Long Reads A, B, C and D bacteria dataset (see images in Figure 1.9, p. 23) to align A, B, C and D with each other, and then with the S5 diatom dataset (see images in Figure 1.15, p. 28) to align L1, ..., L10 to each other. What are your conclusions?

Here is a spoiler. Such a "human" 2D rotation/reflection visual post-processing is only possible for the Long Reads dataset. As discussed in Section 1.8.2, the respective Long Reads A, B, C, and D can be viewed as randomized samples from the same larger Long Reads ABCD sample. Consequently, they are assumed to represent the same population and therefore have the same MDS point cloud. Since MDS is defined up to a unitary transformation, all that remains is to perform a 2D rotation/reflection to align them. In fact, we will see in Chapter 5 that it is even sufficient to consider only the reflections along the first two vectors e_1 and e_2 of the canonical basis of the plane (see solution in Figure 5.23, p. 150), which is related to Issue 4 discussed in this chapter. On the other hand, the L1, ..., L10 represent diatoms from Lake Geneva collected at different monthly intervals. We can therefore assume that they are related



(a) Block-diagonal pattern (b) Full pattern (c) Extended block-diagonal pattern

Figure 3.1 – High-level data access pattern to the dissimilarity matrix D (in grey) when processing point clouds independently (a), altogether (b), or almost independently but augmented to ensure a robust alignment (c). Only orange blocks need to be accessed.

to each other, but we cannot *a priori* assume that they represent the same population, as it can evolve over time. The *a posteriori* conclusion of this chapter is that Figure 1.15, p. 28) is not well suited to comparing L1, ..., L10 with each other. The development of this chapter will provide a more appropriate visualisation for comparing them (the resulting visualisation can be seen in Figure 3.30, p. 93 in Section 3.8.3).

Here is a reading in terms of data access. When computing the point clouds to be compared (such as the L_i samples) independently, it is sufficient to access a block-diagonal pattern such as shown in Figure 3.1a. This method was used to generate Figure 1.15 (p. 28). At the other end of the spectrum, we can construct the full sample (e.g. the entire S5 (L1-L10) sample) and extract the corresponding point clouds (such as the L_i samples) from it. We can do this by relying on the HPC MDS from Chapter 2. However, it requires access to the full reference dissimilarity matrix, as shown in Figure 3.1b. We show in this chapter that an intermediate solution, which only needs to access a small portion of the reference dissimilarity matrix out of the diagonal blocks, as shown in Figure 3.1c, can provide a robust solution too. We will use such a method to generate Figure 3.30, p. 93).

There is also a deeper motivation. When we consider a full sample consisting of several related samples that we want to compare, we may want to preserve the structure of each of the samples. For example, if we consider the L1, ..., L10 samples, they are all samples of diatoms, but they were collected at different monthly intervals. We may therefore want to relate them *while* preserving their respective structure. The alignment method developed in this chapter does just that.

The proposed method relies on the same ingredients as MDS based on a divide-and-conquer paradigm [130, 76]. However, while such divide-and-conquer methods are usually employed for efficiently building a large MDS, on the contrary, our aim is to align point clouds for the purpose of comparing them.

In addition to the visualisation problem, the comparison of point clouds may be captured more synthetically through a measure of how far apart they are. This is also a well-known problem. The Hausdorff distance [70] and its variants are well established metrics to answer this question. We combine the alignment method we develop together with the application of variants of the Hausdorff distance to obtain a measure of MDS point clouds.

We first elaborate on distance between point clouds in Section 3.2. We present related work on the Hausdorff distance between point clouds [70] and discuss possible variations. We then

tackle the alignment problem. Although this is a well known problem, we will first recall the alignment effect of MDS, which is actually the same as that of principal component analysis (PCA), in Section 3.3. Since these methods align point clouds on principal axes, they implicitly perform some alignment. However, four potential issues prevent one from guaranteeing that two related point clouds will be correctly aligned directly by PCA or MDS: **Axes permutation (Issue 1), Rotation (Issue 2), Translation (Issue 3) and Reflections around principal axes (Issue 4)**. Sections 3.4, 3.5 and 3.6 deal with the alignment of point clouds on orthogonal Procrustes analysis [117, 65, 60, 90] and General Procrustes Analysis [58] in Section 3.7. We apply all these methods to the study of the $L_1, \ldots, L_{10}$ dataset in Section 3.8. This leads us to a discussion on the dimensionality of our data in Section 3.9.

3.2 Distance between point clouds

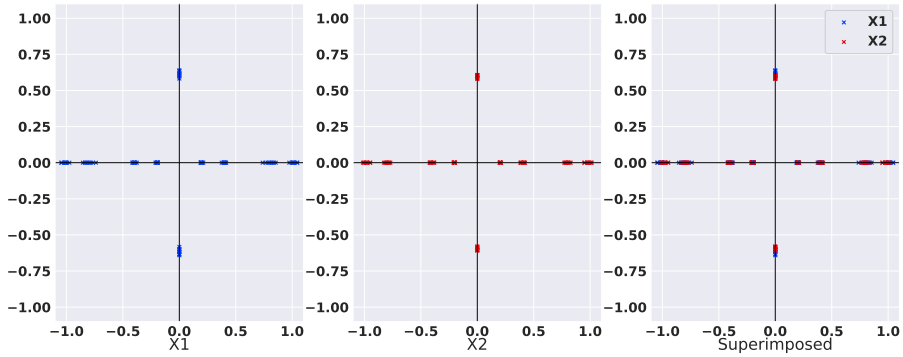


Figure 3.2 – Measuring the distance between two point clouds **X1** and **X2**.

In this section, we are going to go over different means of comparing point clouds. First, we are going to define some terminology, using the simplest case possible. In this case, both point clouds are aligned, have the same number of points, and we know of the correspondence between the points. Instinctively, we want to define aligned point clouds as point clouds that can be superimposed as best as possible, as illustrated in Figure 3.2. Let us consider two point clouds **X1** and **X2**. As long as there are the same number of points in both clouds and that there is a correspondence between the rows of the coordinate matrices of **X1** and **X2** *i.e.* the point represented in a row of **X1** is aligned to the point represented in the same row in **X2**, calculating a distance between **X1** and **X2** boils down to computing the Frobenius Norm $\|X1 - X2\|_F$.

All of the aforementioned properties do not necessarily hold in practice, sometimes none do. In this chapter, we discuss all of those cases, and show the steps needed to deal with these constraints. In sections 3.2.1, 3.2.2 and 3.2.3 we present the Hausdorff distance, a metric used to compare shapes, that can still compare point clouds when we have no correspondence between points. In fact, the Hausdorff distance does not even require both point clouds to have the same number of points.

In Section 3.3, we discuss all the sources of non alignment between point clouds, followed in Section 3.4, by a presentation of the existing methods of aligning point clouds. Using all of these methods, we present in Section 3.5 methods for aligning point clouds arising from MDS.

3.2.1 Hausdorff distance

A classical metric to compare point clouds is the Hausdorff distance [70]. The Hausdorff distance has been defined betwen compact spaces, and as a (finite) point cloud is a compact space (it is closed and bounded), Hausdorff distance can be used for computing distances between point clouds. To define the Hausdorff distance H between two sets of points A and B, we first consider the distance between a point a in A and the set B as

$$d(a, B) = \min_{b \in B} \|a - b\| \tag{3.1}$$

here $\| \quad \|$ is the euclidean distance. From this definition, we consider the directed Hausdorff distance from set A to set B to be equal to the distance from the point of set A that is the furthest away from any point of set B, it is noted

$$h(A, B) = \max_{a \in A} d(a, B) \tag{3.2}$$

Finally, we define the Hausdorff distance H between A and B as the maximum between both directed distance. The Hausdorff distance writes

$$H(A, B) = \max(h(A, B), h(B, A)) \tag{3.3}$$

If the Hausdorff distance between A and B is δ then for any point a in A there exist at least one point b in B such that $\|a - b\| \leq \delta$ (the same is true from B to A as well). We can see that this distance will not be impacted by the number of points, neither does it require a correspondence between the points of both sets.

3.2.2 Variation of the Hausdorff distance

The Hausdorff distance is known to suffer from an outlier issue [40], in the sense that the distance between two point clouds may be completely skewed by the presence of a single point far away from the other ones, even if the two point clouds are otherwise identical. This is a problem for many applications. To counteract the potential impact of these outliers, we rely on existing work that solves this issue by using a variations of the Hausdorff distance. There exist many variations of the Hausdorff distance [40, 70, 88, 103], most of them driven by the idea of minimizing the outliers problem. They are often motivated with a precise application in mind and use extra parameters to take advantage of known specificities of the data used.

3.2.2.1 Modified Hausdorff distance

As mentioned in the previous section, a lot of the variations of the Hausdorff distance found in the literature are dependant on extra parameters, which are often chosen with some specificities of the data in mind. For this reason, we decided to only consider the Modified Hausdorff distance (MHD) [40] out of those methods, as it doesn't require any extra parameter. The MHD is obtained by taking the definition of the classical Hausdorff distance and replacing the definition of the directed distance with

$$h_{MHD}(A, B) = \frac{1}{N_A} \sum_{a \in A} d(a, B) \tag{3.4}$$

where N_A is the number of points of A. Here, by taking into account all the points of the clouds, the effect of outliers is dampened. We have to note that this method is not a proper distance, as the triangle inequality is not respected.

3.2.2.2 Squared Modified Hausdorff distance

To expand on the MHD, we also propose to consider another variation of the Hausdorff distance that to the best of our knowledge is not present in the literature. We chose this metric because it has a similar behavior to the Frobenius distance under certain circumstances. This distance, that we will call the squared modified Hausdorff distance (denoted H_2) is defined by using the following directed distance

$$h_2(A,B) = \sqrt{\frac{1}{N_A}\sum_{a\in A} d(a,B)^2} \tag{3.5}$$

In order to show the link with the Frobenius norm, let us consider two point clouds A and B. In the case where A and B have a correspondence between their points and are aligned (meaning A and B have the same number of points and the closest point to the i^{th} point of A is the i^{th} point of B and reciprocally), then we can show that the squared modified Hausdorff distance and the Frobenius distance will be the same up to a factor $\sqrt{N}$, where N is the number of points. We can write this condition more formally in Equation 3.6.

$$\forall i, \forall j \neq i, \|a_i - b_i\| < \|a_i - b_j\| \tag{3.6}$$

This leads us to define lemma 1.

Lemma 1. *If A and B are two point clouds that satisfy 3.6, then $H_2(A,B) = \frac{1}{\sqrt{N_A}}\|A-B\|_F$*

Proof. Let us consider $A \in \mathbb{R}^{m\times k}$ and $B \in \mathbb{R}^{m\times k}$, representing two point clouds composed of m points in k dimensions that satisfy 3.6 and let us compute $H_2(A,B) = max(h_2(A,B), h_2(B,A))$. The directed distance writes

$$h_2(A,B) = \sqrt{\frac{1}{m}\sum_{a\in A} d(a,B)^2} \tag{3.7}$$

$$h_2(A,B) = \sqrt{\frac{1}{m}\sum_{i=1}^{m} d(a_i,B)^2} \tag{3.8}$$

$$h_2(A,B) = \sqrt{\frac{1}{m}\sum_{i=1}^{m} \min_{b\in B} \|a_i - b\|^2} \tag{3.9}$$

By 3.6, we know that the b that satisfy this min is b_i, which gives us

$$h_2(A,B) = \sqrt{\frac{1}{m}\sum_{i=1}^{m} \|a_i - b_i\|^2} \tag{3.10}$$

$$h_2(A,B) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k} (a_{ij} - b_{ij})^2} \tag{3.11}$$

$$h_2(A,B) = \frac{1}{\sqrt{m}}\|A-B\|_F \tag{3.12}$$

We can show in the same way that $h_2(B,A) = h_2(A,B)$ and therefore we have $H_2(A,B) = \frac{1}{\sqrt{m}}\|A-B\|_F$. □

3.2.3 Relative Hausdorff distance

When studying distances between point cloud, we want to be able to use a reference to have a relative distance. We can here make the analogy to the classical linear algebra problem of solving $Ax = b$. To evaluate the quality of a solution, one would compare it to the norm of the right hand side b to have a relative error $\frac{\|Ax-b\|_F}{\|b\|_F}$. The problem of scaling the Hausdorff distance to a reference has been mentioned in the literature. For instance, in [134] the authors use a reference length to scale the data and in [77] they normalize their point clouds to have their coordinates in the range of [-1,1].

To construct such a reference, we propose to take a slightly different approach and we define the Hausdorff diameter of a centered point cloud.

We define, for a point cloud A and a variation of the Hausdorff distance d, the corresponding diameter as

$$d(A) = d(A,O) \tag{3.13}$$

with O being the point cloud composed of a single point of zero coordinates in all dimensions (i.e., the centre of the point cloud).

We can now show that whether we use the Hausdorff distance, the MHD of the H_2, equation 3.13 respect the following properties, described in lemma 2.

Lemma 2. *Let $A \in \mathbb{R}^{m\times k}$ representing a point cloud composed of m points in k dimensions. For any d (among H, MHD, H_2) $d(.)$ have the folowing properties :*

1. *Absolute homogeneity - $d(\lambda A) = |\lambda| d(A)$*
2. *Positive definiteness - $d(A) = 0 \iff A = 0$*

Proof. Let A be a point cloud composed of m points in k dimensions and O is a point cloud composed of only one point (the centre).

For the Hausdorff distance:

By using the definition of $d(A)$, and the definition of the directed oriented distance we have

$$d(A) = max(h(A,O), h(O,A)) = max(\max_{a\in A}\|a\|, \min_{a\in A}\|a\|) = \max_{a\in A}\|a\| \tag{3.14}$$

Because of the properties of the Euclidean norm and the max function, the absolute homogeneity property is obvious. Concerning the positive definiteness - $H(A) = 0 \iff A = 0$
Since we know that $H(A) = \max_{a\in A}\|a\|$, then all points in A is equal to the centre O, and then, $A = O$.
For the MHD:

First, we detail the two directed distances

$$h_{MHD}(A,O) = \frac{1}{N_A}\sum_{a\in A}\|a\| \tag{3.15}$$

$$h_{MHD}(O,A) = \min_{a\in A}\|a\| \tag{3.16}$$

It is easy to see that $h_{MHD}(A,O) > h_{MHD}(0,A)$ and then

$$MHD(A) = max(h_{MHD}(A,O), h_{MHD}(O,A)) = \frac{1}{N_A}\sum_{a\in A}\|a\|.$$

The absolute homogeneity properly - $MHD(\lambda A) = |\lambda| MHD(A)$ is evident as

$$MHD(\lambda A) = \frac{1}{N_A}\sum_{a\in A}\|\lambda a\| = \frac{1}{N_A}\sum_{a\in A}|\lambda|\|a\| = \frac{|\lambda|}{N_A}\sum_{a\in A}\|a\| = |\lambda| MHD(A)$$

For the positive definiteness - $MHD(A) = 0 \iff A = 0$, we show this by using exaclty the same arguments that we used for the Hausdorff distance.

For the H_2:

The proof is similar to the MHD one since we have

$$H_2(A) = max(h_2(A,O), h_2(O,A)) = \frac{1}{\sqrt{N_A}}\sum_{a\in A}\|a\|^2.$$

1. Absolute homogeneity - $H_2(\lambda A) = |\lambda| H_2(A)$
We have

$$H_2(\lambda A) = \sqrt{\frac{1}{N_A}\sum_{a\in A} d(\lambda a, 0_k)^2} = \sqrt{\frac{1}{N_A}\sum_{a\in A}\|\lambda a\|^2} = \sqrt{\frac{1}{N_A}\sum_{a\in A}|\lambda|^2\|a\|^2}.$$

As a consequence, we obtain:

$$H_2(\lambda A) = |\lambda|\sqrt{\frac{1}{N_A}\sum_{a\in A}\|a\|^2} = |\lambda| H_2(A).$$

2. Positive definiteness - $H_2(A) = 0 \iff A = 0_k$
The positive definiteness can again be shown using the same method as in the previous two cases. □

Using our various Hausdorff diameter, we introduce this relative distance

$$d_r(A,B) = \frac{d(A,B)}{max(d(A), d(B))} \tag{3.17}$$

to evaluate the distance between the two point clouds A and B. We chose this denominator in Equation 3.17 as we wanted our relative distance to be symmetric with $d_r(A,B) = d_r(B,A)$. Replacing it with $d(A)$ would be another valid relative distance in the case we consider A to be the reference we compare B to.

3.3 Do PCA and MDS implicitly align two related point clouds?

Both PCA and MDS align point clouds on principal axes. As a consequence, they implicitly perform some alignment. If we apply PCA or MDS on two point clouds, we may expect that part of the problem of aligning them gets implicitly solved. This is partially true. However, four potential issues prevent one from guaranteeing that two related point clouds will be correctly

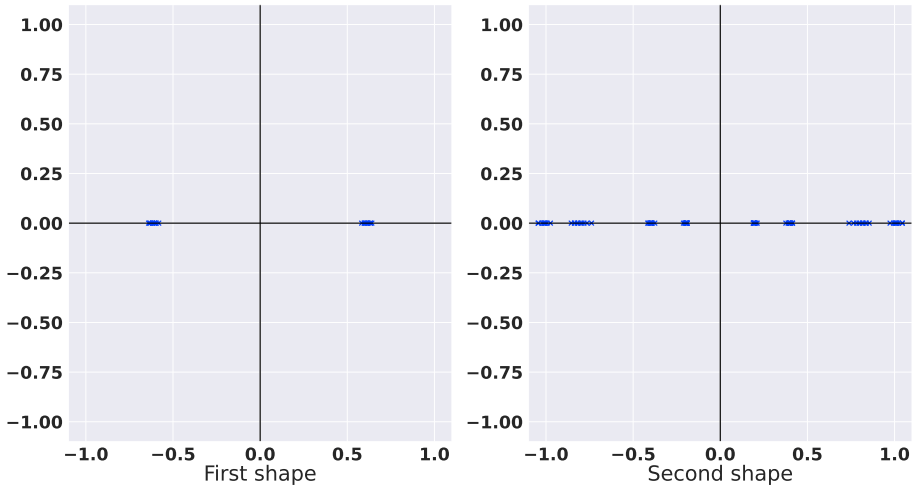


Figure 3.3 – The two basic shapes we consider to create point clouds.

aligned directly by PCA or MDS: **Axes permutation (Issue 1), Rotation (Issue 2), Translation (Issue 3) and Reflections around principal axes (Issue 4)**.

This is a well known problem and we invite the knowledgeable reader to skip this section.

We illustrate that even when working with very similar data, small effects can significantly affect the alignment of point clouds. Throughout this section, we will create point clouds by composing the two shapes shown in Figure 3.3 to illustrate our discussion. On the left, we have a point cloud composed of two clusters, located around (-0.6) and $(+0.6)$ and, on the right, we have a point cloud composed of clusters at $(-1, -0.8, -0.4, -0.2)$ and their symmetrical parts. We use them to illustrate how small variations in some parameters can have a large effect on the point cloud once we apply PCA [105, 69, 74] or MDS. Because PCA and MDS have the same alignment effect, we will limit our discussion to PCA, as it is easier to discuss in terms of point clouds only without having to resort to a distance matrix.

3.3.1 Issue 1: Axes permutation

For the first effect, we want to show that under some circumstances, two principal axes can be exchanged. Here we construct our point cloud by arranging our two shapes from Figure 3.3 to be orthogonal to each other. Our reference point cloud will be the one from Figure 3.4. In this figure, we compute the two singular values associated with the main axes. Then, we influence the singular values by modifying the density of the clusters inside this point cloud. As long as the singular values are far enough apart, like they are in Figure 3.5 we can see that the PCA of our point clouds remains the same, and the output we get are aligned and can be compared. However, in Figure 3.6 we can see that what used to be the second singular value became greater than the first one. As a result, we can see that the two point clouds are not aligned anymore. This is the worst possible case, as a visual expertise tells us that the point clouds have the same shape, yet computing a distance on these point clouds will make it appear that they are very far apart. In real data, we can expect these kinds of axes inversion to occur when we deal with two singular values that are very close to each other. Then any form of noise might cause an inversion of axes. While this can in theory be fixed at the cost of a point cloud alignment step, we have to remind that the MDS is a dimension reduction algorithm. If we combine a permutation of some axes with a projection in a lower number of axes, then the

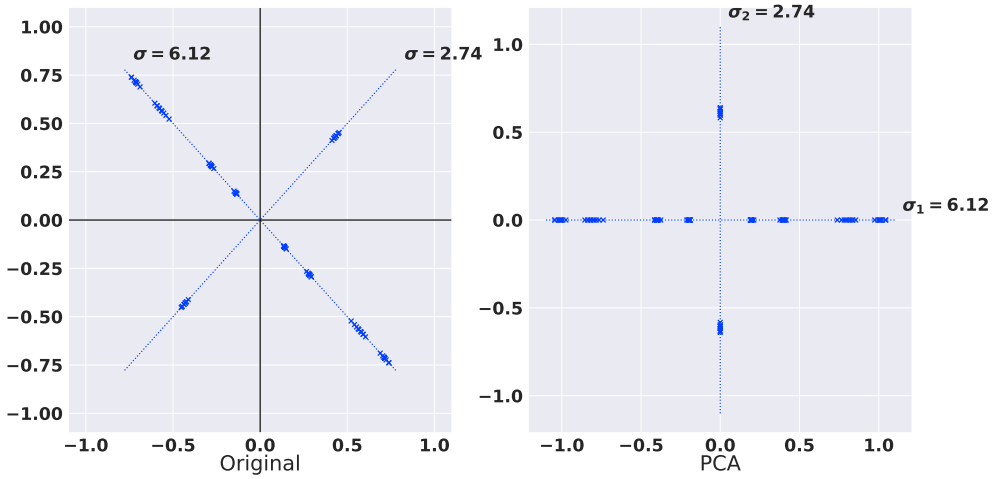


Figure 3.4 – The two shapes presented in an orthogonal way. Here we make sure that the singular values associated with each principal axis are very distinct.

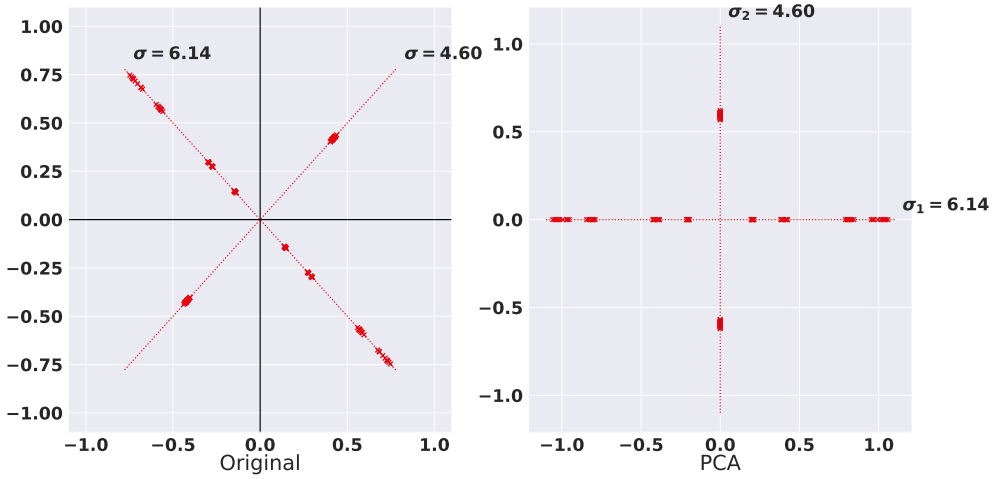


Figure 3.5 – The two shapes presented in an orthogonal way. As long as the singular values remain distincts, the PCA of the point cloud remains the same.

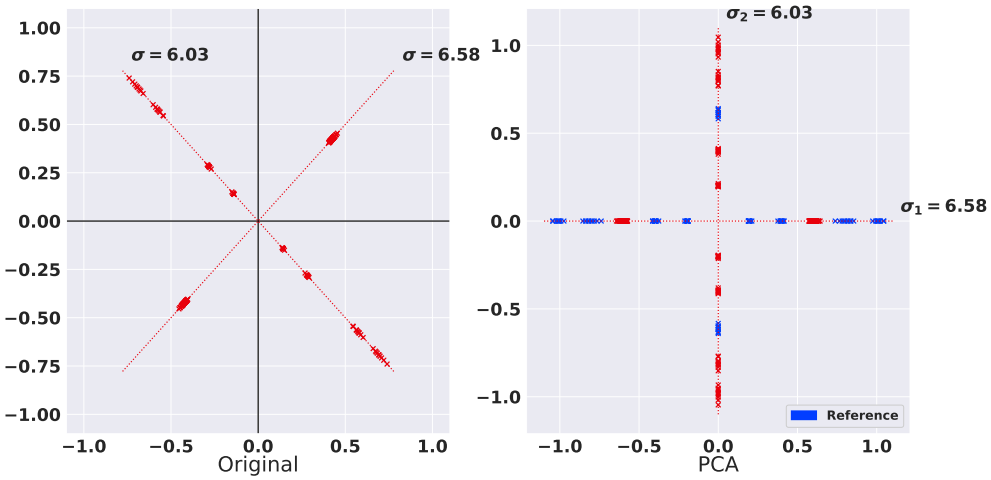


Figure 3.6 – Permutation of the axes when the singular values become closer.

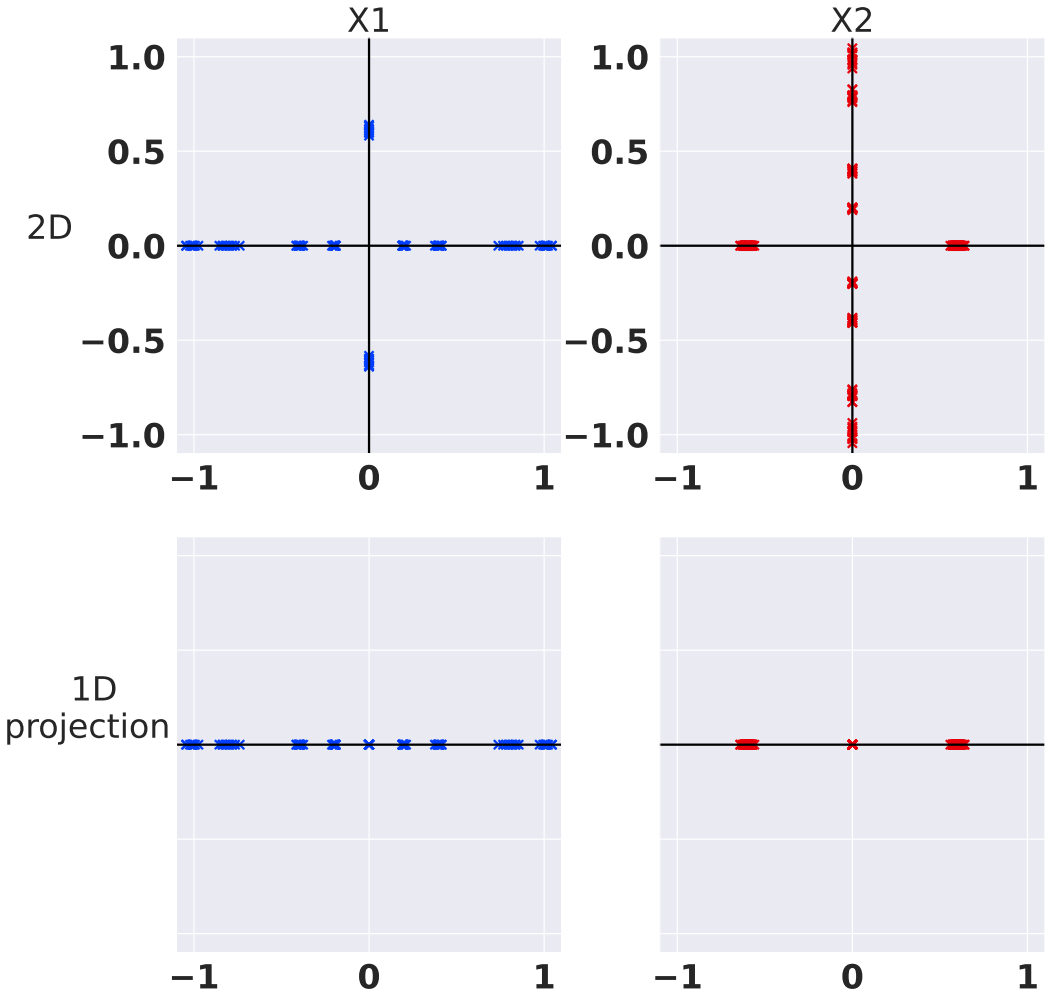


Figure 3.7 – Effect of the axes permutation on dimension reduction.

information can be definitely lost. We illustrate this possibility in Figure 3.7 where we take the point clouds we were studying before and project them onto the first axis. Here we can see that even though the two point clouds are similar and can be recognized as such in their original version, after the projection on a single axis, they are not similar at all anymore.

3.3.2 Issue 2: Rotations

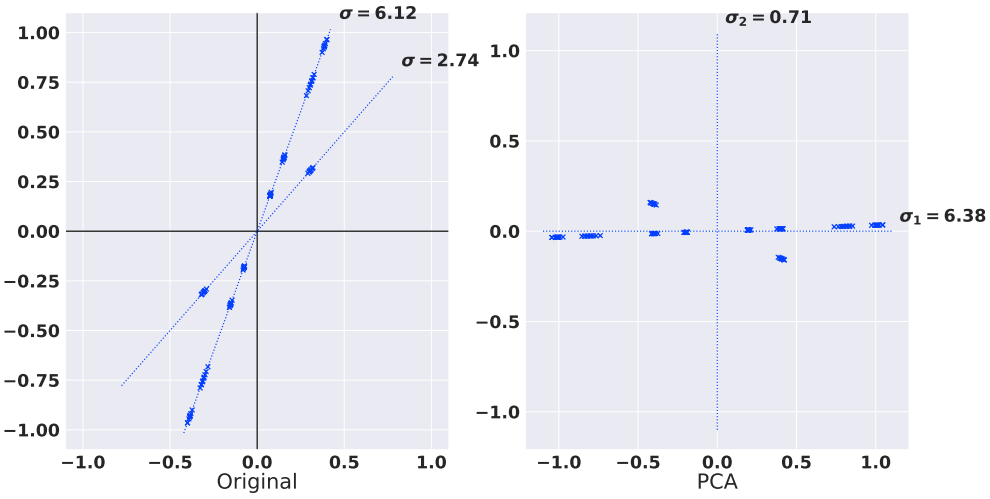


Figure 3.8 – Point cloud built form the initial shape arranged in a oblique manner.

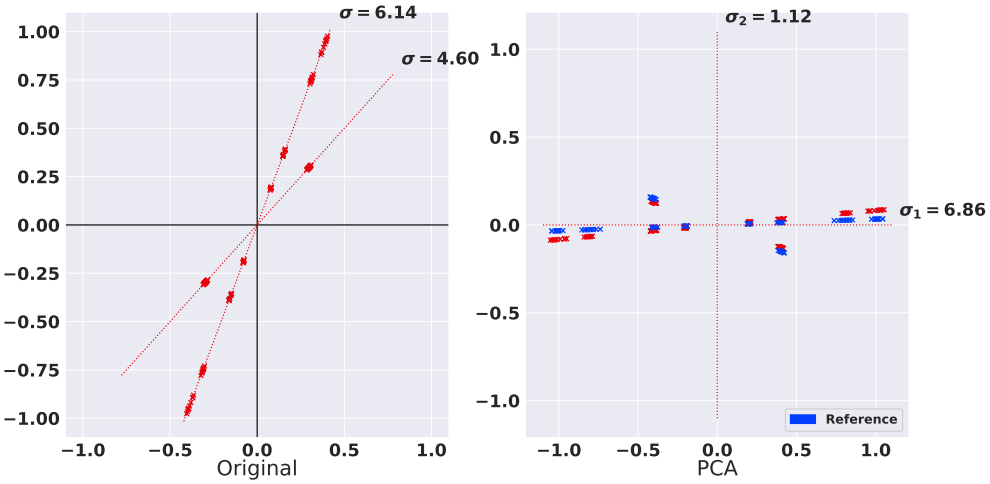


Figure 3.9 – Oblique point cloud being compared to the reference when the weight of the clusters changes.

The second effect we are tying to highlight is the fact that a slight change in the distribution of the cluster can lead to a rotation. To show this we build a point cloud from the same two shapes, except that this time instead of building an orthogonal point cloud, we arrange them in an oblique fashion. This gives us the reference shape of Figure 3.8. Here, when we change the relative weight of each axes, instead of having an inversion of the axes, we can see that there is a rotation between the two shapes. This rotation is visible in both figures 3.9 and 3.10 and

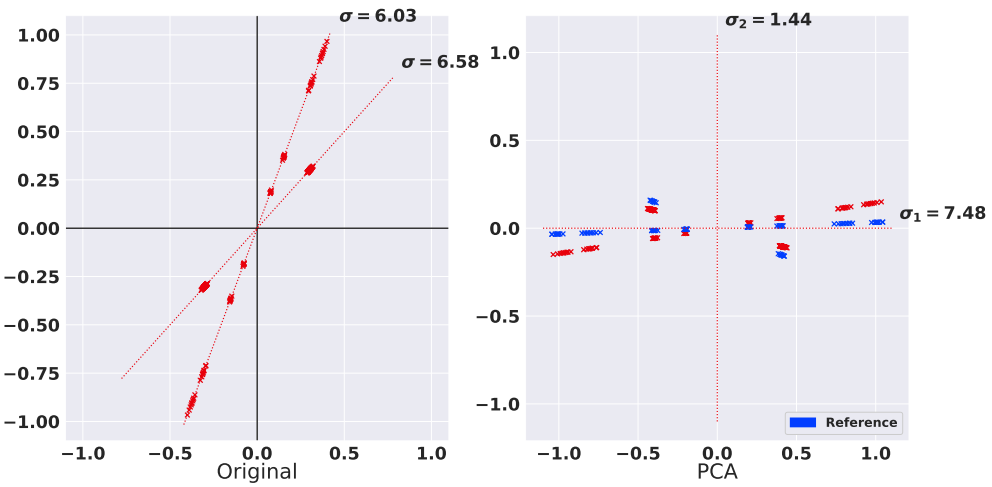


Figure 3.10 – Oblique point cloud being compared to the reference when the weight of the clusters changes.

is more pronounced in Figure 3.10. The more we make the cluster on one of the axes dense, the more it looks like it is pulled towards the principal axes, resulting in a rotation from the reference point cloud. Just like the inversion of the axes presented in Section 3.3.1 any noise in the data, or randomness in the algorithm used can cause such rotations. However in this case, the rotation while impacting the distance between point clouds, the rotation is proportional to the amount of error introduced, and a small rotation will only result in a small increase in distance.

3.3.3 Issue 3: Translation

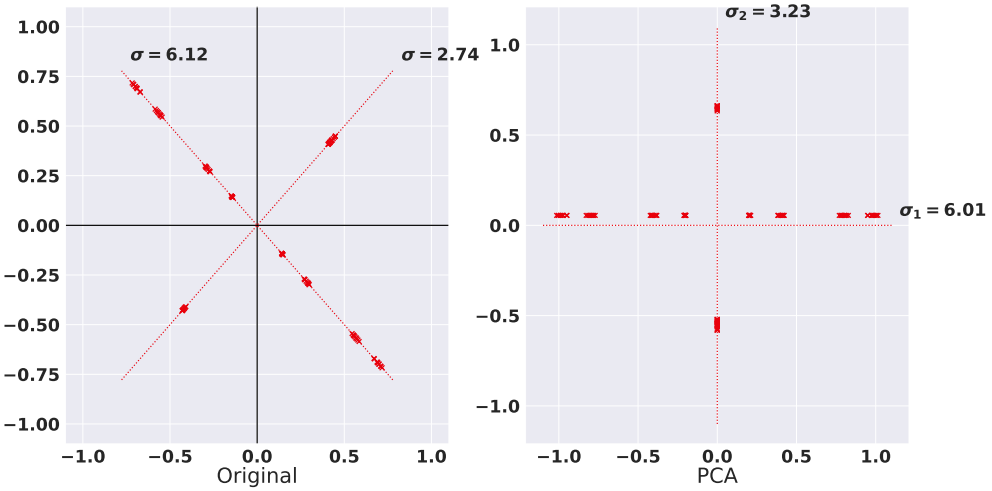


Figure 3.11 – Point cloud that is off-centered due to one cluster being larger than the other in the smaller axes.

The last transformations to consider are translations. We achieve it by having one cluster larger than its symmetric counterpart. It results in the point cloud seen in Figure 3.11 that can be seen

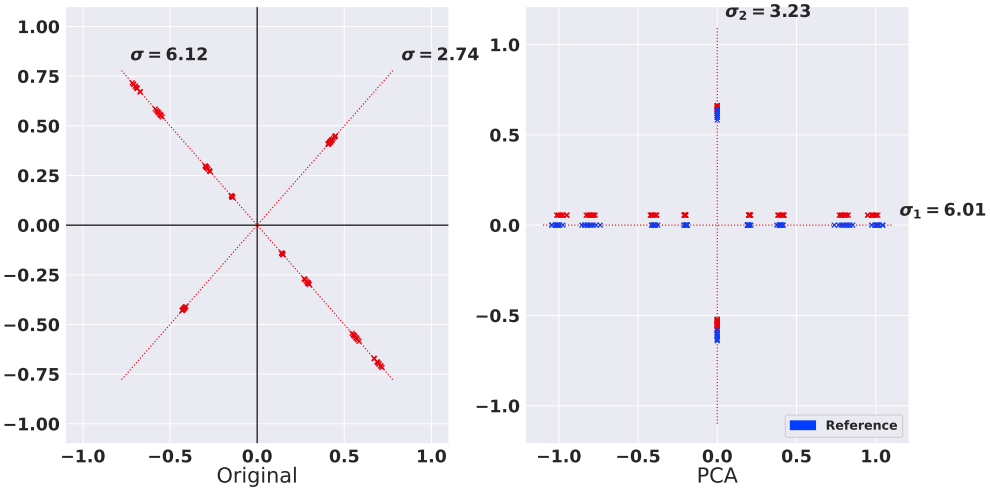


Figure 3.12 – Comparison of this off-centered point cloud compared to the reference point cloud of Figure 3.4.

in comparison to the reference point cloud of Figure 3.4 in Figure 3.12. In general we are not going to consider translations as all the point clouds we consider in the case of MDS have been centered. Still the center of the point cloud is not an absolute reference, and sometimes centering point clouds before comparing them makes sense. This will be discussed further in Section 3.5.

3.3.4 Issue 4: Reflections

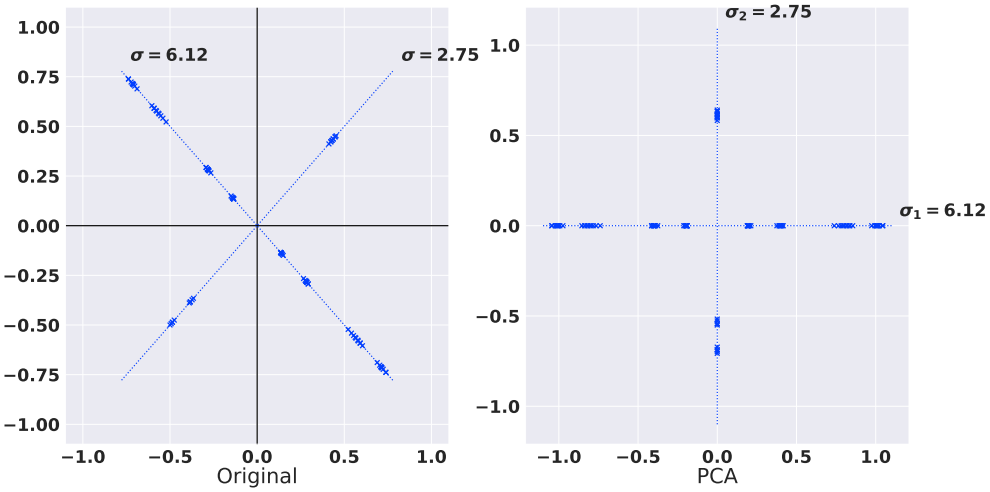


Figure 3.13 – Reference point cloud using a non-symmetric distribution on the second axes.

In this section, we present the problem of reflections around the principal axes. To illustrate it we need to slightly alter the point clouds we are working with as otherwise the point cloud will look exactly the same regardless. This new reference point cloud is presented in Figure 3.13. In this point cloud, we split one of the clusters of the second principal axis into two. We

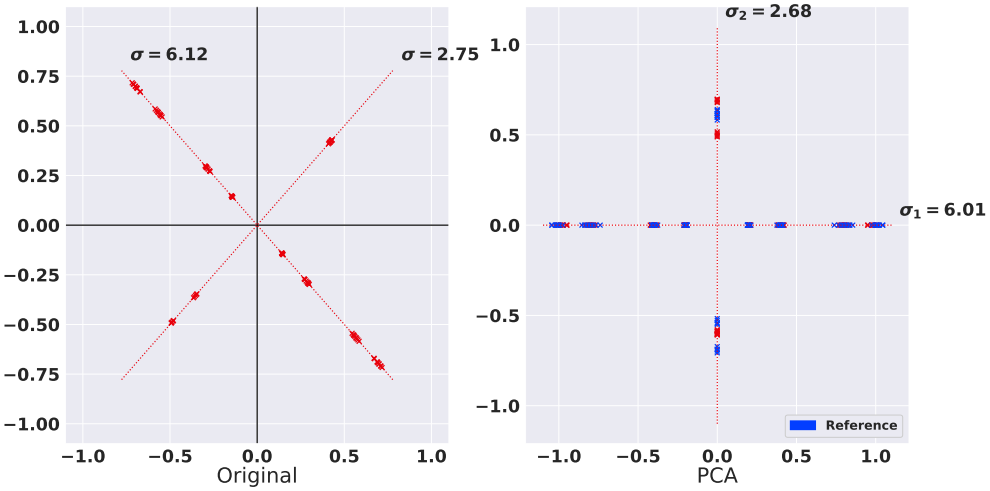


Figure 3.14 – PCA of an almost identical point cloud to the one of Figure 3.13.

then create a point cloud with the same parameters and compute its PCA that we present in Figure 3.14, where we can see that this second axis has been flipped. As such, the two output clouds are not aligned anymore. This behaviour is in fact the most frequent we encounter, as it is a result of the SVD algorithm used to compute the PCA (and MDS). The SVD (presented in Section 1.3.1, on page 13) decomposes a m by n matrix A in $A = U\Sigma V^*$. While in this formula Σ is unique, U and V are only determined up to signs column by column. This means that for any k, if we call (u_k, v_k) the k^{th} columns of U and V, then changing them for $(-u_k, -v_k)$ would give us another valid solution for the SVD. This implies that for the same input matrix depending on the algorithm used, the architecture the program is executed on, the result may change by a reflection around some principal components cause by a change of sign in U and V. Similarly, with similar conditions, a small error on the input matrix can cause a similar change to happen. This can effect any column of U independently. Just like the inversion of axes, the reflections have a significant impact on distance between point clouds, and as such need to be accounted for before comparing point clouds.

3.4 State of the art on point cloud alignment

3.4.1 Correspondence case : Procrustes analysis and the orthogonal Procrustes problem

Procrustes analysis [117, 65, 60, 90] is a way to compare shapes, by fitting one object onto another. It takes its name from a mythological Greek bandit who either would cut of or stretch the limbs of his victims in order to make them fit in his metal bed. Let us consider two matrices $A \in \mathbb{R}^{m\times k}$ and $B \in \mathbb{R}^{m\times k}$, a Procrustes transformation will try and apply the best combination of rotation, reflection, translation and scaling to transform B into A, by trying to find a $W \in \mathbb{R}^{k\times k}$ such that $\|A - BW\|_F$ is minimal. In the particular case where we want to achieve such a result using a orthogonal transformation, meaning only a rotation or a reflection, we talk about the orthogonal Procrustes problem and by convention we write it with Q: $\min_{Q\in\mathcal{O}_k}\|A - BQ\|_F$ with $\mathcal{O}_k$ being the set of orthogonal matrices of size $k \times k$. The solution to the Procrustes transformation is well known. However it can only be applied when there is the same number of points in each cloud. It is even more restrictive as it requires a correspondence between the

points. As we are interested in comparing point clouds that do not have these properties, we present methods that can achieve this in Section 3.4.2

3.4.2 Non-correspondence case : Wasserstein-Procrustes and Gromov-Hausdorff

If both point clouds have the same number of points, without having a correspondence between them, then we can use the Wasserstein-Procrustes [60] distance that consists in using the Wasserstein distance to find a correspondence between the points, in order to find an optimal solution using Procrustes analysis. The Wasserstein distance is defined as:

$$\min_{\Pi \in \mathcal{P}_m} \|\Pi A - B\|_F \tag{3.18}$$

With $\mathcal{P}_m$ the set of permutation matrices. As such, the Wasserstein-Procrustes distance can be written as:

$$\min_{\Pi \in \mathcal{P}_m, Q \in \mathcal{O}_k} \|\Pi A - BQ\|_F \tag{3.19}$$

If the point clouds do not have the same number of points, then we rely on the Gromov-Hausdorff distance [94, 62]. The concept of the Gromov-Hausdorff distance between point clouds is to find the best isometry between the point clouds in the sense that it minimizes the Hausdorff distance between them.

Since both these methods are quite costly to compute in practice, we do not implement them and instead we present in Section 3.5 the methods we choose for comparing point clouds arising from MDS.

3.5 MDS point cloud alignment

Let us consider two datasets **X1** and **X2** that we want to compare. We may want to compute their MDS independently, as it is shown in figure 3.15, however this will lead to two point clouds that will not be aligned a priori. We can then be in a few different cases. Either the point clouds come from data that is close, for instance we extracted two submatrices from a single original dataset. In this case we may expect that the resulting point clouds will be approximately close. Even in this optimistic scenario where the only variation comes form the extraction of the submatrices, the unstability of the SVD presented is Section 3.3.4 will most likely create some reflections around the principal axes between the point clouds similarly to the effect presented in 3.3.4. If the point clouds come from different datasets that we want to compare, then the differences will be even greater as possibly all the sources of error presented in Section 3.3 apply.

The two datasets **X1** and **X2** can be seen as being part of a larger one. The associated full distance matrix **X** can be available or not, but it must exist at least in theory. What we mean here is that the distance between the entries of **X1** and the distance between those of **X2** must come from the same function, otherwise comparing these point clouds would not make sense. As such computing the MDS of **X1** and **X2** separately can be seen as computing the MDS on two submatrices of **X** as it is presented in Figure 3.16. In this figure, the two grey blocks represent the part of the matrix that are not available (or at least accessed) during the MDS of **X**. In such a computation, the problem of aligning the point clouds obtained from **X1** and **X2** is complicated as we do not have a correspondence between points, neither do we have the guaranty that the number of points of **X1** and **X2** are the same. In this case the only method

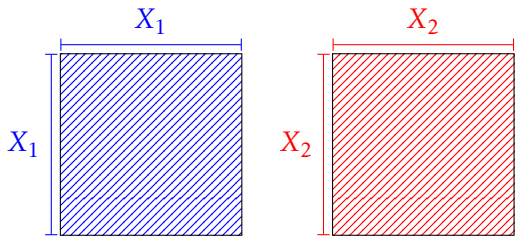


Figure 3.15 – **X1** and **X2** two datasets to be compared. Here we compute independently the MDS on each dataset.

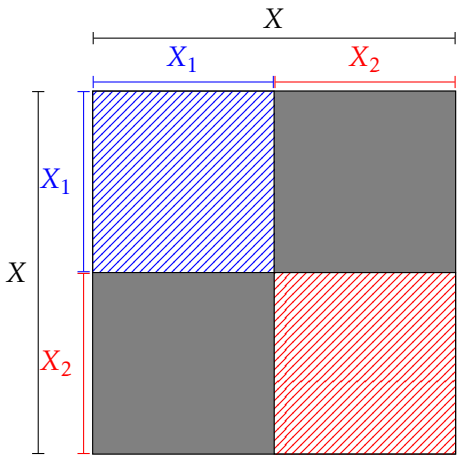


Figure 3.16 – **X1** and **X2** can be seen as part of a bigger matrix **X**. Computing the MDS on **X1** and **X2** independently can be seen as not having any information from the extra-diagonal blocks.

we know of to compute a distance between the point clouds is the Gromov-Hausdorff distance presented in Section 3.4.2 that requires us to compute all isometries between **X1** and **X2**.

Instead of doing this, we could form and compute the full matrix **X**. After all computing the MDS of **X** would align **X1** and **X2** along the principal axes of **X** and mean that they would be already aligned and can be compared using the Hausdorff distance without any alignment phase. This method is illustrated in Figure 3.17. While this method will provide us with the best possible alignment of **X1** and **X2**, it is also more costly than the previous one, as the MDS that needs to be performed is larger. On top of that, the extra-diagonal blocks between **X1** and **X2** may not be available and would need to be computed, which is often the most costly step of the whole MDS process, often requiring order of magnitudes more time to complete.

To find a compromise between those two methods, where we limit the amount of extra-diagonal blocks that need to be computed, while still being able to align the point clouds cheaply, we propose a new approach of comparing point clouds. The idea is to only compute part of the extra-diagonal blocks. To achieve this, we select a number of landmark points in **X2** to be added to **X1** and compute the necessary part of the extra-diagonal blocks. This new version of **X1** is called **X1+**. Then we can compute the MDS of **X2** and **X1+** separately. Once this is done, we have a subset of **X1+** that we know to have a correspondence with a subset of **X2**. We can then find the optimal transformation to map the landmark of **X2** to their counterpart in **X1+** using Procrustes analysis. The transformation matrix can then be applied to the full **X2** point cloud. This procedure is explained in Algorithm 11 and illustrated in Figure 3.18.

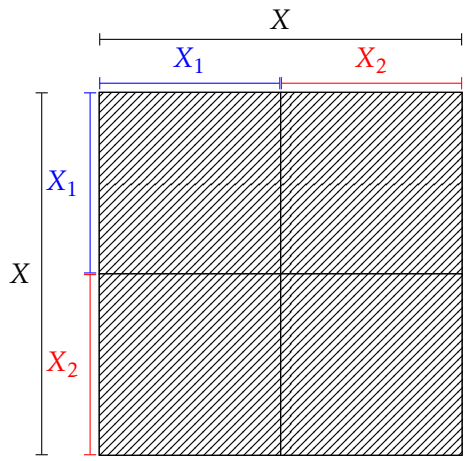


Figure 3.17 – By computing the MDS on the entire matrix **X**, we garantee that **X1** and **X2** will be aligned as we took into account the entirety of the information and **X1** and **X2** are therefore projected onto the same axes and can be compared directly.

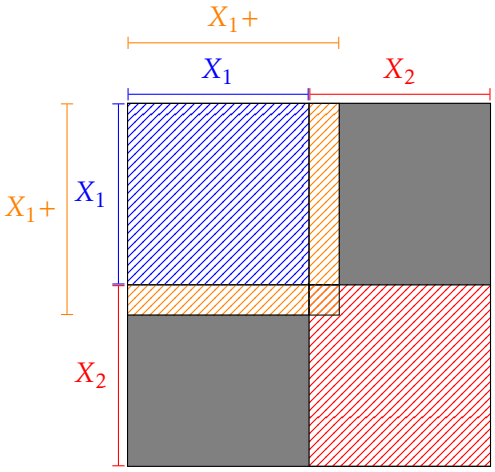


Figure 3.18 – By augmenting **X1** with entries from **X2** we obtain **X1+**. The orange part being computed in both **X1** and **X2**, these points serve as landmarks to find an optimal transformation to map **X2** to **X1** using Procrustes analysis.

Algorithm 11: Landmark Procrustes MDS.

Input: D_1 a $m \times m$ matrix, D_2 a $n \times n$ matrix, l the number of landmarks
Output: X_1, X_2 two point clouds corresponding to the MDS of D_1 and D_2 that are aligned

1 Draw a l landmarks from D_2
2 Compute D_1+ the matrix D_1 augmented with the landmarks
3 Compute X_1+, X_2, the MDS of D_1+, D_2
4 Extract X_{1l} and X_{2l} the landmarks points from X_1+ and X_2
5 Compute Q such that $\|X_{1l} - QX_{2l}\|$ is minimal using orthogonal Procrustes
6 Form X_1 by removing the landmarks from X_1+
7 Form $X_2 = QX_2$
8 **return** X_1, X_2

$$\begin{pmatrix} 1 & \dots & 1 \\ & X^T_{landmarks} & \end{pmatrix} \begin{pmatrix} \omega_1 \\ \vdots \\ \vdots \\ \omega_k \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Figure 3.19 – Augmented system aimed at expressing the barycenter of $X_{landmarks}$ as a function of the landmarks.

This method however does not take into account the centering [23] between **X1** and **X2**. In order to fix this, we propose to use the augmented system presented in Figure 3.19 to express the barycenter of **X1+** as a function of the landmark, allowing us to recenter **X2** to the center of **X1+** in order to align the two point clouds. We want to find a vector ω such that if we name l the number of landmarks and x_i the landmarks themselves we have:

$$\sum_{i=1}^{l} \omega x_i = 0 \tag{3.20}$$

We however have to add an extra condition since we are trying to find the center of the point cloud, which is always in 0 as $\omega = 0$ is a solution to this system. As such we impose the condition:

$$\sum_{i=1}^{k} \omega_i = 1 \tag{3.21}$$

This condition is translated in Figure 3.19 by the first row of ones on the first matrix as well as the first 1 on the right-most matrix and guarantee that we do not find 0 as our solution. Once we have computed omega, we can apply it to the landmarks in the second point cloud, to find where the center of the first point cloud is located inside the second point cloud and recenter the second point cloud.

3.6 Generalized MDS alignment

3.6.1 Generalized Procrustes Analysis

In the case where we want to compare more than one point clouds to each other, we may want to rely on what is called Generalized Procrustes Analysis (GPA) [58]. The base concept of GPA presented in Algorithm 12 is to use Procrustes analysis to a reference shape to compute an average shape between the point clouds. In the GPA algorithm, the reference shape is built by iteratively until all the shapes reach an average shape.

Algorithm 12: Standard Generalized Procrustes Analysis.

1 Choose a reference shape among the ones in input
2 Align all other shapes to this one
3 Compute the mean shape
4 Compute the distance between the mean shape and the reference
5 If the distance is greater than a threshold, set the reference as the mean shape and repeat the algorithm

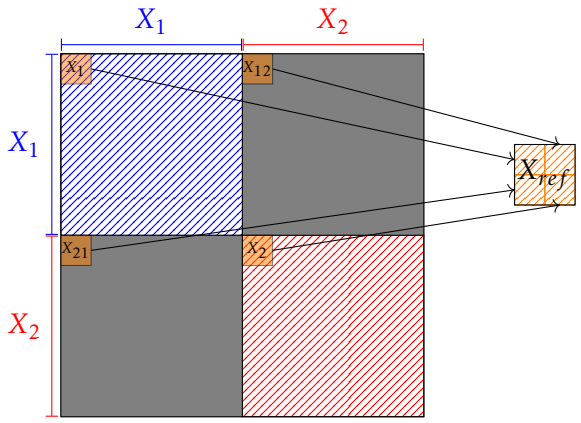


Figure 3.20 – Generalized alignment using landmarks as a reference shape in a 2×2 block matrix.

3.6.2 Application to MDS (alignment of non corresponding MDS point clouds)

In our context, we would want to use it to align point clouds that come from different parts of a distance matrix, and as such the condition of correspondence is not verified. We can still come with a solution inspired from the idea of Figure 3.18 where we rely on landmarks as common points to align our matrices using Procrustes analysis. However instead of augmenting one or more matrices, we propose this time to select a few entries from each matrices to be rebuilt into a reference matrix of landmarks as illustrated in Figure 3.20. Computing the MDS on the landmarks results in a reference shape that does not need to be iterated upon. The GPA algorithm for MDS analysis is described in Algorithm 13. In Section 3.8.3 we use this method to compare the different datasets composing the $L_1,\ldots,L_{10}$ sample of Section 1.8.4 (p. 21). In this configuration, the diagram from Figure 3.20 would be better illustrated in Figure 3.21.

Algorithm 13: Generalized Procrustes MDS Analysis.

Input: L_{in} a list of matrices to compare, l a number of landmarks
Output: X_{out} all the point clouds aligned

1 Choose l landmarks spread between all the entries of L_{in}
2 Build $L_{landmark}$ the distance matrix composed of the landmarks
3 Compute $X_{landmark}$ the point cloud of the landmark matrix
4 Compute X_{MDS} the list of the MDS of all entries of L_{in}
5 Use Procrustes analysis between each entries of L_{MDS} and their corresponding part in X_{MDS} to build X_{out}.
6 **return** X_{out}

3.7 Block Diagonal Landmark Procrustes MDS (BDLPMDS)

When considering MDS, the limiting factor often comes to either the cost of building the distance matrix, or the sEVD computation. As we mentioned in the introduction, there exist algorithms that limit the amount of the matrix we have to build such as Landmark MDS (LMDS), which we present in Section 3.7.1. We consider here an alternative way based on a divide-and-conquer paradigm for designing an MDS [130, 76]. We review related work in Section 3.7.2. We show that the generalized alignment method we presented in Section 3.6 is similar to these

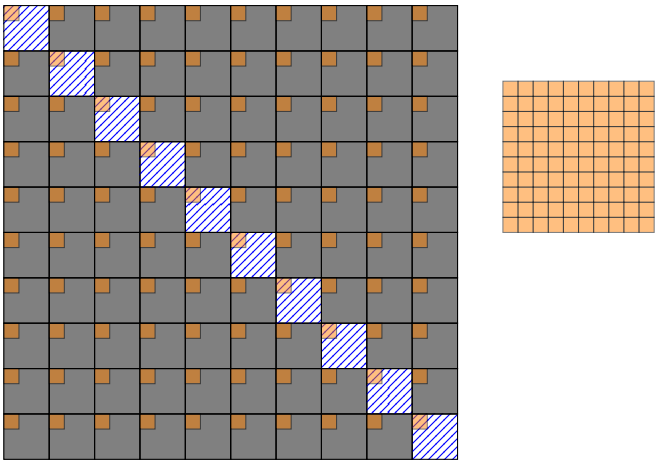


Figure 3.21 – Generalized alignment using landmarks as a reference shape in a 10×10 block matrix.

algorithms. However, contrary to these algorithms, our idea is to exploit the already computed MDS on subsamples (associated with principal submatrices of the dissimilarity matrix) in order compute the MDS of the full sample. For example in our case, we would have all the L_i of Section 1.8.4 (p. 21) but not yet the extra diagonal blocks in the distance matrix.

3.7.1 State of the art on interpolation-based MDS (LMDS-like methods)

Due to the cost of the MDS, there exist extensive work in the literature dedicated to finding ways of computing it using approximations of the distance matrix. Among these methods, the idea of starting with a partial MDS followed by an interpolation step to map the remaining points using their distance to the computed position in the reference, and in term rebuild the whole solution have been studied by many authors [118, 31, 100, 99, 44, 129]. In particular, landmark MDS (LMDS) [118, 31] (Algorithm 14 for an overview, see Figure 1 of [31] for a more extensive version) is a interpolation-based MDS based on the computation of a first MDS on a restricted number of points of the input matrix called landmarks, and in a second step places the remaining point using a triangulation method. For $G \in \mathbb{R}^{m \times m}$, we want to find $l << m$ points to act as landmarks. In this thesis, we restrict ourselves to random landmarks, although the authors of [31] also propose a MaxMin scheme to select more relevant points. The interpolation step is based on different methods. In [100], Paradis bases his approach on the formulas of [59]. In [106], Platt links LMDS (and other interpolation-based MDS) to Nyström approximation [39].

Algorithm 14: Landmark MDS algorithm.

Input: A a $m \times m$ matrix, l the number of landmarks, k desired output dimension
Output: X, Λ, an approximate MDS of A

1 Form the Gram matrix G
2 Select l landmark points from G
3 Compute the classical MDS on landmarks
4 Compute A distance-based triangulation of the remaining points to the landmarks
5 **return** X, *Lambda*

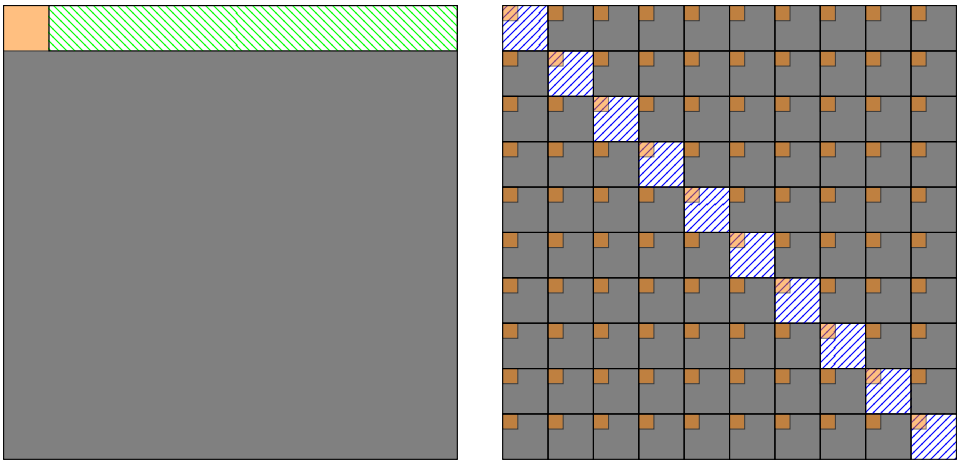


Figure 3.22 – LMDS (left) and BDLPMDS (right). The oranges blocks are reference landmark matrices, the blue dashed diagonal blocks are MDS on coherent samples. The green dashed block is used in the LMDS interpolation. The grey parts do not need to be accessed.

3.7.2 Related work on divide-and-conquer MDS

As we mentioned in the general introduction, apart from randomized algorithms, another class of scalable MDS has also been developed based on a divide-and-conquer paradigm [130, 76]. The idea is to perform independent MDS on overlapping principal submatrices. Extra-diagonal blocks allow diagonal blocks to overlap with each other. The overlap is used to align the respective MDS by Procrustes analysis [117]. The method was introduced under the name of FastMDS by Yang et al. [130]. Ketpreechasawat independently introduced a closely related method under the name of hierarchical landmark charting [76]. Lee and Choi recast them as ensemble learning methods under the name of Landmark MDS Ensemble (LMDSE) [85].

3.7.3 Block Diagonal Landmark Procrustes MDS

Our idea of Block Diagonal Landmark Procrustes MDS (BDLPMDS) is to start by executing Algorithm 13 to compute and align all the diagonal blocks of our matrix, however at the end instead of returning the point clouds, we merge them into the solution. In this step the centering is crucial, as each point cloud is centered separately and there is no reason for these centers to be coherent. Any variation in the position of the center will greatly impact the reconstructed point cloud. This issue is discussed in Section 3.3. The size of the reference can be very small compared to the full matrix as we are only interested in using those for the alignment step, and in our scenario we have already built the diagonal blocks, meaning that we can build the reference matrix at a reasonable cost. The comparison between LMDS and BDLPMDS is presented in Figure 3.22

3.8 Application

We applied the Landmark Procrustes MDS presented in 3.5 to the large L1 to L10 dataset of Section 1.8.4 (p. 21). In Section 3.8.1, we compare the result from RSVD-MDS obtained in [7] and shown in Section 1.8.4 (p. 21) and the result from RsEVD-MDS presented in Section 2.8.2 (p. 57) with the goal of estimating whether or not the issues discussed in Section 2.3 have an impact on this result. In Section 3.8.2, we propose an application of this first method on the

two point clouds L1 and L2. In Section 3.8.4 we show results using the generalized alignment method presented in Section 3.6 to study the full S5 dataset by trying to align all the L_i to a reference shape. For the sake of consistency, we will be using the relative H_2 distance even in the cases where we compute the Procrustes distance between.

3.8.1 Comparison of the point clouds resulting from RSVD-MDS and RsEVD-MDS

In this section, we compare the point clouds from the S5 dataset obtained with either randomized method from Chapter 2. As a reminder, the point cloud obtained from RsEVD-MDS is presented in Figure 2.19 (p. 60) and the one obtained from RSVD-MDS is presented in Figure 1.13 (p. 26). As both represent the same point cloud and have been computed using the same distance matrix, we are in the case where we have a correspondence between the points of both clouds, we can use Procrustes analysis as described in Section 3.4.1. Using this method to align point clouds, we then computed the distance with the relative H_2 distance and found both point cloud to be mostly identical with a difference of 1.82×10^{-6} when compared in two dimensions and a difference of 6.92×10^{-5} when using the 20 available dimensions we computed of both point clouds. The results is shown in Figure 3.23.

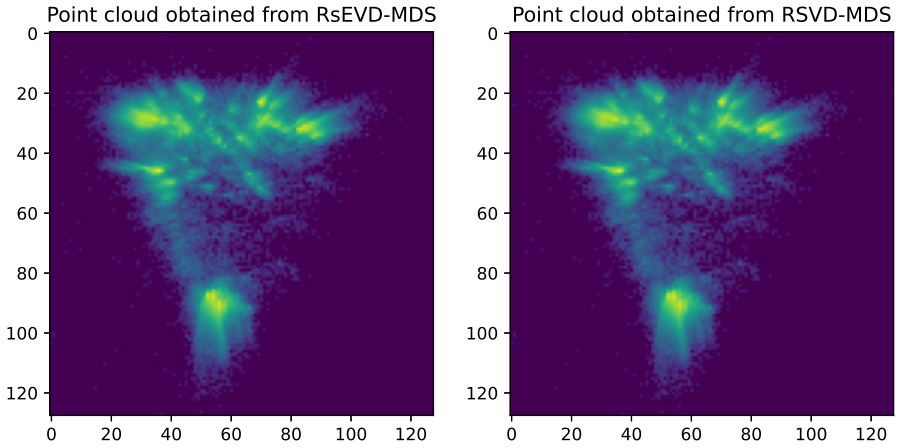


Figure 3.23 – Comparison of the heatmaps of the point clouds of the S5 sample obtained using RsEVD-MDS (on the left) and RSVD-MDS (on the right).

3.8.2 Comparison of L1 and L2

To apply the Landmark Procrustes MDS procedure, we randomly selected 1000 points from L2 to build L1+ that we present in Figure 3.24. In this figure, we present in red the landmarks point that come from L2 and will serve to align L2 to L1 using orthogonal Procrustes analysis. The result of this alignment step can be seen in Figure 3.25. We can see that the alignment found seem quite convincing, and using the relative H_2 distance between L1 and L2 gives us a distance of **0.013** in **0.37** seconds.

The alignment was computed using the first 20 dimensions of both point clouds, although we only kept the first two dimensions for the final distance computation. It is important to take a high enough number of dimensions to do this computation to avoid the issue of axes inversion presented in Section 3.3.1 as there is no guarantee that the principal axes of L1 and L2 are the same, if we only take into account the first two axes to do the orthogonal Procrustes step, we

have a risk of not comparing the same axes and ending in a wrong configuration. We illustrate this example in Figure 3.27 in which we try to reproduce the same aligning step using only the first two dimensions. Here we can see that the final point cloud obtained from L1+L2 is quite different, as highlighted for instance by the part in the red circle. This 2D comparison was performed in **0.32** seconds, and gave a distance between L1 and L2 of **0.10** so almost an order of magnitude higher than the previous solution for a negligible gain in term of performance compared to using 20 dimensions.

In order to determine what dimension we needed to use for the comparison, we studied the evolution of the relative H_2 distance between the point clouds in their first two dimensions with regards to the dimension of the point clouds considered in the orthogonal Procrustes step, and present this result in Figure 3.28. In this figure we see that the distance is unstable until about the 20^{th} dimension after which it starts to stabilize. We believe it to be due to the fact that the point clouds we obtain only have relevant information in the first 20 dimensions with the rest being noise. This is discussed in more details in Section 3.9.

The final point we want to address in this section is to compare our approach to the actual L1+L2 dataset computed using the full distance matrix following the diagram presented Figure 3.17. In our method we have a bias towards L1 as it is used as the reference of the computation of the point cloud since the orthogonal Procrustes step has the effect of mapping L2 onto the principal axes of L1. If the principal axes of L1 and L2 are distinct enough, it is actually possible that the principal components of L1 are not the ones of L1_L2. since both the reconstruction of L1+L2 and the full point cloud possess the same number of point, and that we build them maintaining the correspondence between all points, we can apply the orthogonal Procrustes step to compare them. Figure 3.29 presents the point clouds obtained through both methods and we can observe that they are indeed very similar, which is confirmed with the relatively low relative H_2 distance between them that we computed to be 3.24×10^{-3}.

3.8.3 Pairwise L_i comparison using generalized alignment

In this section we use the generalized alignment method proposed in Algorithm 13 of Section 3.6 (p. 82) to align all L_i to the same shape. In order to obtain our reference shape, we extracted a few submatrices from the full dataset, to compare the impact of the size of the reference to the quality of the alignment. Since we have access to the full matrix, we decided to use it among our references. Other than this we considered references of size 120 000, 1 000, 250 and 180. In order to build those references, we randomly selected indexes from the full matrix to be put into each reference matrix. This has for effect to have selected roughly the same proportion of element from each L_i matrix, however as they vary in size from one to another, this translates in some matrices having more points in the references. We summarize the results from the extraction in Table 3.1. Once we built the different reference matrices and computed their MDS, we centered them all and used Procrustes analysis to align them all to the full sample for clarity. We then took the point clouds of all the L_i and aligned them to each reference using again our centering method and an orthogonal Procrustes step. All these point clouds are represented in Figure 3.30. We can see that using the three largest references yield very comparable point clouds, where the smaller sizes can sometimes yield correct results, while at the same time giving completely unrecognizable point clouds for other entries. To evaluate the quality of these alignments, we compared each obtained point cloud with the ones obtained from a full MDS, meaning that even the L_i that were aligned to the full point cloud were compared to the L_i directly extracted from this point cloud. The distances we computed are presented in Table 3.2. With all the L_i aligned to the same shape, it is now possible to compare them with one another. For the sake of conciseness, we only compared **L1** with all other L_j but every other comparison would have been possible. The result of these comparisons are

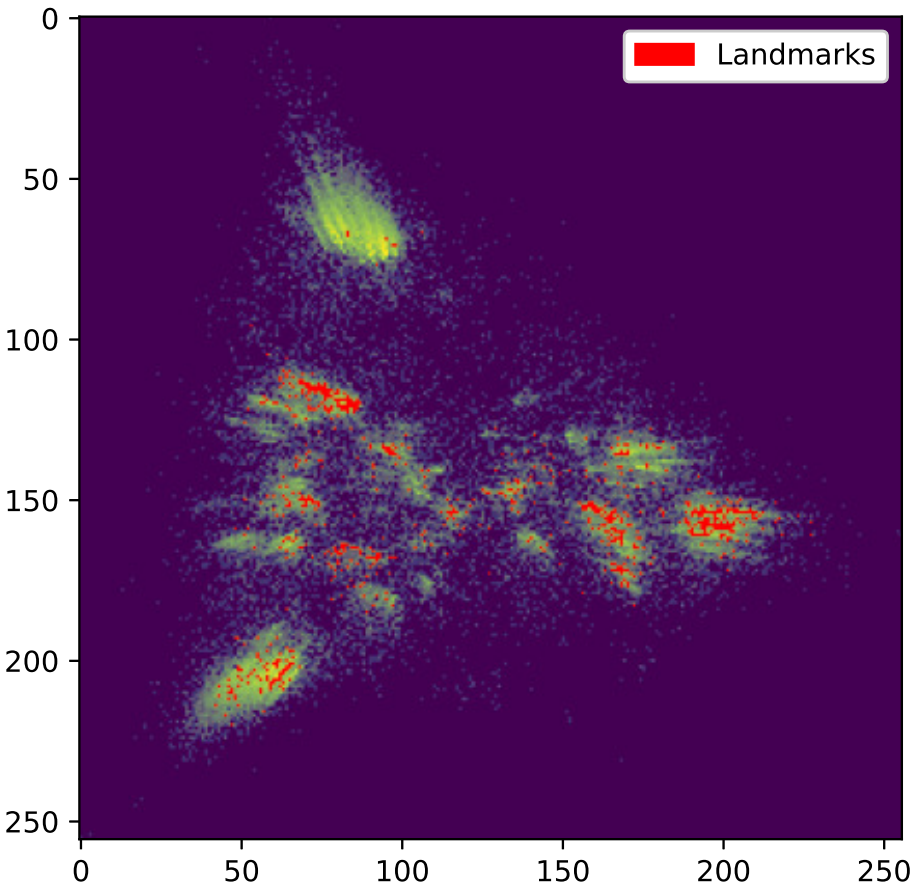


Figure 3.24 – L1+ point cloud using 1000 points of L2 as landmarks, presented in red on the heatmap.

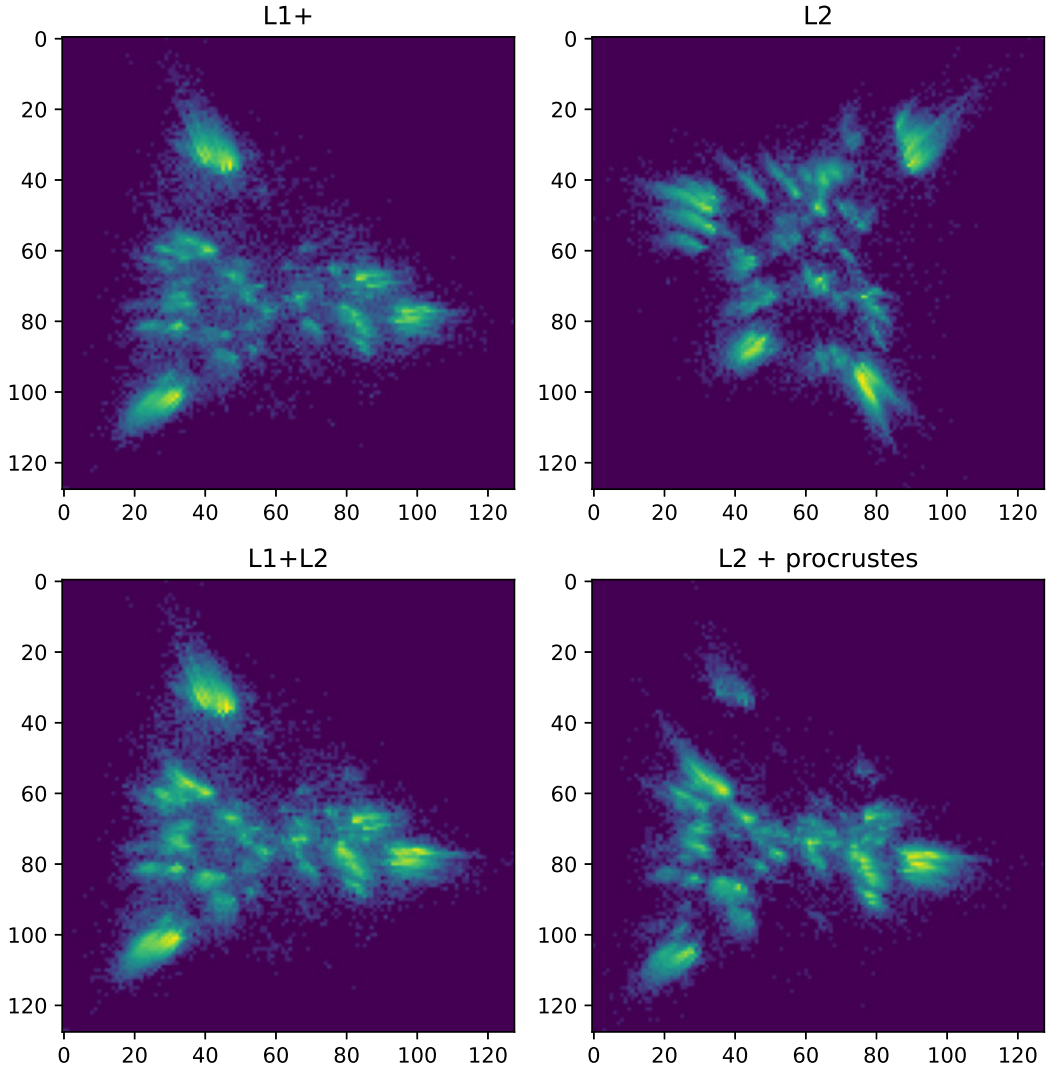


Figure 3.25 – Comparison of the heatmaps of L1 and L2 using the procrustes landmark approach in 20 dimensions. On the top we see the separated point clouds L1+ on the left. and L2 on the right. On the bottom, we see the L2 point cloud after having applied the procrustes transformation on the right and the full reconstructed L1+L2 point cloud on the left.

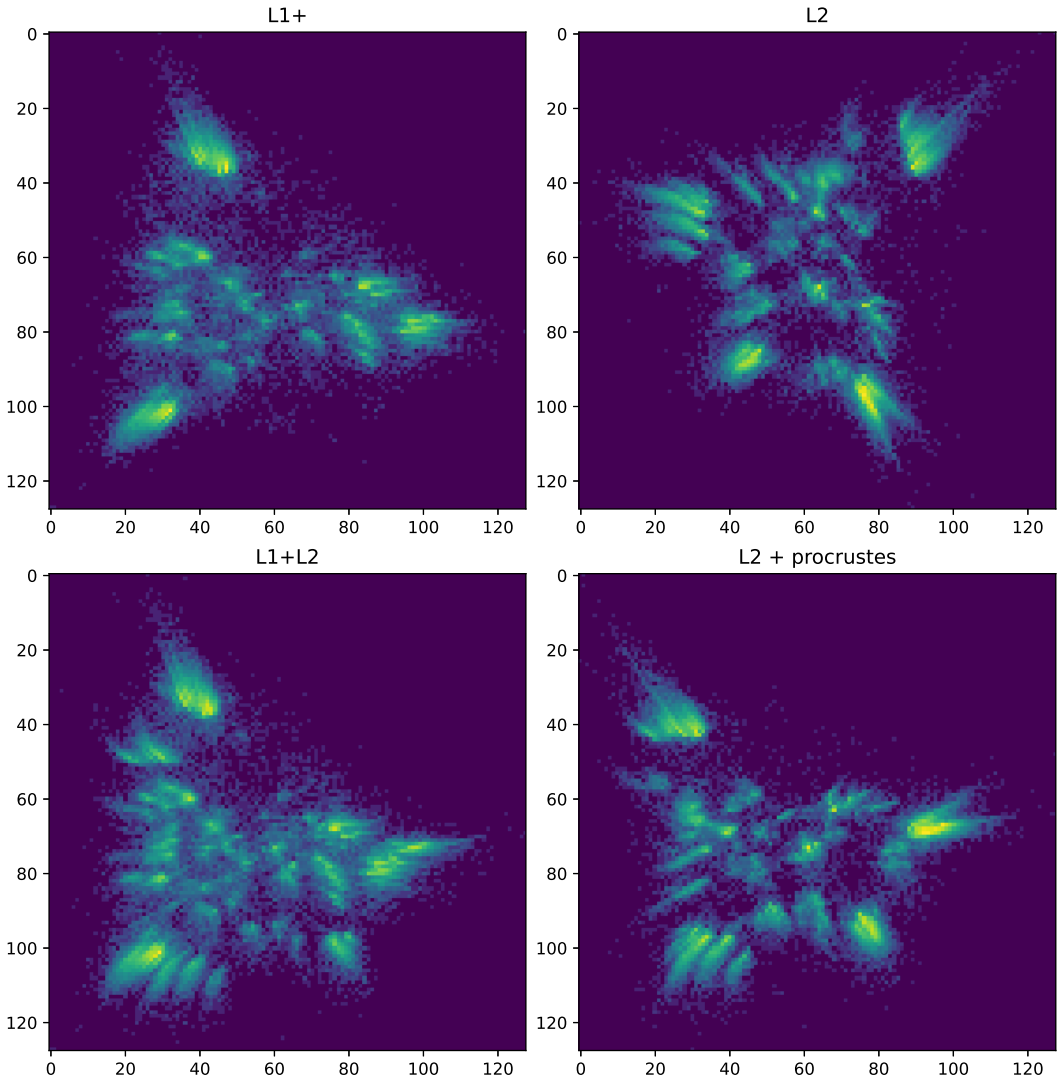


Figure 3.26 – Comparison of the heatmaps of L1 and L2 using the procrustes landmark approach in 2 dimensions. On the top we see the separated point clouds L1+ on the left and L2 on the right. On the bottom, we see the L2 point cloud after having applied the procrustes transformation on the right and the full reconstructed L1+L2 point cloud on the left.

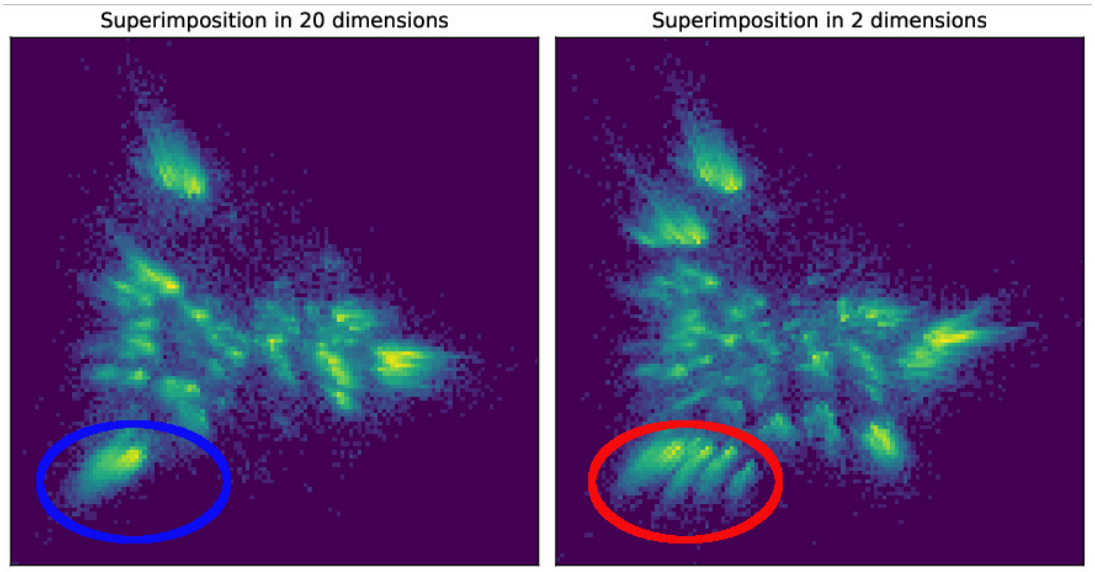


Figure 3.27 – Comparison of the heatmaps of L1 and L2 using the procrustes landmark approach in 20 dimensions (left) and in 2 dimensions (right). The circles Highlight parts with significant visual difference.

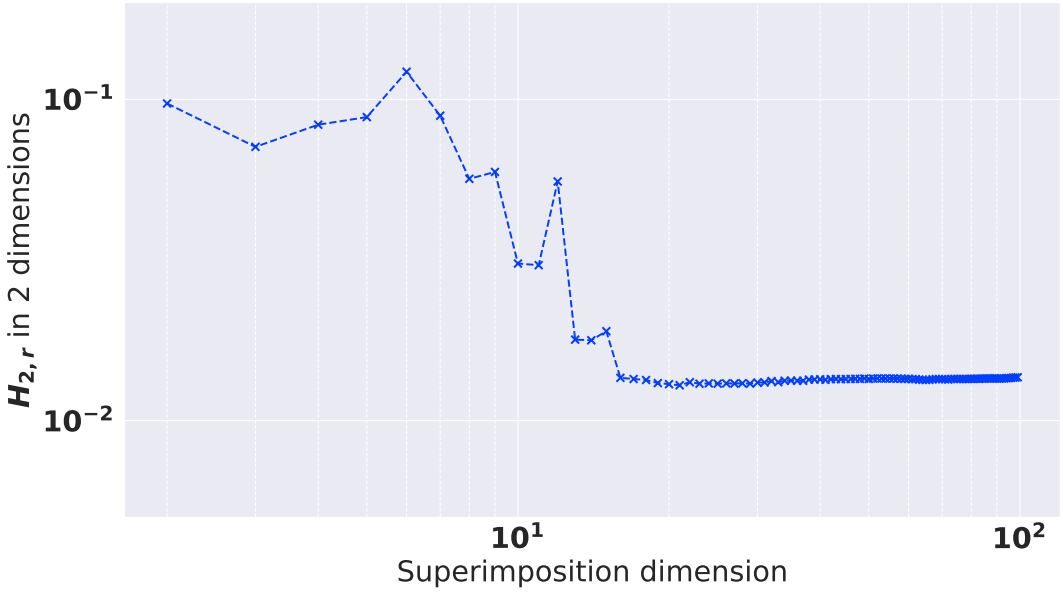


Figure 3.28 – $H_{2,r}$ distance between the two dimensional projection of L1 and L2 in function of the dimension used for the Procrustean analysis step ranging betwen 2 and 100 dimensions.

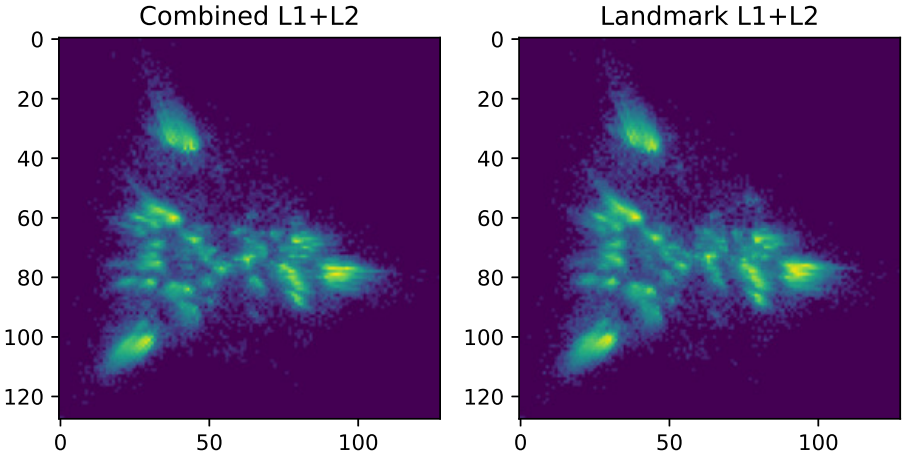


Figure 3.29 – Heatmaps of L1+L2 obtained using the MDS on the full L1_L2 submatrix (left) presented in Figure 3.17 or obtained using the landmark method (right) presented in Figure 3.18.

performed using the $H_{2,r}$ distance and presented in Table 3.3. The same results using all the variation of the Hausdorff distance presented in Section 3.2 are available in Appendix B.1.

	full		120,000		1,000		250		180	
	nb	%	nb	%	nb	%	nb	%	nb	%
L1	72,083	100	8,422	11.7	76	0.11	14	0.02	4	5.5×10^{-3}
L2	98,492	100	11,232	11.4	85	0.09	30	0.03	15	0.02
L3	72,897	100	8,499	11.7	82	0.11	20	0.03	12	0.02
L4	136,450	100	15,684	11.4	127	0.09	36	0.03	24	0.02
L5	75,218	100	8,685	11.5	81	0.11	14	0.02	16	0.02
L6	99,594	100	11,536	11.6	87	0.09	26	0.03	15	0.02
L7	124,367	100	14,162	11.4	107	0.09	33	0.03	22	0.02
L8	115,607	100	13,214	11.4	94	0.08	19	0.02	19	0.02
L9	81,983	100	9,314	11.4	86	0.10	19	0.02	17	0.02
L10	166,501	100	19,252	11.6	175	0.11	39	0.02	36	0.02

Table 3.1 – Ratio of each L_i dataset extracted for the generalized alignment.

3.8.4 Reconstruction of the full S5 point cloud

In this section, we build upon the results from Section 3.8.3 to rebuild the full point cloud S5 (L1-L10) from all the L_i after they have been aligned. The resulting point clouds can be seen in Figure 3.31. In this figure we can see that the reconstructed point cloud becomes increasingly different from the reference the less point there are in the reference. Just like the results on the alignment of the point clouds, we see that up to a size as low as 1,000, the resulting point cloud seem close to the reference. This is also observable through the distances to the reference that we obtain in Table 3.4. These results can be expected as when we sample with that few points, then each sample is only represented by a few point in the reference (for a reference of size 180, we can at best have 18 points per sample). At this size, we cannot simply rely on a randomized extraction of the landmarks and must rely on other methods. Such concern have been addressed in other work related with MDS for instance for LMDS [31] the authors state that we need at least one point more than the number of dimension. As we perform

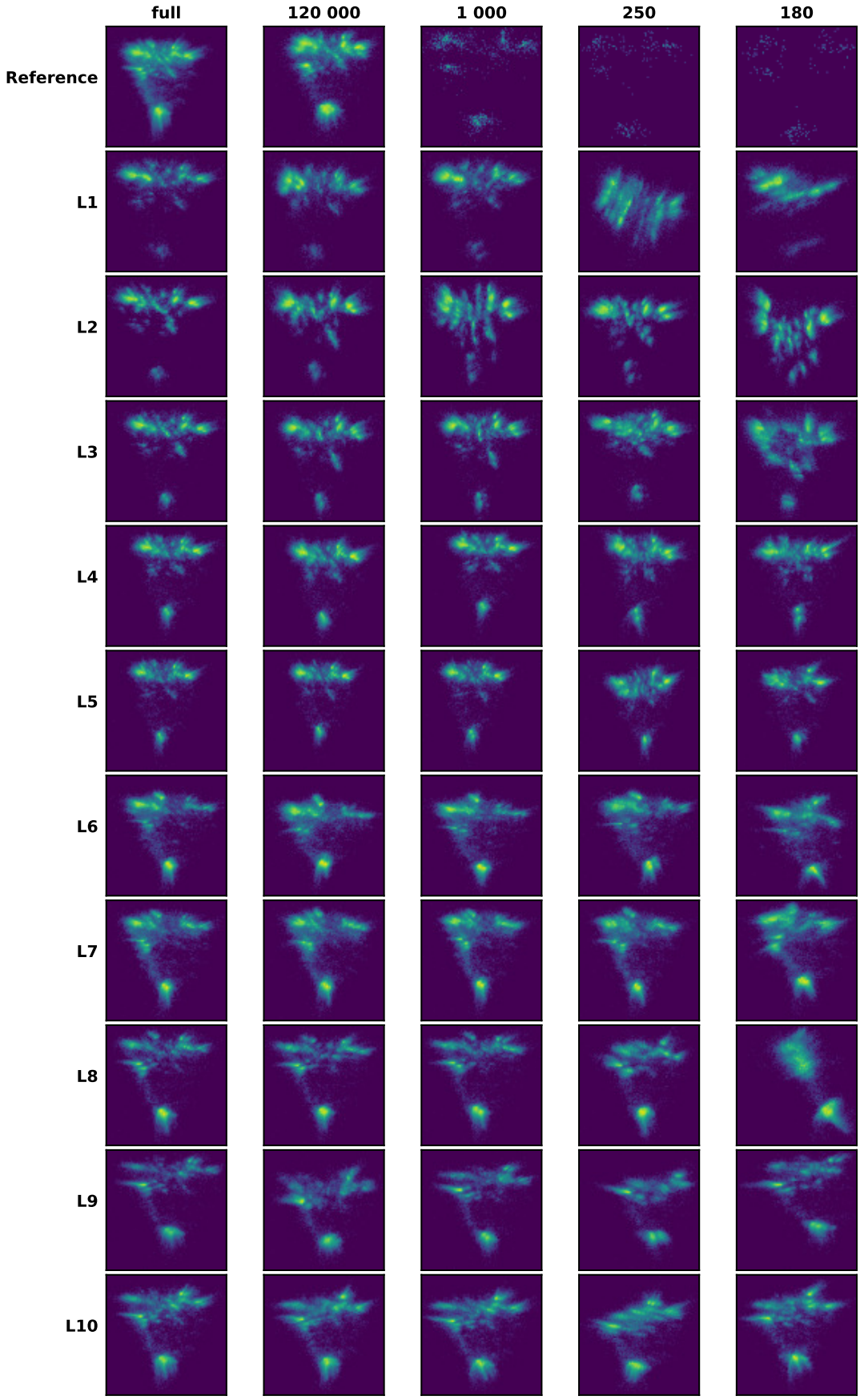


Figure 3.30 – Generalized alignment with various references (top row).

Sample	full	120,000	1,000	250	180
S5	0.00e+00	1.96e-02	2.98e-02	5.96e-02	6.80e-02
L1	1.52e-02	1.13e-02	2.37e-02	8.74e-02	7.12e-02
L2	2.23e-02	2.84e-02	3.54e-02	5.63e-02	6.06e-02
L3	1.18e-02	1.97e-02	1.70e-02	5.11e-02	4.72e-02
L4	9.92e-03	1.13e-02	1.61e-02	1.78e-02	9.73e-02
L5	1.03e-02	1.51e-02	2.18e-02	3.91e-02	2.63e-02
L6	5.62e-03	1.93e-02	3.42e-02	2.57e-02	3.58e-02
L7	4.93e-03	1.76e-02	3.46e-02	2.02e-02	4.29e-01
L8	5.24e-03	1.37e-02	2.22e-02	1.79e-02	4.81e-01
L9	8.07e-03	3.15e-02	3.31e-02	2.65e-02	4.20e-02
L10	5.03e-03	2.22e-02	3.00e-02	2.23e-01	2.62e-02

Table 3.2 – Distance computed betwen aligned point cloud to reference extracted from full computation.

	Direct extraction from full *S5*	Alignment with Xref				
		S5	120,000	1,000	250	180
L2	8.53e-03	9.29e-03	1.75e-02	2.18e-02	1.85e-01	4.90e-02
L3	6.65e-03	6.93e-03	1.30e-02	2.24e-02	1.86e-01	4.86e-02
L4	1.03e-02	1.10e-02	1.09e-02	1.61e-02	9.40e-02	3.18e-01
L5	1.34e-02	1.45e-02	1.53e-02	1.80e-02	1.01e-01	3.85e-02
L6	1.54e-02	1.61e-02	1.93e-02	3.35e-02	1.05e-01	1.45e-01
L7	1.68e-02	1.72e-02	1.94e-02	3.32e-02	9.83e-02	3.76e-01
L8	1.35e-02	1.47e-02	2.20e-02	3.12e-02	1.53e-01	2.05e-01
L9	1.73e-02	1.92e-02	2.65e-02	3.80e-02	1.08e-01	1.30e-01
L10	1.79e-02	1.96e-02	2.68e-02	3.88e-02	2.97e-01	4.60e-02

Table 3.3 – $H_{2,r}(L_1, L_j)_{j\in[\![2;10]\!]}$.

our alignment in 20 dimensions, and all of our samples need to be aligned separately to the reference, it needs to be comprised of at least 21 points per sample, so be at least of size 210. In Algorithm 1 of [100], Paradis proposes a selection method where random sample are not drawn to the region of high density of the point clouds. Other studies, unrelated to MDS have also been dealing with the issue of properly selecting the samples of the matrix, we can cite adaptive cross approximation [16]. Regardless, computing MDS is often constrained by the computation of the distance matrix, and such method may lead to the computation of many more entries of the matrix than simply using a larger but completely random method, and as such this is the method we will be focusing on.

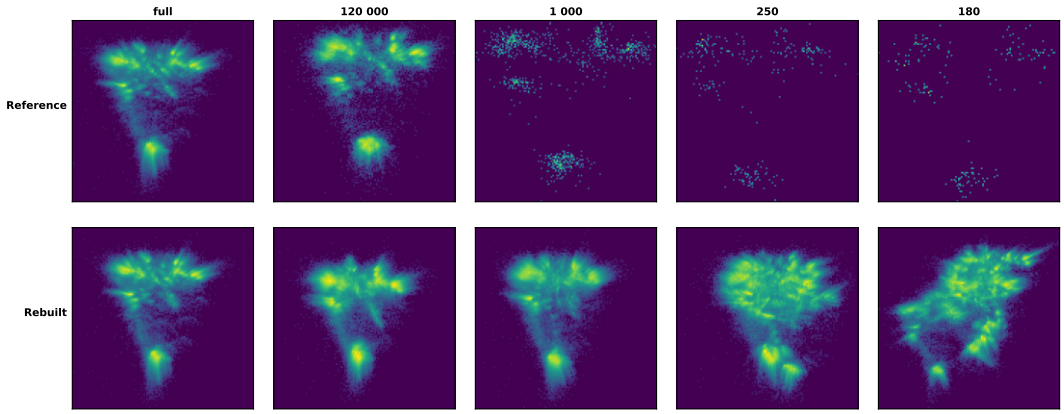


Figure 3.31 – Comparison of point clouds obtained using BDLPMDS with various number of landmarks.

full	120,000	1,000	250	180
1.58e-03	3.11e-03	2.97e-03	2.18e-02	1.01e-01

Table 3.4 – Distance computed betwen rebuilt S5 point cloud to exact reference computed with regards to the number of landmarks.

3.9 Discussion about dimensionality

When comparing point clouds, one very important question is how many dimensions to take into account. For a visual comparison, it is only possible to consider two (or sometimes three) at once bar using a plot similar to the the one in Figure 1.13 (p. 26) that is still just doing many 2 dimensional comparisons. With the use of the Hausdorff distance (and variations) we are not limited by the possibility or not of considering more dimensions. However even if we can do it, it does not necessarily mean that we should and that these higher dimensional comparison would make sense. There are effectively two considerations to have about the dimension of the comparisons, one about the dimensionality of the high-dimensional space and the other about the dimensionality of the data we study. The first is about the meaning of high-dimensional comparison. A distance in 3 dimensions and one in 2 dimensions are not directly comparable as the higher the dimension, the emptier the space gets and in the end no point end up being close. More results about the size of high-dimensional space can be found in [80]. About the dimensionality of the data, we compute the MDS with various ranks, and can have access to thousands of dimensions, however we need to evaluate how many of these dimensions are actually relevant. In our case, we believe the number or relevant dimensions to be around 20, with the rest being mostly noise. To highlight this observation, we propose to look at the coordinates of the points of a point cloud projected in one dimension at

a time and represent the histogram of these coordinates. What we expect is to see the clusters represented as different areas of high density in the histograms, and to observe the distribution of the coordinates to start converging towards a normal distribution when there is no more information and we are capturing noise. We present these experiments in figures 3.32 for the L1 dataset and 3.33 for the full S5 sample. The grey bars represent the density of the position while the reference curve represents the normal distribution we would expect using the mean and standard deviation of the position. The closer the coordinates match with the red distribution the less information we expect to be in the corresponding dimension of the point clouds. We can see that as expected, the higher the dimensions we consider, the closer the distribution tends to match with the normal distribution. This comforts us in our idea that we are expecting to have around 20 dimensions of relevant information in our point clouds. The same experiment was performed on some of the other datasets that we have used in the context of this thesis, the result of those are available in Appendix B.2.

In the end, in order to both consider the maximum amount of information while keeping the distance we compute as relevant as possible, we decided that the alignment step should take into account the first 20 dimensions of our point clouds, in accordance to the results from Figure 3.28, to correct possible issues with axes permutation presented in Section 3.3.1 for instance and then compute the Hausdorff distance in 2 dimensions.

3.10 Conclusion

In this chapter we explored the comparison of point clouds arising from MDS. Our first goal was to be able to compare the different $(L_1, \ldots, L_{10})$ point clouds that make up the S5 dataset of Section 1.8.4 (p. 21). We have shown that computing each L_i independently leads to point clouds that are not directly comparable and must first be realigned.

We proposed to rely on the Hausdorff distance to evaluate the distance between point clouds using their shapes. We presented a new variation of the Hausdorff distance that we linked to the Frobenius distance and also proposed a definition for a relative Hausdorff distance.

We identified four issues (**1: Axes permutation, 2: Rotation, 3: Translation and 4: Reflections**) that can affect the alignment of point clouds, and thus affect their distances even if they are close up to an isometry. We showed that by using orthogonal Procrustes analysis, we can correct issues **1, 2 and 4**, while we can use a simple centering method to solve issue **3**. We showed that when comparing point clouds that have no correspondence between their points, it is possible to use landmarks to act as a reference and find the transformation to align these point clouds. Such landmarks can be obtained either by augmenting one point cloud with points from the other, or by selecting points in both point clouds to compute a smaller reference MDS.

Aligning point clouds for comparison also allows one to superimpose the aligned point clouds and reconstruct the full point cloud. This can be thought of as a different view of FastMDS [130], originally motivated by the comparison of the intermediate point clouds rather than the direct calculation of MDS. We call this method Block Diagonal Landmark Procrustes MDS (BDLPMDS).

We applied the BDLPMDS algorithm to the L_i blocks using $1,000$ landmarks and were able to reconstruct the full S5 point cloud using a single computing node, whereas previously it had only been possible using a distributed-memory MDS over 100 nodes. We validated the quality of this solution by comparing it with the solution obtained using our HPC library and presented in Chapters 1 and 2. Such methods have the potential to save a significant amount of the computing time required to construct the distance matrix, as the use of the diagonal blocks together with a negligible (with respect to the size of the original sample) number of

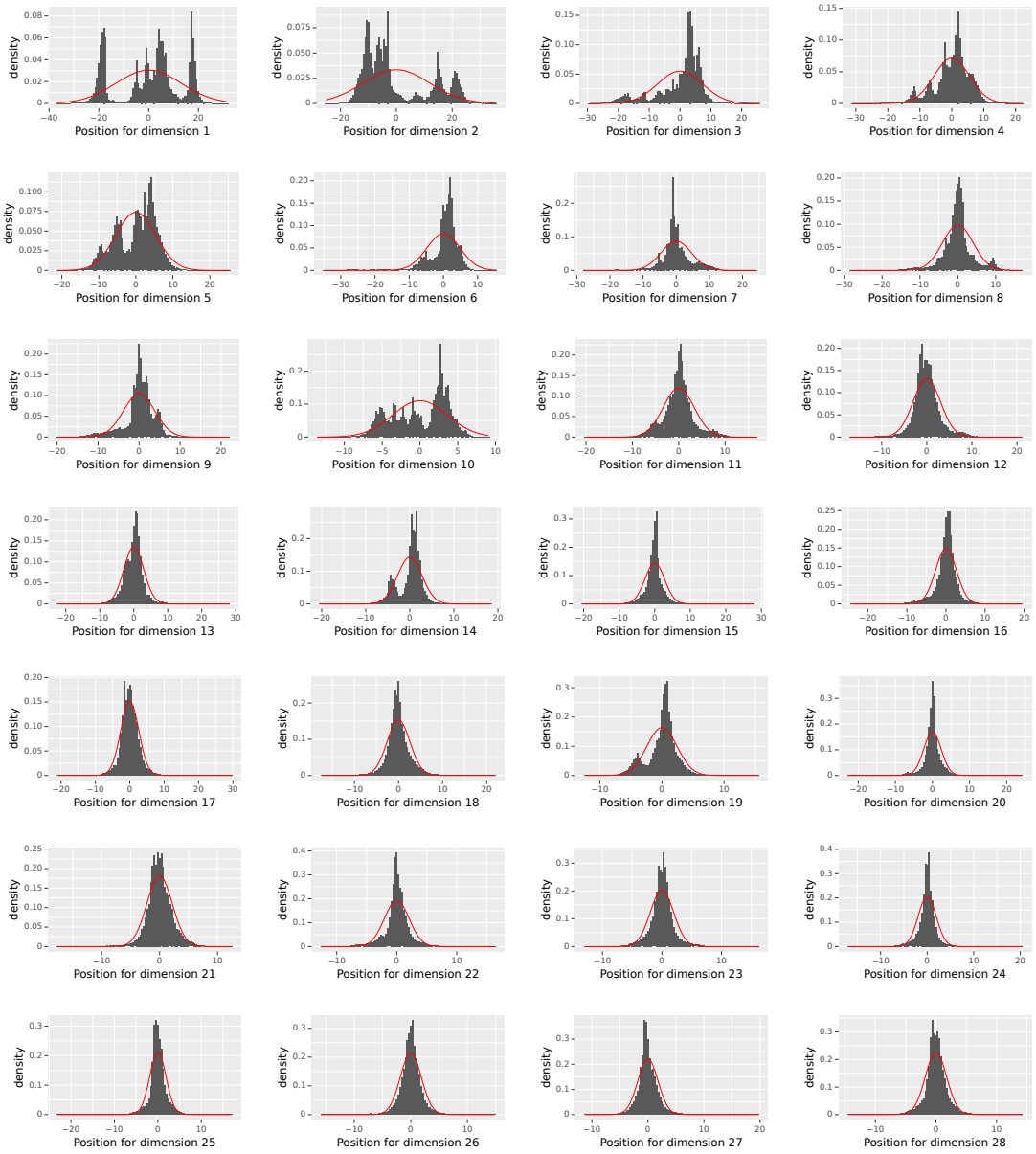


Figure 3.32 – Histogram of the distribution of the coordinates per dimension in L1 for dimensions varying from 1 to 28.

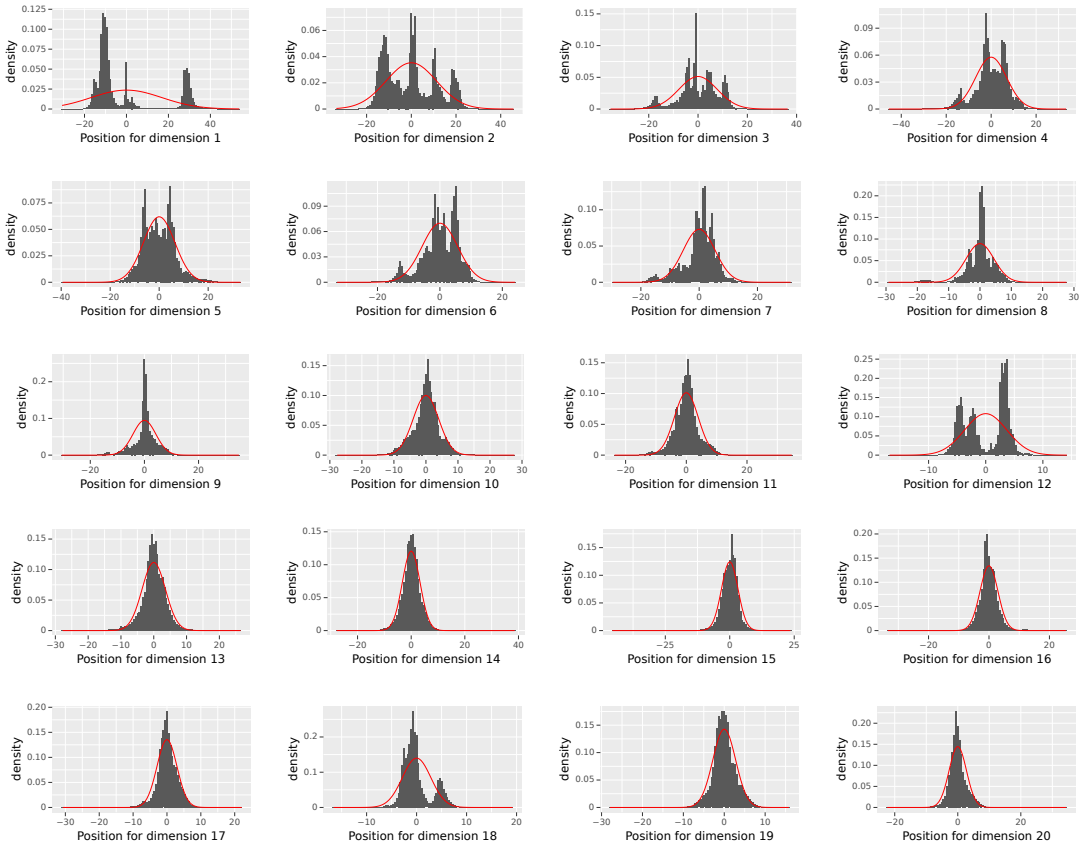


Figure 3.33 – Histogram of the distribution of the coordinates per dimension in S5 for dimensions varying from 1 to 20.

landmarks allowed us to compute the MDS of the full dataset while accessing only about 10% of the elements of the distance matrix.

As a result of this work, we can now propose a coherent set of tools for the comparison and reconstruction of point clouds, based on classical algorithms, up to the use of HPC library.

We closed this chapter with a discussion on the dimensionality of our metabarcoding datasets, having shown that the point clouds we consider can contain significant information up to the 20^{th} dimensions which can affect the orientation of the point clouds in the first two dimensions.

Chapter 4

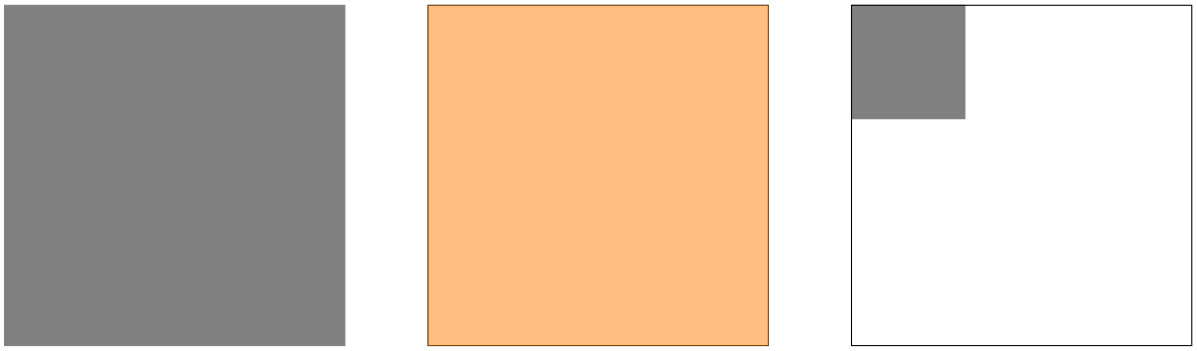
Iterative uniform sampling MDS

4.1 Introduction

The first part of this thesis, presented in Chapter 2, has revisited the numerical [20, 21, 19, 99] and HPC [7] design of MDS based on randomized linear embedding. The present chapter tackles another class of randomized algorithm, commonly referred to as *sampling* [93]. The idea is to construct a reduced MDS, i.e. a subset of the points in the cloud.

As already mentioned in the general introduction, most randomized sampling methods discussed in the literature first assess the importance of the items before randomly selecting (and possibly scaling) a subset of them according to their respective importance [93]. These *importance sampling* methods require access to the entire distance matrix D of the reference sample. Indeed, as illustrated in Figure 4.1, the reference sample (a) is entirely scanned (b) to compute importance before obtaining a reduced sample (c). Importance sampling algorithms are often outstanding when the question – referred to as *Question Q1* in the following – is to *approximate well a reference input sample* (for a given objective). Their strength is that they provide excellent *a priori* error guarantees [93]. These guarantees depend on the spectrum of the data matrix associated with the sample (and of course of the objective). Although we do not know the spectrum a priori, in practice the first (positive) eigenvalues of the datasets we consider have a strong decay, as shown in Section 1.8. This theoretical dependence should therefore not be a major problem in practice. A more fundamental problem is that the size of the reference sample with which we are dealing may be overestimated or underestimated in order to obtain an output of the quality we are intending to achieve. Indeed, while this thesis deals only with the processing of an input dissimilarity matrix D, in practice, the construction of such a matrix for the study of biodiversity is a daunting task, as discussed in Section 1.2. Therefore, approximating the original reference sample with a reduced sample may not be the good question to address for that purpose.

An alternative question – referred to as *Question Q2* in the following – is *whether we can estimate the intrinsic quality of a sample*? Figure 4.2b is a schematic view of the question: instead of approximating a reference sample as in Figure 4.2a, the aim is to estimate an unknown whole population (represented by dashes in Figure 6). This latter question is related to the use of *a posteriori* error estimators. Together with iterative schemes, such estimators are advocated by Martinsson and Tropp when sampling lacks precise guarantees [93, Section 9.7]. This chapter explores the use of the distances proposed in Chapter 3 as a means of constructing *a posteriori* error estimators for the intrinsic value of a sample. The estimator is computed using a basic

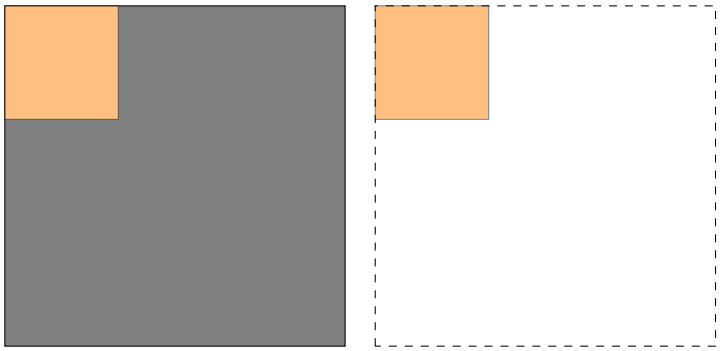


(a) Reference sample.

(b) The entire reference sample is scanned to compute importance.

(c) Resulting sample.

Figure 4.1 – MDS high-level data access pattern (up to a symmetric permutation) to the dissimilarity matrix D for *importance sampling*. The reference sample (a) is entirely scanned (b) to compute importance before obtaining a reduced sample (c). Orange blocks must be accessed.



(a) Question Q1: uniform sampling with a reference sample

(b) Question Q2: uniform sampling without any reference sample

Figure 4.2 – MDS high-level data access pattern (up to a symmetric permutation) to the dissimilarity matrix D for *uniform sampling* with (Q1) and without (Q2) reference sample (in grey). Only orange blocks need to be accessed.

resampling [72, Chapter 5, 133] technique.

It is also interesting to relate our questions to streaming algorithms [96]. Streaming algorithms are algorithms for processing data streams in which the input is presented as a sequence of items and can be examined in only a few passes, typically just one. They often produce approximate answers based on a summary or "sketch" of the data stream. As Muthukrishnan notes [96], making one or few passes for selection and sorting [79] received early attention. The concept of making few passes over the data to perform computations has been around since the early 1980s, and the work of Munro and Paterson [95] (regarded as a gem by Muthukrishnan) and Flajolet and Martin [45]. Alon, Matias and Szegedy formalised and popularised streaming algorithms in 1996 [8]. We refer the reader to [96] for an introduction to streaming algorithms. In the perspective of streaming algorithms, Question Q1 may be interpreted as the common streaming problem of examining the items of the reference sample in only a few passes, ideally just one. On the contrary, Question Q2 aims at deciding whether it is even necessary to scan all items once at all (or, conversely, whether the reference sample is sufficient to reach the intended quality).

Note that the two questions can be linked. For example, we could propose an algorithm that aims at deciding when to stop scanning the data (in the sense of Question Q2) in order to obtain an intermediate reference sample, and then to compress this data further (thus applying Question Q1 with the intermediate reference sample as input). Such a link is not explored in this work and is left for future work. Instead, our aim is to focus on Question Q2, which is less frequently investigated. We have mentioned that Question Q1 is generally best answered with importance sampling [93]. On the contrary, the first goal of Question Q2 is not to efficiently compress a reference sample but to *avoid* going through it entirely. Therefore, unless one considers combining both questions (which is not the aim of the present work), the natural way to answer Question Q2 is by *uniform sampling*. Therefore, in the rest of the chapter, we will focus only on *uniform sampling*. Although the main objective of the chapter is to address Question Q2, we present both questions side by side in order to better understand their differences, both in nature and numerical terms. For the sake of consistency, both questions are addressed using *uniform sampling* (although, as we mentioned earlier, we would prefer to treat Q1 with importance sampling if our aim was to maximise its numerical performance).

Sections 4.2 and 4.3 elaborate on questions Q1 and Q2, respectively. Section 4.4 proposes a class of iterative algorithms to address these questions in practice. Sections 4.5 and 4.6 study their numerical and performance behavior, respectively. Section 4.7 concludes the chapter.

4.2 Question Q1: iterative uniform sampling *with* a reference sample

Figure 4.3 presents the sketch of an iterative uniform sampling scheme with the aim of approximating a reference sample. Although uniform sampling only requires access to the data in orange, the quality of the resulting sample must be checked against the reference sample (in grey), which must also be scanned. As a result, this scheme requires to access to as much data (the entire reference sample) as importance sampling (see Figure 4.1). As mentioned earlier, we do not expect to gain any advantage from such a uniform sampling scheme over iterative importance sampling schemes, as the latter ones generally achieve a better numerical performance. However, examining Question Q1 using uniform sampling will allow a better comparison between questions Q1 and Q2.

Since we make no *a priori* assumptions about the population we are sampling (be it diatoms from Lake Geneva or sequences of tropical trees in French Guiana), the challenge is to design an

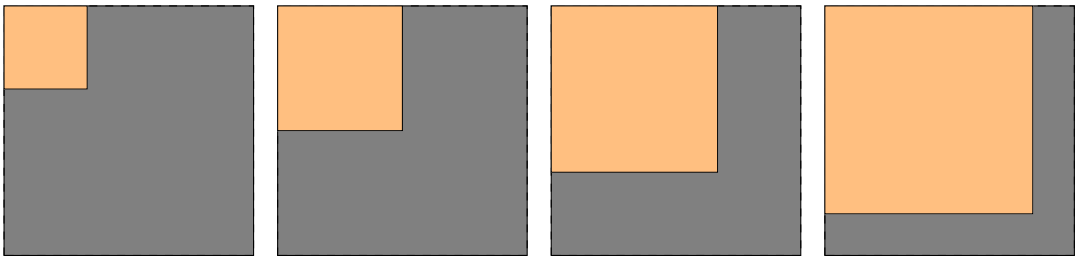


Figure 4.3 – Question Q1: Four steps of iterative uniform sampling with a reference sample (in grey). Although uniform sampling only requires access to data in orange, the quality of the resulting sample must be checked against the reference sample (in grey), which must also be scanned.

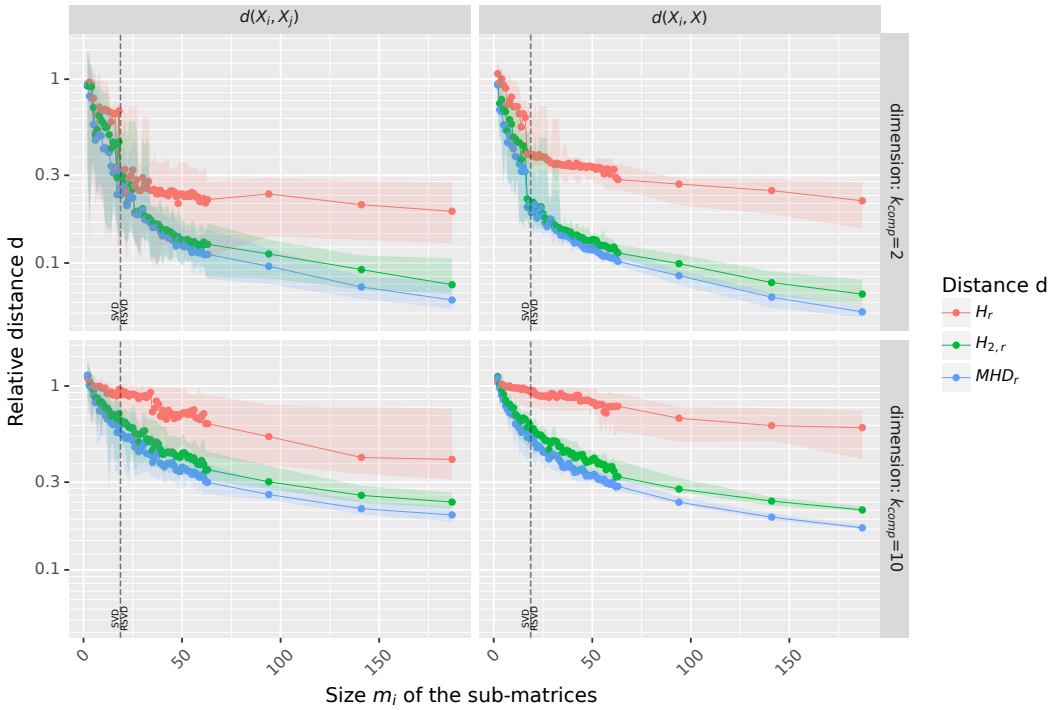


Figure 4.4 – Iterative evaluation of the H_r, MHD_r and $H_{2,r}$ distances for the Atlas Guyane dataset. Right: addressing Question Q1 from Section 4.2. Left: addressing Question Q2 from Section 4.3. Top: dimension 2. Bottom: dimension 10. Eight samples are considered for a given size (m_i). Median value is displayed in solid line.

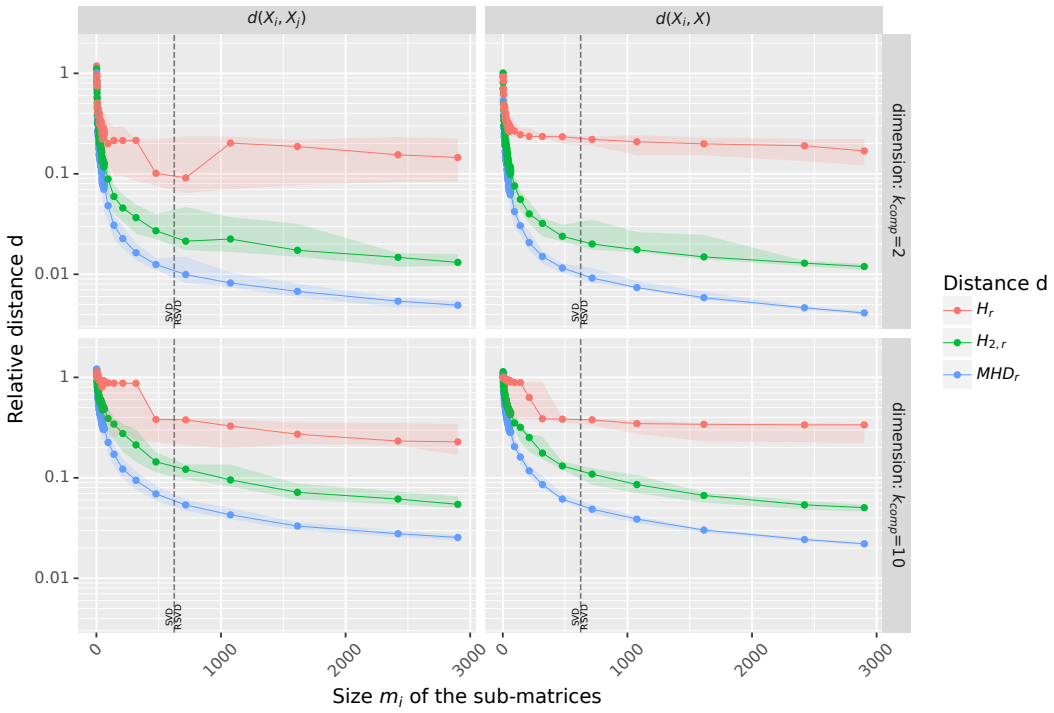


Figure 4.5 – Iterative evaluation of the H_r, MHD_r and $H_{2,r}$ distances for the 10V-RbcL. Right: addressing Question Q1 from Section 4.2. Left: addressing Question Q2 from Section 4.3. Top: dimension 2. Bottom: dimension 10. Eight samples are considered for a given size (m_i). Median value is displayed in solid line.

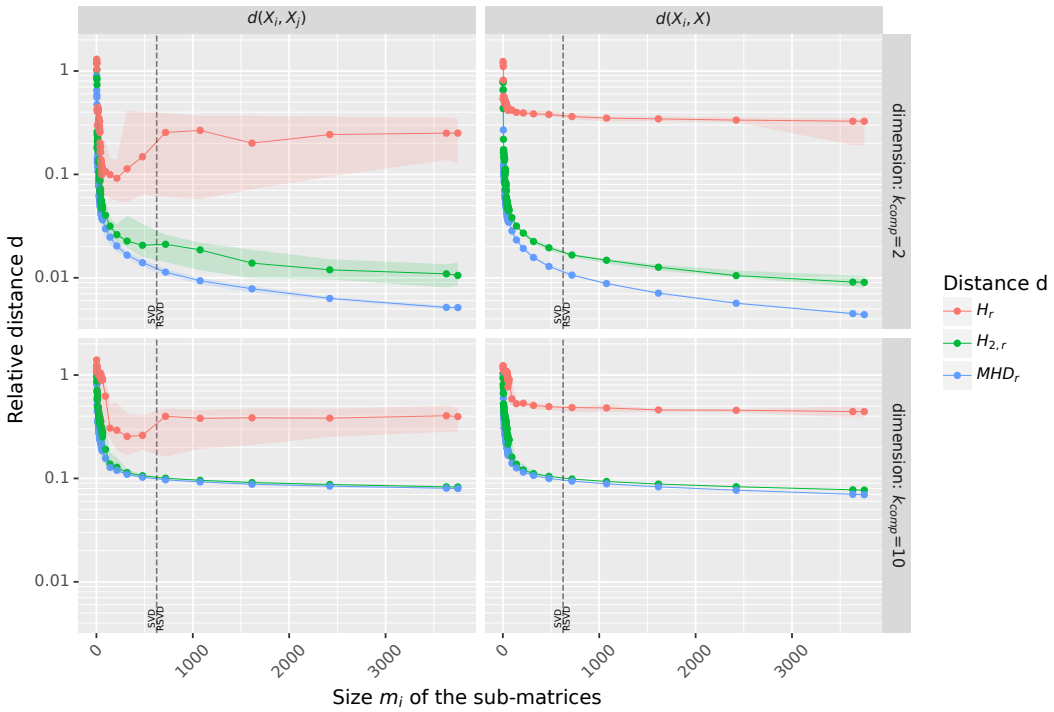


Figure 4.6 – Iterative evaluation of the H_r, MHD_r and $H_{2,r}$ distances for the Long Reads A dataset. Right: addressing Question Q1 from Section 4.2. Left: addressing Question Q2 from Section 4.3. Top: dimension 2. Bottom: dimension 10. Eight samples are considered for a given size (m_i). Median value is displayed in solid line.

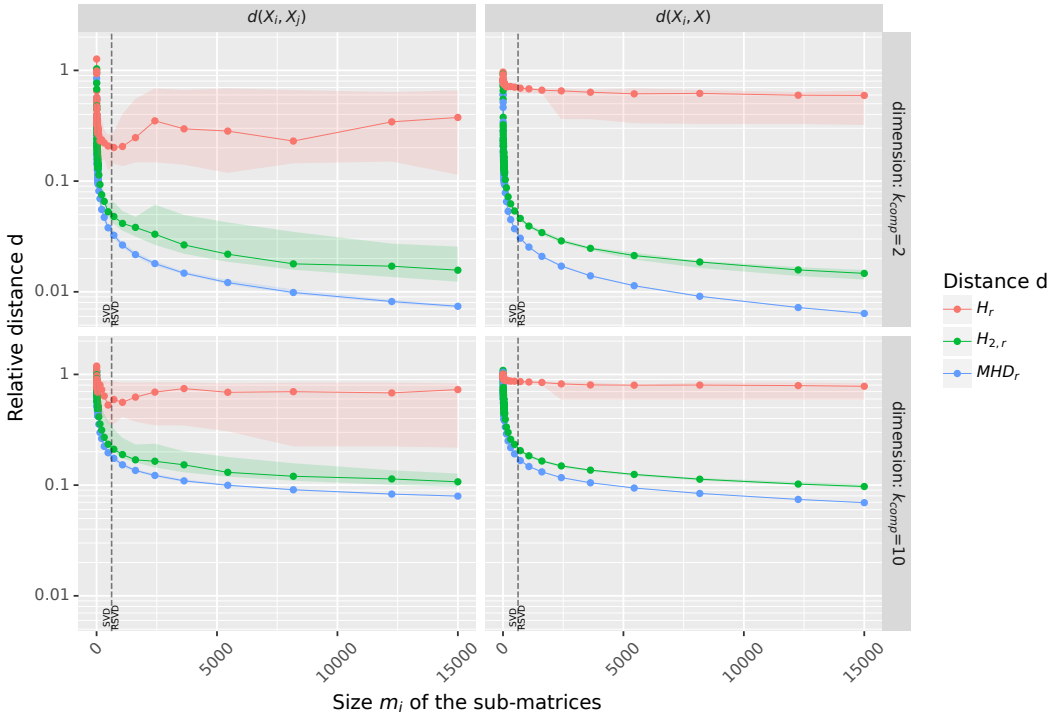


Figure 4.7 – Iterative evaluation of the H_r, MHD_r and $H_{2,r}$ distances for the S5 (L1-L10) dataset. Right: addressing Question Q1 from Section 4.2. Left: addressing Question Q2 from Section 4.3. Top: dimension 2. Bottom: dimension 10. Eight samples are considered for a given size (m_i). Median value is displayed in solid line. The Lref sample obtained from Chapter 3 has been used as a substitute to the full S5 dataset for a matter of speed.

a posteriori criterion to decide whether the sample under consideration is satisfactory. With the aim of approximating a reference sample (Question 1), the idea is to compare the sample under consideration (in orange in Figure 4.2a) with the reference sample (in grey in Figure 4.2a). We investigate to do so by using the distances proposed in Chapter 3. We have considered both the absolute (H, MHD and H_2) and relative (H_r, MHD_r and $H_{2,r}$) Hausdorff, Modified Hausdorff and Squared Modified Hausdorff distances from Chapter 3. However, we only report on relative distances as it is more relevant to use a generic criterion to deal with an unknown population with a relative criterion.

Before considering practical implementation issues (the subject of Section 4.4), we study the numerical behavior of an algorithm that would ideally follow this scheme. To do this, we consider the point cloud X (of size m) associated with the entire reference sample and we (randomly) extract a reduced sample $X^{(l)}$ (of size $m^{(l)}$ satisfying $m^{(l)} \leq m$). The sample of an iteration $l+1$ consists of an extended sample (of size $m^{(l+1)}$ satisfying $m^{(l)} < m^{(l+1)} \leq m$) of the sample of iteration l. Obviously, this algorithm is not very practical, since it requires us to have computed the full reference sample X. We call this naive reference algorithm *full*. More effective algorithms will be proposed in Section 4.4. However, in the case of Question Q1, we would still have to compute the entire reference sample for comparing the reduced sample $X^{(l)}$ to the overall sample X. At an iteration l, we assess the distance $d(X_i^{(l)}, X)$ to the reference sample of eight reduced samples $\{X_i^{(l)}\}_{(i\in[1;8])}$ of respective sizes $m_i^{(l)}$ of equal size m_i ($m_i^{(l)} = m_i$) to the reference sample. In the remainder of the chapter, we omit the superscript (l) when it is implicit.

Following the scheme in Figure 4.3, we have applied the technique iteratively to the four main datasets used in this thesis (see Section 1.8). The **right-hand** plots in figures 4.4, 4.5, 4.6 and 4.7 show the results for the Atlas Guyane, 10V-RbcL, Long Reads A and S5 (L1-L10) datasets, respectively. The most important observation is that the results show a regular decreasing median distance with iterations for both relative Modified Hausdorff (MHD_r) and relative Squared Modified Hausdorff ($H_{2,r}$) on all datasets. They can therefore be used as *robust a posteriori* estimators for answering Question Q1, as we sought. On the contrary, the relative classical Hausdorff distance (H_r) yields a less reliable estimator. Not only is the decrease less regular, but it also induces greater variability (the range of all possible values is shown in the transparent area around the median). For the sake of conciseness, we report only results associated with the relative Squared Modified Hausdorff distance ($H_{2,r}$) in the remainder of this chapter, but results associated with the relative Modified Hausdorff (MHD_r) would be as much reliable.

Figure 4.8 shows the resulting point cloud X_i when the aim is to achieve a relative Squared Modified Hausdorff $H_{2,r}(X_i, X)$ with the reference sample X equal to 0.5, 0.1, 0.05, 0.01 in columns 1, 2, 3 and 4, respectively. The rightmost column (5) also shows the full reference sample. As it has a zero distance to itself, this means that it achieves an *a posteriori* estimate equal to 0 for Question Q1 ($H_{2,r}(X, X) = 0$). From top to bottom, Atlas Guyane, 10V-RbcL, Long Reads A and S5 (L1-L10) datasets are considered. We used the 120,000 Lref sample from Chapter 3 as a substitute for the full S5 (L1-L10), as we have shown shown in Table 3.4 that it is a very good approximation of the full matrix.

4.3 Question Q2: iterative uniform sampling *without* a reference sample

An alternative question is *whether we can estimate the intrinsic quality of a sample*? Figure 4.2b is a schematic view of the question: instead of approximating a reference sample as in Figure

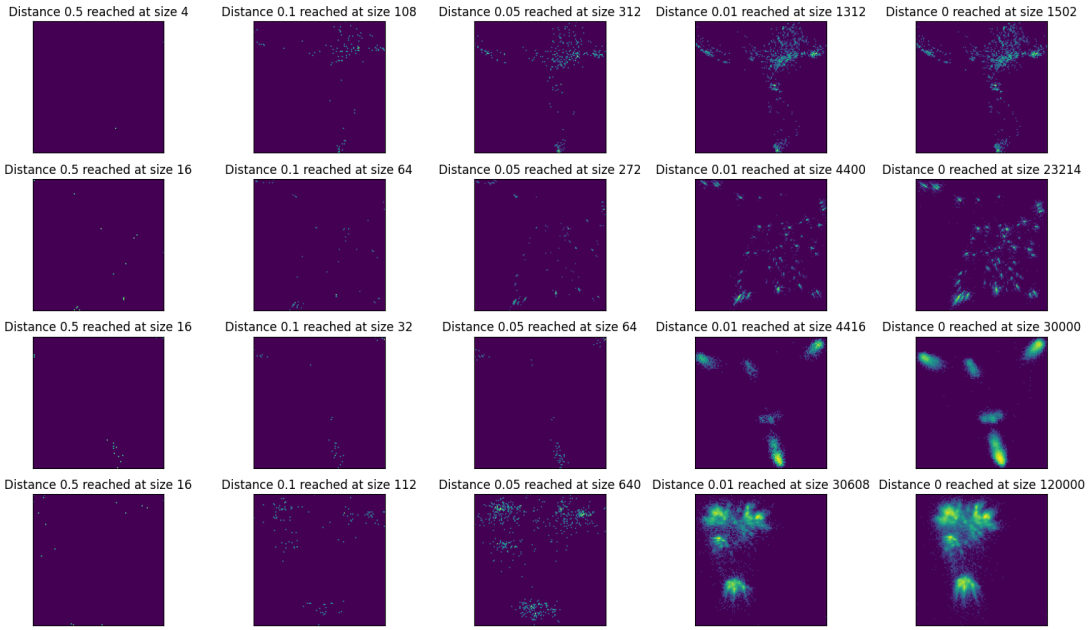


Figure 4.8 – Question Q1: What is the sample size (and resulting point cloud) required to approximate the input sample (rightmost column) at a prescribed distance $H_{2,r}$? From top to bottom: Atlas Guyane, 10V-RbcL, Long Reads A, S5 (L1-L10). The Lref sample obtained from Chapter 3 has been used as a substitute to the full S5 dataset for a matter of speed. A single sample is considered when evaluating a given size. In the rightmost column, the size required to achieve a distance of exactly 0 is the size of the reference sample itself.

4.2a, the aim is to estimate an unknown whole population (represented by dashes in Figure 6). The main difference with Question 1, is that we no longer assume that there is a reference sample to compare with.

Here is a computational interpretation in terms of streaming algorithms [96]. Question Q1 may be viewed as the common streaming problem of examining the items of the reference sample in only a few passes, ideally just one. On the contrary, Question Q2 aims to decide whether it is even necessary to scan all items once at all (or, conversely, whether the reference sample is sufficient to achieve the desired quality). The computation of the distance matrix is often the most expensive step in the MDS process. For example, computing the distance matrix of the **S5** dataset took about $1,000,000$ CPU hours (a CPU hour is the equivalent of running a calculation for one hour on a single core, so running a calculation for 2 hours on 2 nodes composed of 20 cores each would take $2 \times 2 \times 20 = 80$ CPU hours). The RsEVD-MDS calculation on this dataset, presented in Figure 2.18 (Section 2.8.2, page 57) took 151 CPU hours (129.7s on 105 nodes consisting of 40 cores each). This means that the cost of computing the distance matrix was over $6,600$ times more expensive than the MDS computation itself. Being able to assess when a submatrix is representative of the dataset can therefore save significant computation time.

There is also a deeper interpretation. In the context of this thesis, we are processing metabarcoding datasets for the study of biodiversity (see Section 1.2 page 12). These data are mainly generated by collecting samples from water bodies to extract the DNA information present. Our input datasets are therefore only a partial, and potentially biased, representation of the biodiversity present at the site where the sample was taken, and we do not know how accurately it represents the actual biodiversity. In a sense, the amount of data collected is already a first randomized sample of the population we want to study. It might therefore be interesting to assess the value of the sample *a posteriori*. Moreover, if we are able to do so, because we have

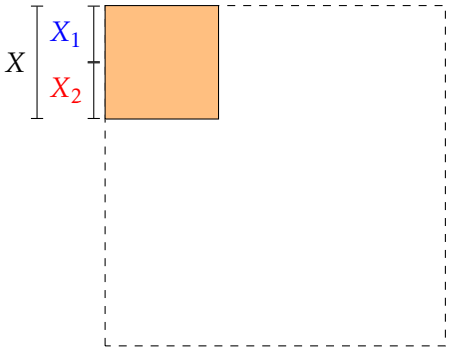


Figure 4.9 – Resampling with the creation of two new samples (**X1** and **X2**) based on the sample X under consideration.

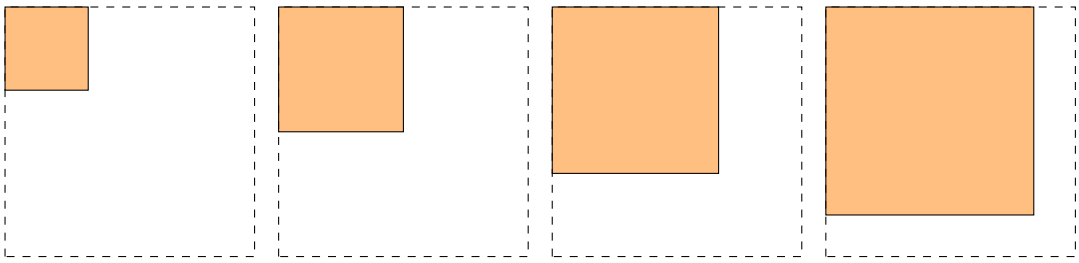


Figure 4.10 – Question Q2: Four steps of iterative uniform sampling without a reference sample. Only the data in orange are accessed. As there is no reference sample to compare with, the resulting sample can only be assessed intrinsically.

seen that computing the dissimilarity matrix can be very costly, it may also be worthwhile to compute a smaller sample and decide whether it achieves the desired quality.

Contrary to Question 1 for which there is a natural *a posteriori* estimate in terms of a distance $d(X_i, X)$ between the sample under consideration X_i and the reference sample X, there is no reference sample to do so in Question 2. We propose to rely on a basic *resampling* [72, Chapter 5, 133] scheme to construct such an estimator. To do so, we split the sample $X^{(l)}$ under consideration into multiple (two in the pictures, eight in the numerical experiments) new samples, as shown in Figure 4.9. Now that we have multiple samples $\{X_i^{(l)}\}_{(i\in[1;8])}$, we can compare them with each other by computing the distance $d(X_i^{(l)}, X_j^{(l)})$ between two new samples $X_i^{(l)}$ and $X_j^{(l)}$ resulting from the split of $X^{(l)}$. Once again, we use the distances from Chapter 3 to do so.

Note that Question Q1 addressed in Section 4.2 can also be interpreted in terms of resampling (and we will effectively do so when designing practical algorithms in Section 4.4). Indeed, we could compute a larger sample $X^{(l)}$ and resample it to obtain new smaller samples (say eight) $\{X_i^{(l)}\}_{(i\in[1;8])}$. However, while such a resampling aims to provide a more reliable estimate by reducing the variance, we could have avoided doing so (at the cost of increasing the variance of the estimate) by directly computing the distance $d(X^{(l)}, X)$ instead of the mean $d(X_i^{(l)}, X)$ (and we did so in the experiment associated with Figure 4.8). On the contrary, resampling is necessary to obtain an estimate for Question 2.

Figure 4.10 presents the sketch of our iterative uniform sampling scheme aiming to approximate a population without a reference. We have applied the technique iteratively to the four main datasets used in this thesis (see Section 1.8). The **left-hand** plots in figures 4.4, 4.5, 4.6 and 4.7 show the results for the Atlas Guyane, 10V-RbcL, Long Reads A and S5 (L1-L10)

datasets, respectively. The most important observation is that the results show a regular decreasing median distance with iterations for both relative Modified Hausdorff (MHD_r) and relative Squared Modified Hausdorff ($H_{2,r}$) on all datasets. They can therefore be used as *robust a posteriori* estimators for answering Question Q2, as we sought. On the contrary, the relative classical Hausdorff distance (H_r) yields a less reliable estimator. Not only is the decrease less regular, but it also induces greater variability (the range of all possible values is shown in the transparent area around the median). These results are thus similar to those observed for Question Q1, which is a very positive result as Question Q2 may be viewed as harder to answer.

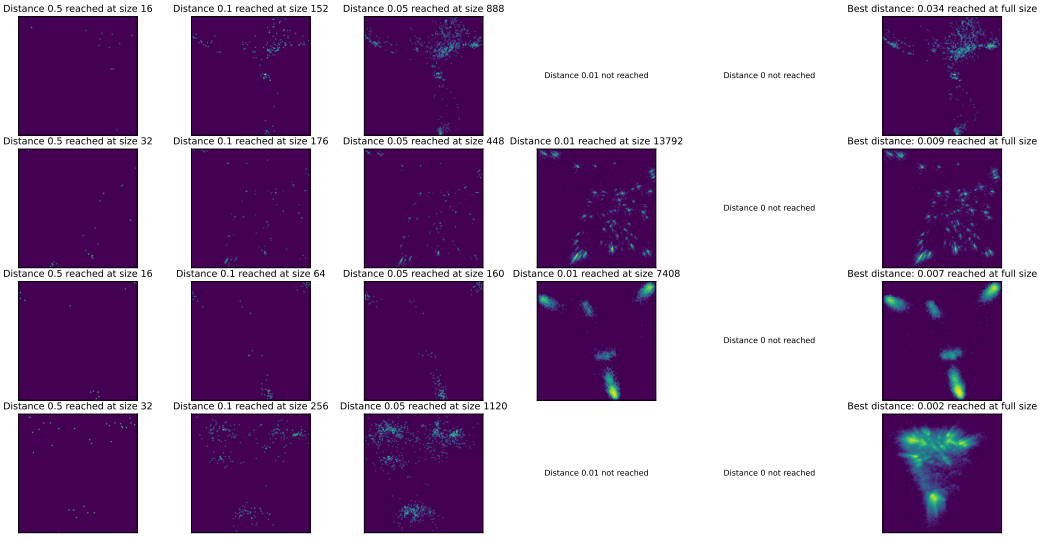


Figure 4.11 – Question Q2: What is the sample size (and resulting point cloud $X = X_1 \cup X_2$) required to approximate an unknown population at a prescribed tolerance $H_{2,r}(X_1, X_2)$? From top to bottom: Atlas Guyane, 10V-RbcL, Long Reads A, S5 (L1-L10). The actual S5 sample is used (no substitute). Resampling X in two new samples X_1 and X_2 ($X = X_1 \cup X_2$). In the rightmost column, evaluation of the full sample.

Figure 4.11 shows the resulting point cloud $X = X_1 \cup X_2$, resampling X in two new samples X_1 and X_2 ($X = X_1 \cup X_2$), with the aim to achieve a relative Squared Modified Hausdorff $H_{2,r}(X_1, X_2)$ equal to 0.5, 0.1, 0.05, 0.01 in columns 1, 2, 3 and 4, respectively. The rightmost column (5) also shows the full sample. We recall the estimate for the full sample obtained with the *a posteriori* estimate associated with Question Q1 was consistently zero and did not provide a plus-value. On the contrary, the new estimate associated with Question Q2 is able to attribute an *intrinsic* value to a sample. As a consequence we are able to evaluate the respective full samples. Their estimates are equal to 0.034, 0.009, 0.007, and 0.002 for the Atlas Guyane, 10V-RbcL, Long Reads A, S5 (L1-L10) datasets, respectively. We assessed both the full S5 (L1-L10) dataset and the 120,000 Lref sample from Chapter 3. We obtained an estimate of 0.002 (reported in the figure) and 0.01 (not reported in the figure) for the full S5 and Lref samples, respectively.

4.4 A class of iterative algorithms

As we mentioned earlier, the *full* algorithms of sections 4.2 and 4.2 are not practical because they require us to have computed the full reference sample X. In the case of Question Q1, we would still have to compute the full reference sample in order to compare it with the reduced sample $X^{(l)}$ under consideration at iteration l in order to obtain the estimate. However, we

do not need to access the full reference sample to answer Question 2, which makes the *full* algorithm overkill.

We therefore propose a class of more effective iterative algorithms. Algorithm 15 presents their sketch. They follow the iterative design of the *full* algorithms of sections 4.2 and 4.3. However, they do not extract the sample $X^{(l)}$ (or X for short) under consideration at iteration l from a reference sample $Xref$ (the overkill). Instead, they compute a reduced MDS on a m_s by m_s principal submatrix D_s of D where m_s is the size of the sample X under consideration. We have implemented both variants answering questions Q1 and Q2 but we restrict our presentation to algorithms that answer Question Q2 for the sake of conciseness.

All the algorithms follow the same resampling and distance computation step (lines 6 and 7, respectively, in Algorithm 15) as discussed in Section 4.3. However, they differ in how they compute the MDS for a given iteration (lines 3, 4, 5). We first propose conservative algorithms in Section 4.4.1.1. They compute an MDS at an iteration $l+1$ without relying on the MDS of iteration l. We call such algorithms *redundant*. We then propose *progressive* algorithms in Section 4.4.2. They compute an MDS at an iteration $l+1$ by extending the MDS of iteration l.

Algorithm 15: Iterative uniform sampling MDS algorithm without a reference sample.

Input: D an $m \times m$ dissimilarity matrix, ϵ the desired accuracy, $\#s$ the number of new samples for resampling, m_s the starting size of the submatrices
Output: X a (reduced) sample point cloud

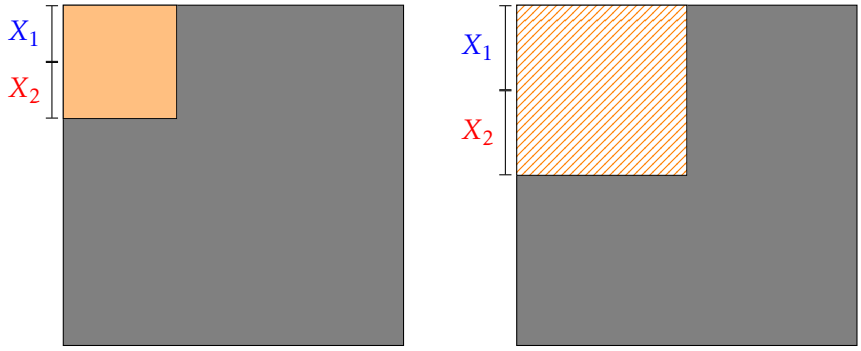
1 **do**
2 Iterate m_s to the next size
3 Extract a m_s by m_s principal submatrix D_s of D
4 Form the Gram matrix G_S associated with D_s
5 Compute the MDS out of G_S to get X
6 Split X into $\#s$ point clouds $X_1, X_2, ..., X_{\#s}$ // Resampling
7 Compute d the mean of the distances $\{d(X_i, X_j)\}_{i \neq j}$
8 **while** $d > \epsilon$
9 **return** X

4.4.1 Redundant Algorithms

We first propose two conservative approaches, namely the Combined (Section 4.4.1.1) and Landmark Procrustes (Section 4.4.1.2) algorithms. They compute an MDS at an iteration $l+1$ without relying on the MDS of iteration l.

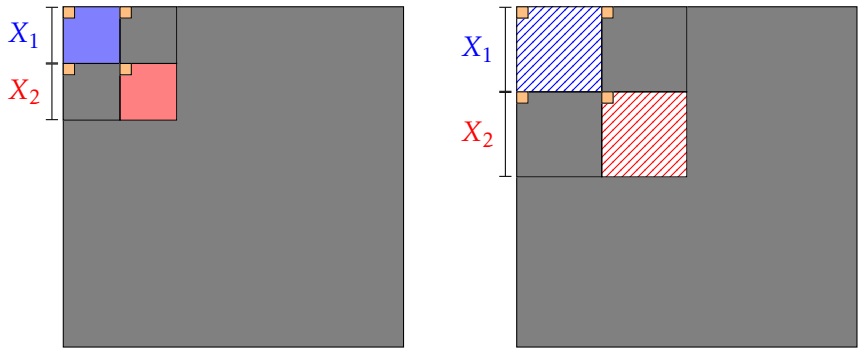
4.4.1.1 Combined

The first algorithm we consider is the Combined approach. It follows the same approach as presented in Figure 3.17 (page 81), as well as being precisely the method presented in Algorithm 15. The progression through the iterations is described in Figure 4.12. In this figure, we present in solid blocks the computations done at iteration l (Figure 4.12a) and in dashed blocks the computations done at iteration $l+1$ (Figure 4.12b). Since we discard the results between iterations, there is no solid block on the right part.



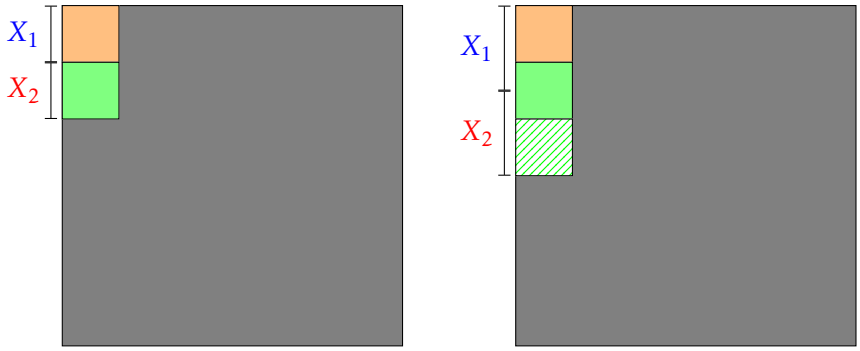
(a) Iteration l (b) Iteration $l+1$

Figure 4.12 – Computation of the Combined redundant algorithm at iterations l and $l+1$. Solid blocks are computed at step l while dashed blocks are computed at step $l+1$. Parts in grey are not accessed.



(a) Iteration l (b) Iteration $l+1$

Figure 4.13 – Computation of the Landmark Procrustes redundant algorithm at iterations l and $l+1$. Solid blocks are computed at step l while dashed blocks are computed at step $l+1$. Parts in grey are not accessed. Parts in orange are used as landmarks.



(a) Iteration l (b) Iteration $l+1$

Figure 4.14 – Computation of the LMDS progressive algorithm at iteration l and iteration $l+1$. Solid blocks are computed at step l while dashed blocks are computed at step $l+1$. Parts in grey are not accessed. Parts in orange are used as landmarks onto which the original partial MDS is performed. Parts in green are used for the interpolation step of the LMDS.

4.4.1.2 Landmark Procrustes

The second algorithm we consider is the Landmark Procrustes algorithm. The concept, presented in Figure 4.13 consists in computing two MDS on diagonal blocks, then aligning the result using generalized alignment to some landmarks, as presented in Section 3.6 (page 82) and more precisely in Figure 3.31. While this algorithm relies on an MDS that accesses fewer blocks than the Combined method, we still discard the result between successive iterations. This is represented in Figure 4.13 by the fact that there are no solid blocks in Figure 4.13b.

4.4.2 Progressive Algorithms

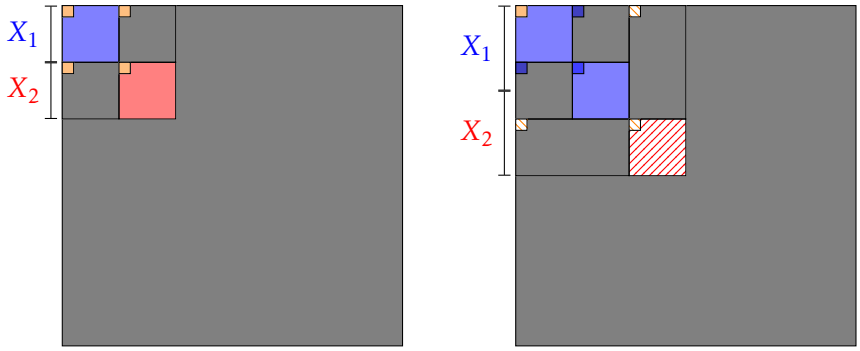
We now present two progressive algorithms, namely the LMDS (Section 4.4.1.1) and BDLPMDS (Section 4.4.1.2) algorithms. They compute an MDS at an iteration $l+1$ by extending the MDS of iteration l.

4.4.2.1 LMDS

The first progressive algorithm we propose is based on LMDS (see Section 3.7.1, page 84). With this iterative version presented in Figure 4.14, we do not compute the full interpolation step, but instead we increment it over each iteration. In this algorithm, we compute the MDS on the landmarks during the first iteration and presented in orange in Figure 4.14a. During each successive iteration, we apply the interpolation step to only part of the remaining points.

4.4.2.2 BDLPMDS

The last scheme we propose is the BDLPMDS algorithm. It is a progressive variation of the Landmark Procrustes version from Section 4.4.1.2 where we reuse the result from one iteration to the next. In the first iteration, we select two submatrices as well as a number of landmarks, we compute the MDS on them and align them using Procrustes analysis. When we go to the next iteration, we use this rebuilt point cloud as one of the two samples to consider during the next iteration. In Figure 4.15a, we compute the MDS on both the blue and the red submatrices. Then on the next iteration presented in Figure 4.15b, this result becomes the solid blue matrix,



(a) Iteration l (b) Iteration $l+1$

Figure 4.15 – Computation of the BDLPMDS progressive algorithm at iteration l and iteration $l+1$. Solid blocks are computed at step l while dashed blocks are computed at step $l+1$. Parts in grey are not accessed.

while the new sample to be added to the result in presented in the dashed red block. While this method is fast to compute, it is not necessarily stable, as we align all point clouds to the first one computed, and therefore the principal axes may not be the ones expected. This is showcased in Section 4.5.

4.5 Numerical evaluation of the iterative MDS algorithms

To evaluate the algorithms presented in Section 4.4, we performed experiments on the four datasets presented in Section 1.8 (page 19). The $120,000 \times 120,000$ Lref submatrix of S5 (L1-L10) presented in Section 3.8.4 (page 92) is used as a substitute to the full S5 (L1-L10) sample when computing the distance to the reference sample for answering Question 1. We recall (see Table 3.4, page 95) that this sample is a good approximation of the full dataset.

In these experiments, for each of the proposed algorithms, we studied the evolution of the distance with regards to the size of the submatrices taken into account, the distance used, the number of dimensions taken into account in the distance comparisons for 8 submatrices between each other as well as between the submatrices and the full matrix. For submatrices of size below 5,000 (150 for Atlas Guyane) we use a full SVD for the MDS, while we use RSVD for larger matrices.

We also consider the *full* algorithm described in Section 4.2 and 4.3 as a reference. We only present results for the relative distances for the sake of brevity, but additional results using absolute distances can be found in Appendix C.1.

4.5.1 Evaluation of the redundant algorithms

In figures 4.16, 4.17, 4.18 and 4.19, we present for the datasets Atlas Guyane, 10V-RbcL, Long Reads A and Lref respectively, the evolution of the distances of 8 submatrices between each other and between the submatrices and the original matrix using the combined and landmark Procrustes approach and compare them with the **full** approach. In these figures, we can see that both approaches give very similar results in 2 dimensions, while the Landmark Procrustes approach seems to lose some accuracy in 10 dimensions. The combined method gives overall better results, in fact almost always as good as the **full** approach while the Landmark Procrustes method sometimes has a higher error. This behavior is not surprising as the

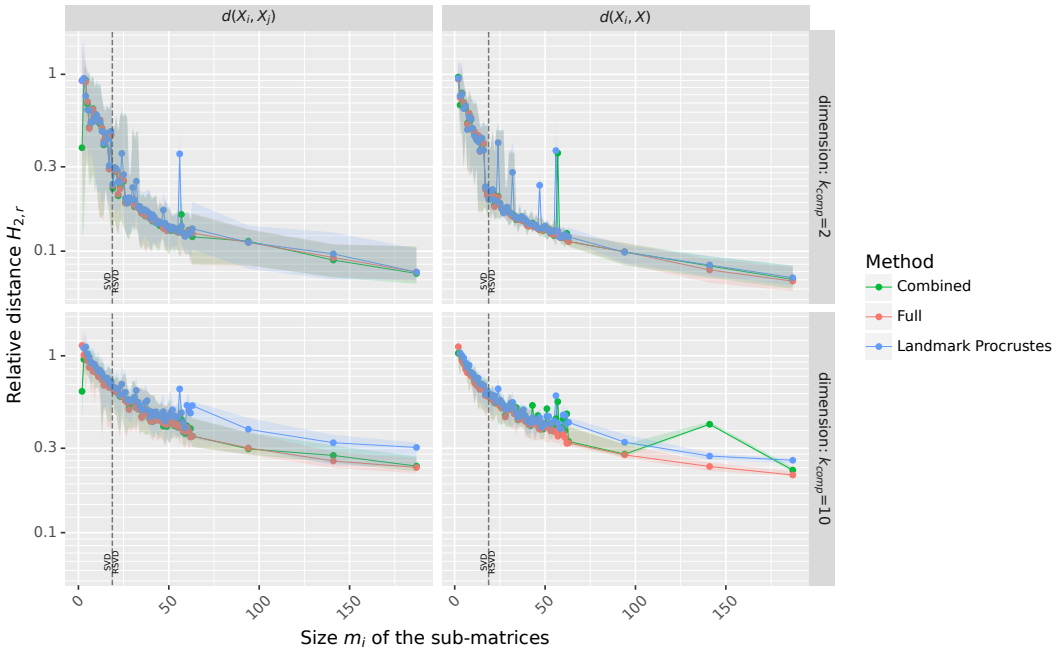


Figure 4.16 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original Atlas Guyane sample (right). Each color represents a method among Full, Combined and Landmark Procrustes. Median value is displayed in solid line.

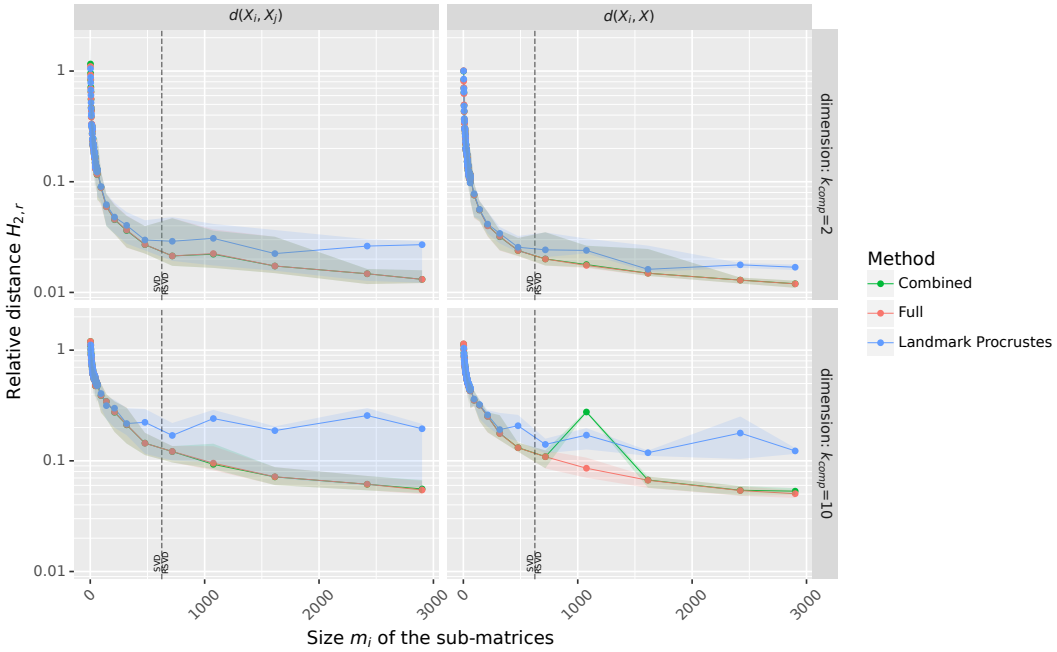


Figure 4.17 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original 10V-RbcL sample (right). Each color represents a method among Full, Combined and Landmark Procrustes. Median value is displayed in solid line.

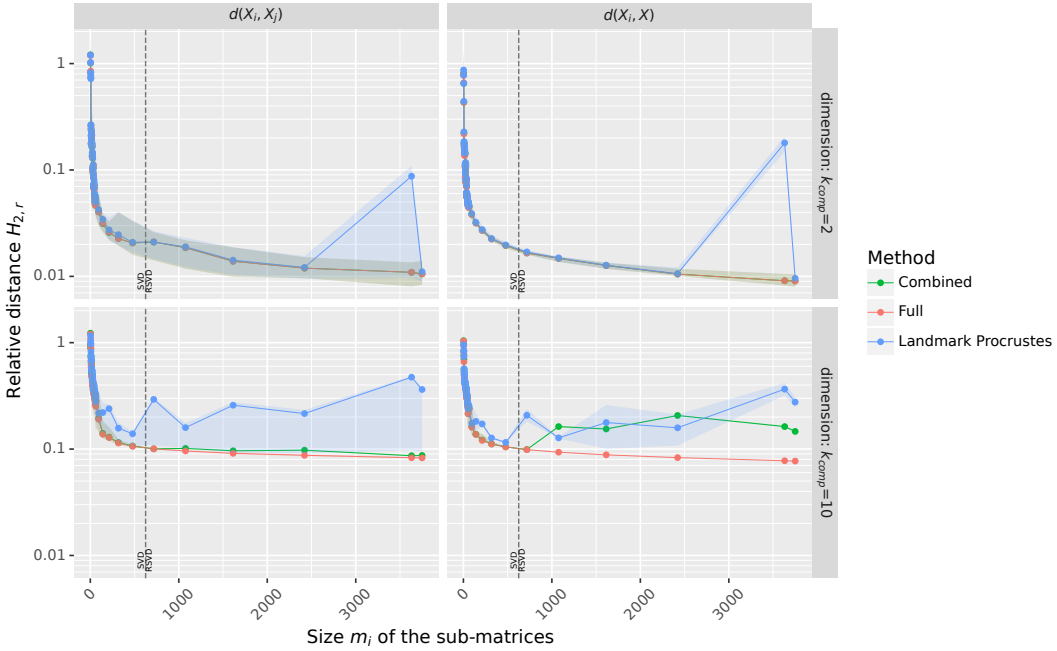


Figure 4.18 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original Long Reads A sample (right). Each color represents a method among Full, Combined and Landmark Procrustes. Median value is displayed in solid line.

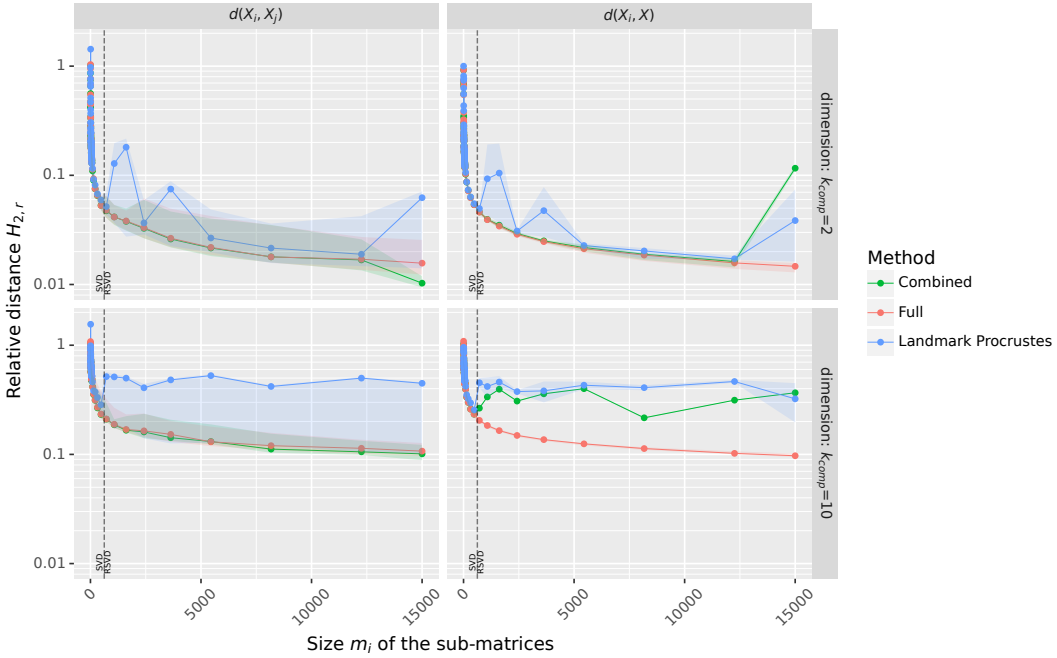


Figure 4.19 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original Lref sample (right). Each color represents a method among Full, Combined and Landmark Procrustes. Median value is displayed in solid line.

combined approach computes a MDS of the complete submatrix extracted while the Landmark Procrustes tries to approach it using two smaller MDSs. In 10 dimensions, the Landmark Procrustes approach shows some limitations, while the combined approach continues to give results similar to the **full** method in the comparisons between the resampled point clouds.

4.5.2 Evaluation of the progressive algorithms

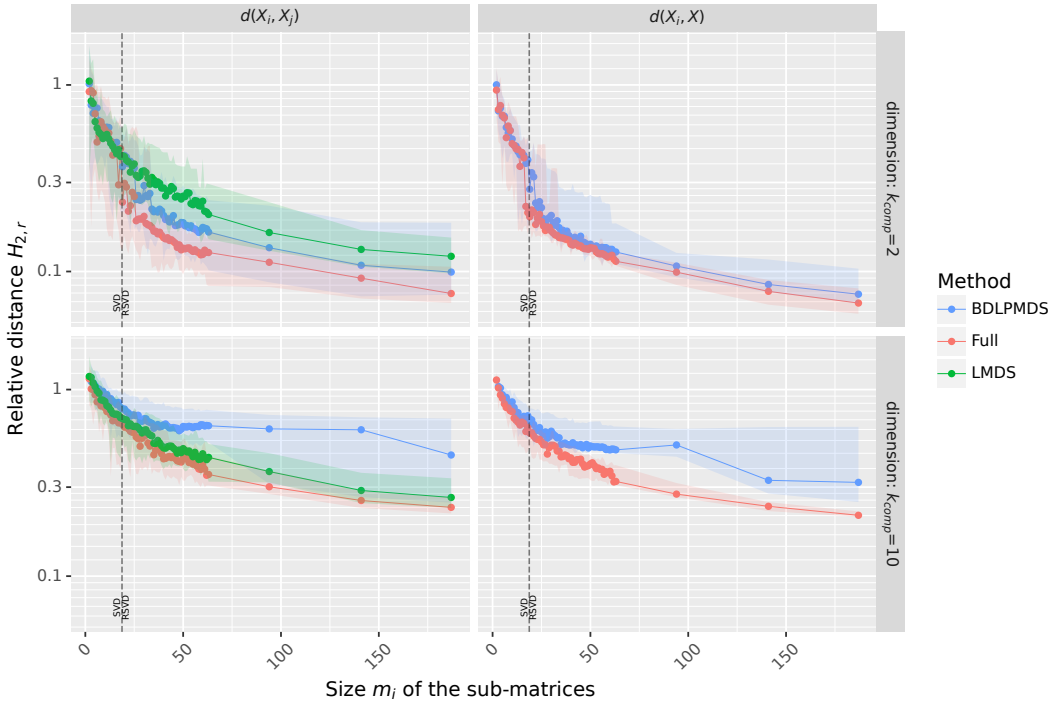


Figure 4.20 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original Atlas Guyane sample (right). Each color represents a method among Full, BDLPMDS and LMDS. Median value is displayed in solid line.

In figures 4.20, 4.21, 4.22 and 4.23, we present for the datasets Atlas Guyane, 10V-RbcL, Long Reads A and Lref respectively, the evolution of the distances of 8 submatrices between each other using the LMDS and BDLPMDS approaches. The distances between the submatrices and the original matrix are only available for the BDLPMDS method. As explained in Section 4.4.2.2, the BDLPMDS approach can lead to inaccuracies because the point cloud are being aligned against the one extracted at the first iteration, as shown in dataset 10V-RbcL in Figure 4.21 in 2 dimensions as well as in all cases when using 10 dimensions. Similar to the results for the redundant methods, the experiments show that our methods do not perform optimally in 10 dimensions.

The LMDS approach seems to always have a slightly higher distance than the full approach. This could be caused by the setup of our experiments, where we followed the design presented in Figure 4.14, in which after the computation the landmarks (in orange) will always end up in X_1, when a version where the elements are randomized may make more sense.

4.6 Discussion on performance

Figures 4.24, 4.25, 4.26 and 4.27 present for the datasets Atlas Guyane, 10V-RbcL, Long Reads A and Lref respectively, the computation time for each method (Combined, Landmark Pro-

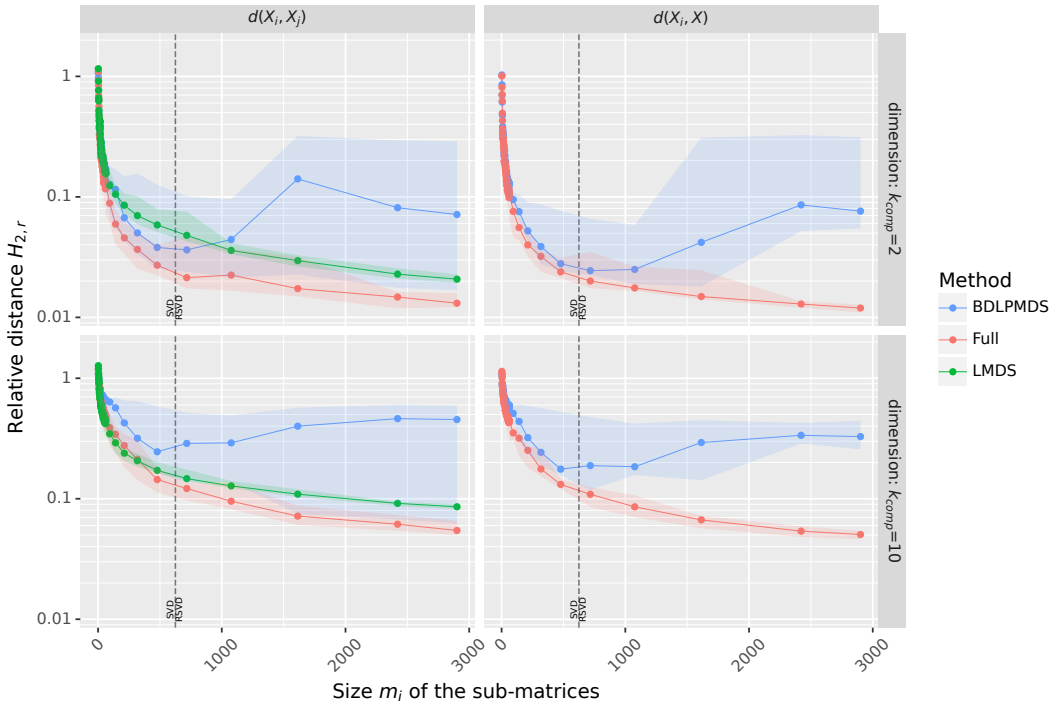


Figure 4.21 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original 10V-RbcL sample (right). Each color represents a method among Full, BDLPMDS and LMDS. Median value is displayed in solid line.

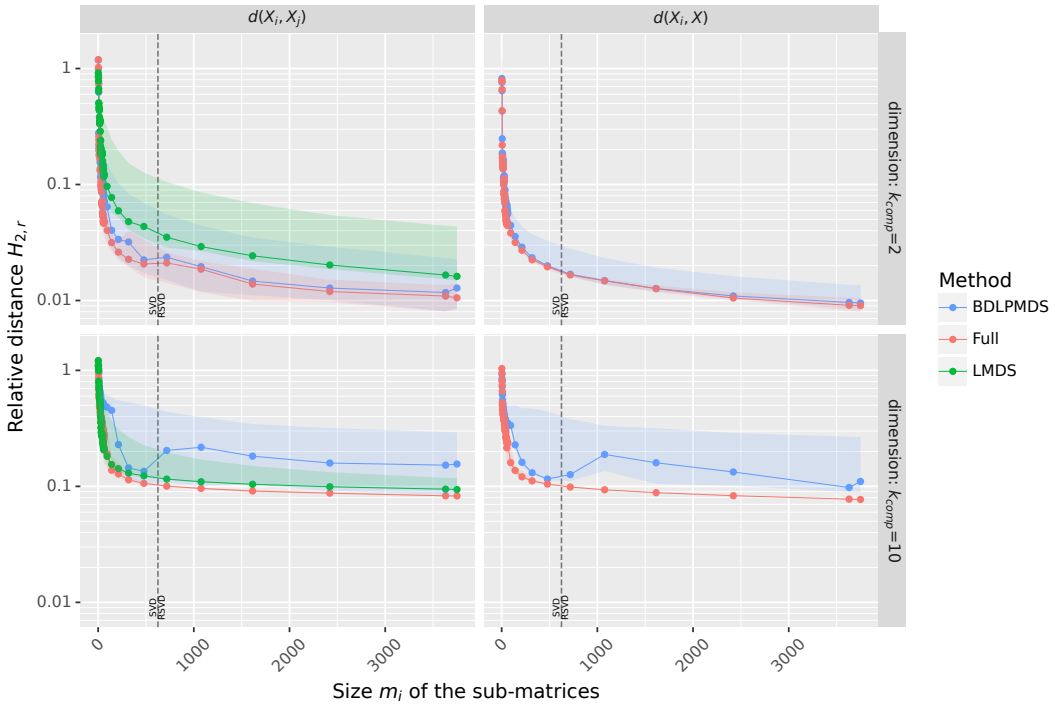


Figure 4.22 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original Long Reads A sample (right). Each color represents a method among Full, BDLPMDS and LMDS. Median value is displayed in solid line.

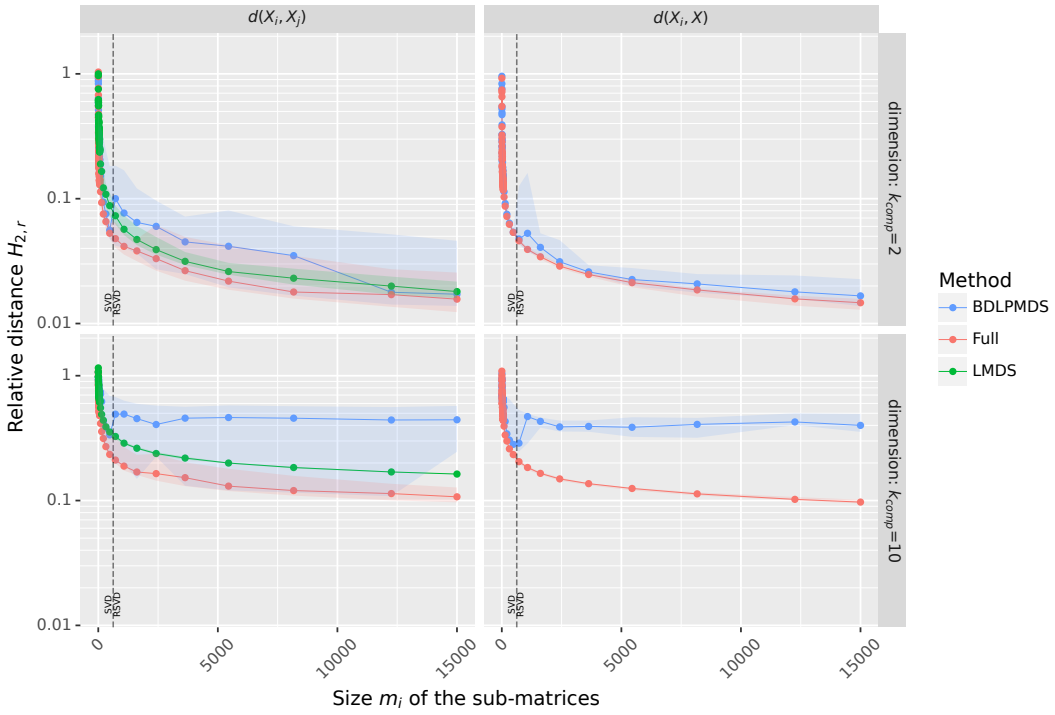


Figure 4.23 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original Lref sample (right). Each color represents a method among Full, BDLPMDS and LMDS. Median value is displayed in solid line.

crustes and BDLPMDS) with respect to the size of the extracted submatrices, presented in log-y scale with both nonlog-x scale (top) and log-x scale (bottom). When computing the LMDS, we first performed the full LMDS and then extracted the necessary rows of the matrix at each size, which means that we cannot get a relevant computation time for this method here. We can see that the combined version is the most expensive method, which is expected as it needs to compute an SVD twice as large as the one from the landmark Procrustes approach. In Section 1.6.3 (page 18), we showed that the complexity of RSVD-MDS is asymptotically $\mathcal{O}(m^2)$ so we expect the combined method to take twice as long to compute than the landmark Procrustes method. The BDLPMDS is faster than both approach due to its progressive nature. This also explains why the last iteration is faster than the previous one, since the size of the samples considered is multiplied by 1.5 at each iteration until reaching the end, the last size considered each time is comparatively less expensive.

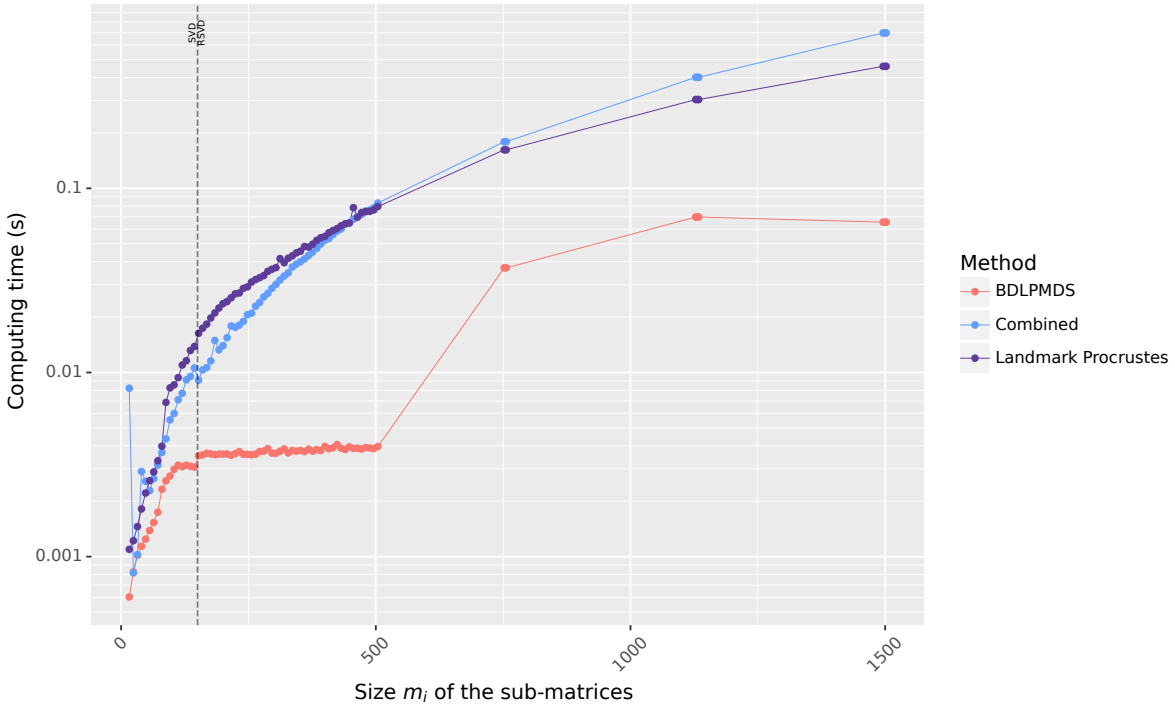
4.7 Conclusion

In this chapter, we discussed and implemented methods to construct reduced MDS, i.e. a subset of the points in the cloud using randomized *uniform sampling*. We discussed two questions that arise from sampling the distance matrix in the context of MDS:

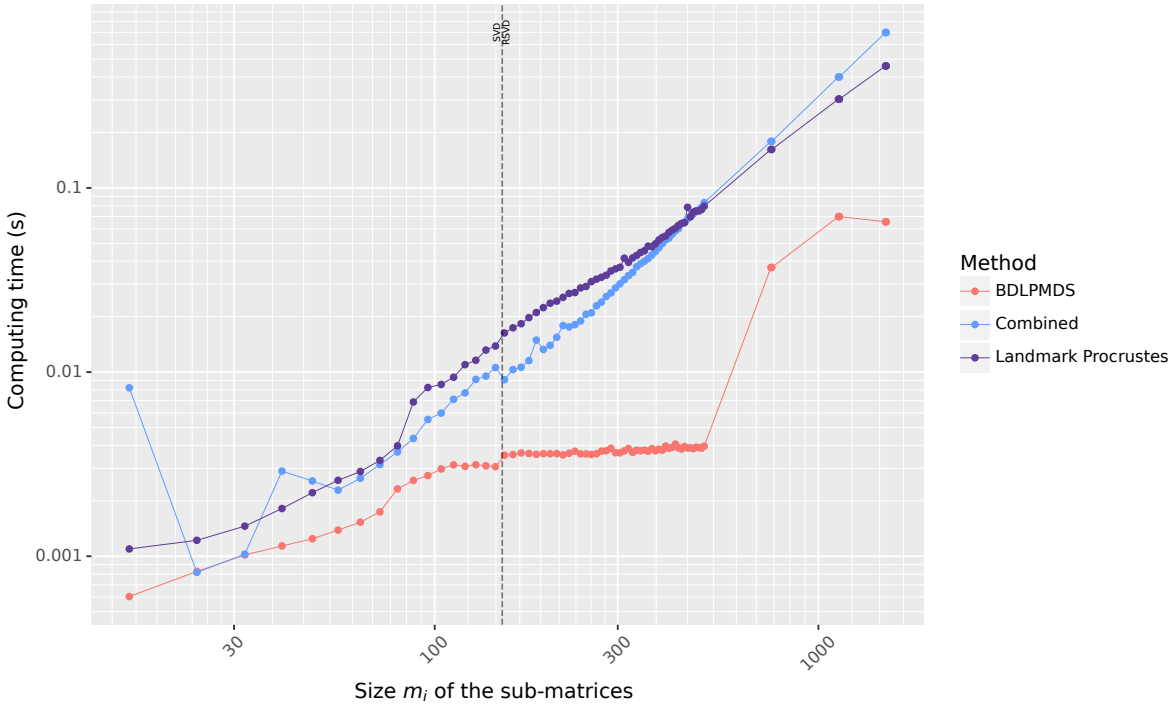
— Question Q1 - How to approximate a reference input sample ?
— Question Q2 - How to estimate the intrinsic quality of a sample ?

Q1 can be seen as the issue of iterative uniform sampling with a reference sample while Q2 corresponds to iterative uniform sampling without a reference sample.

To answer these questions, we iteratively extracted point clouds from the reference sample which were further resampled and compared, using the distances considered in Chapter 3, to

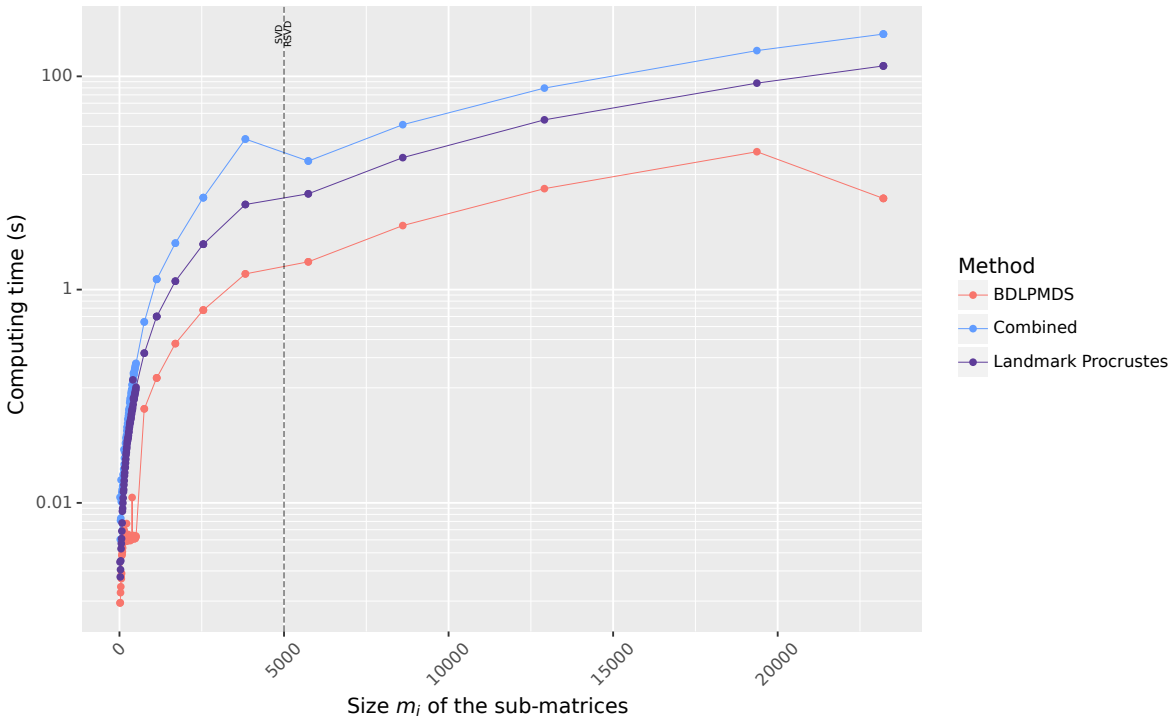


(a) Nonlog x-scale.

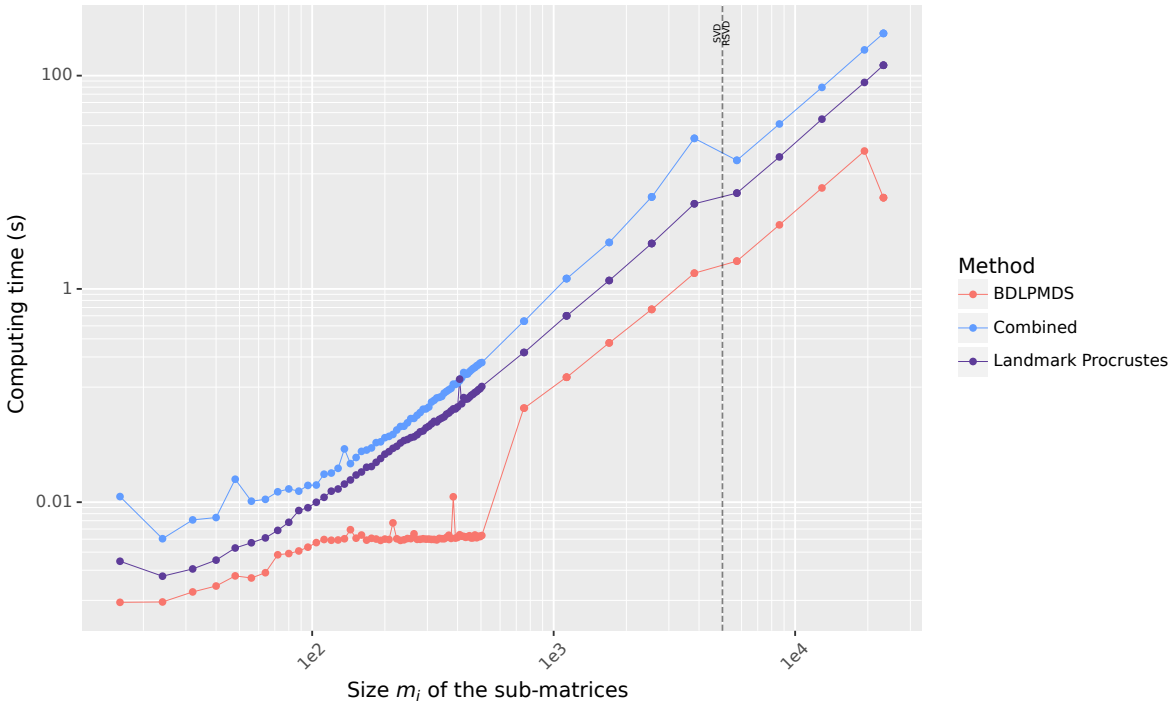


(b) Log x-scale.

Figure 4.24 – Computation time for Combined, Landmark Procrustes and BDLPMDS on dataset Atlas Guyane.

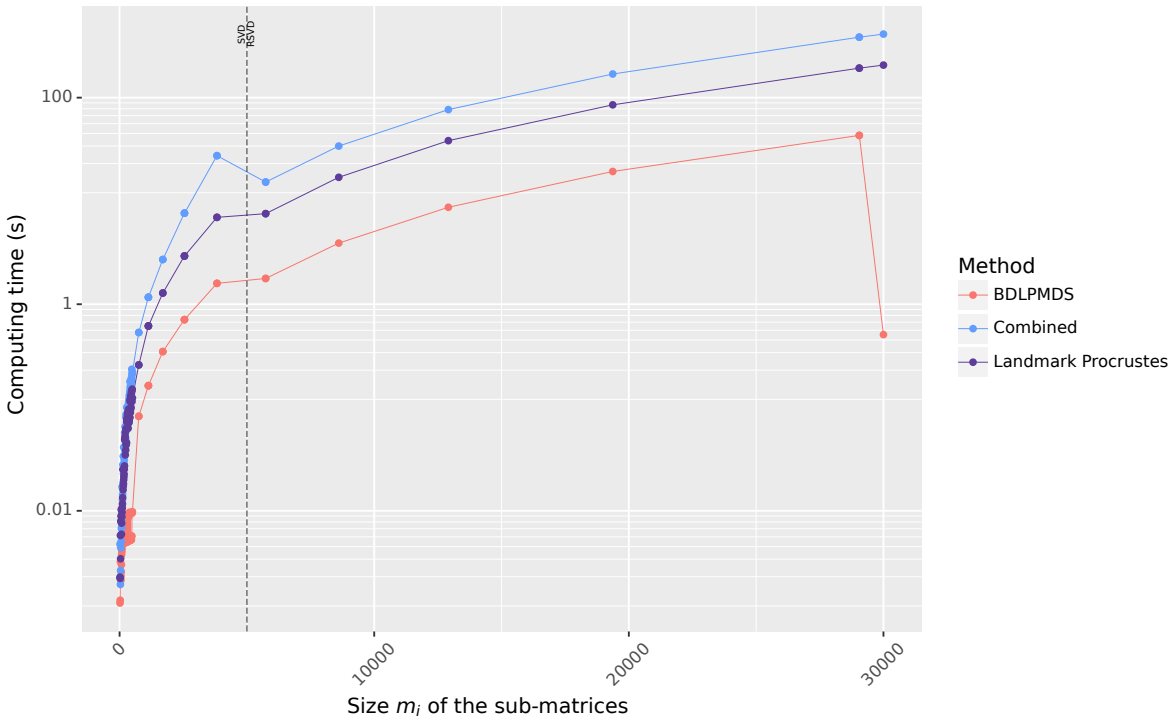


(a) Nonlog x-scale.

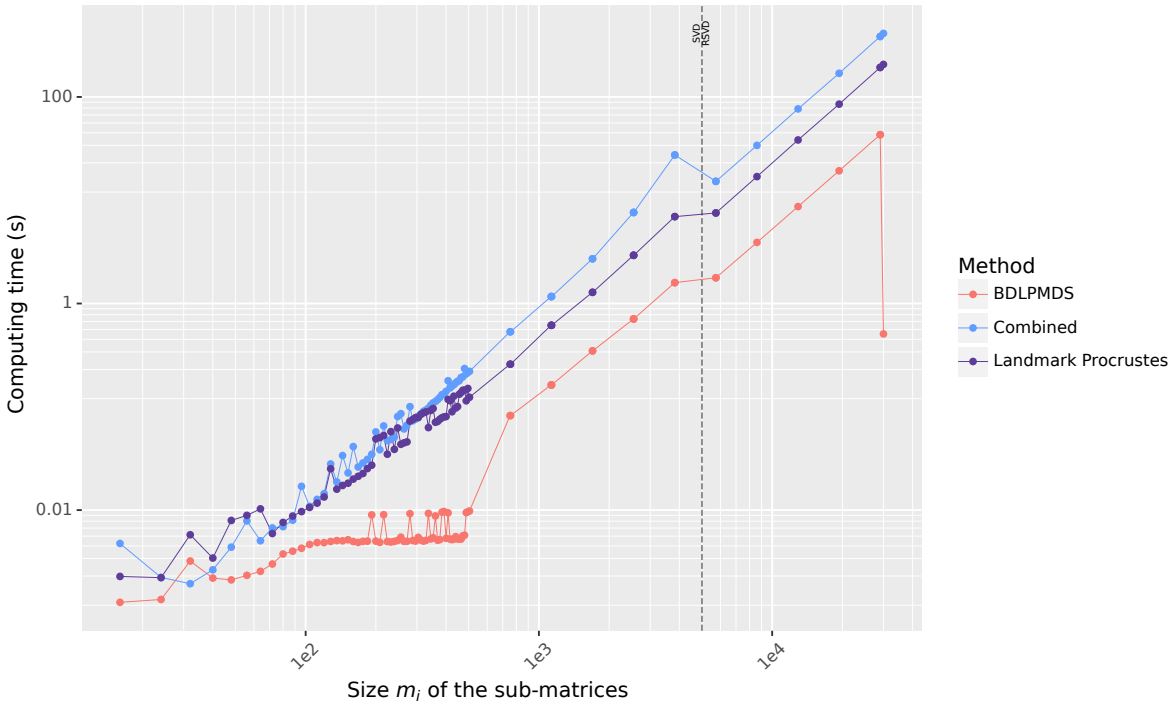


(b) Log x-scale.

Figure 4.25 – Computation time for Combined, Landmark Procrustes and BDLPMDS on dataset 10V-RbcL.

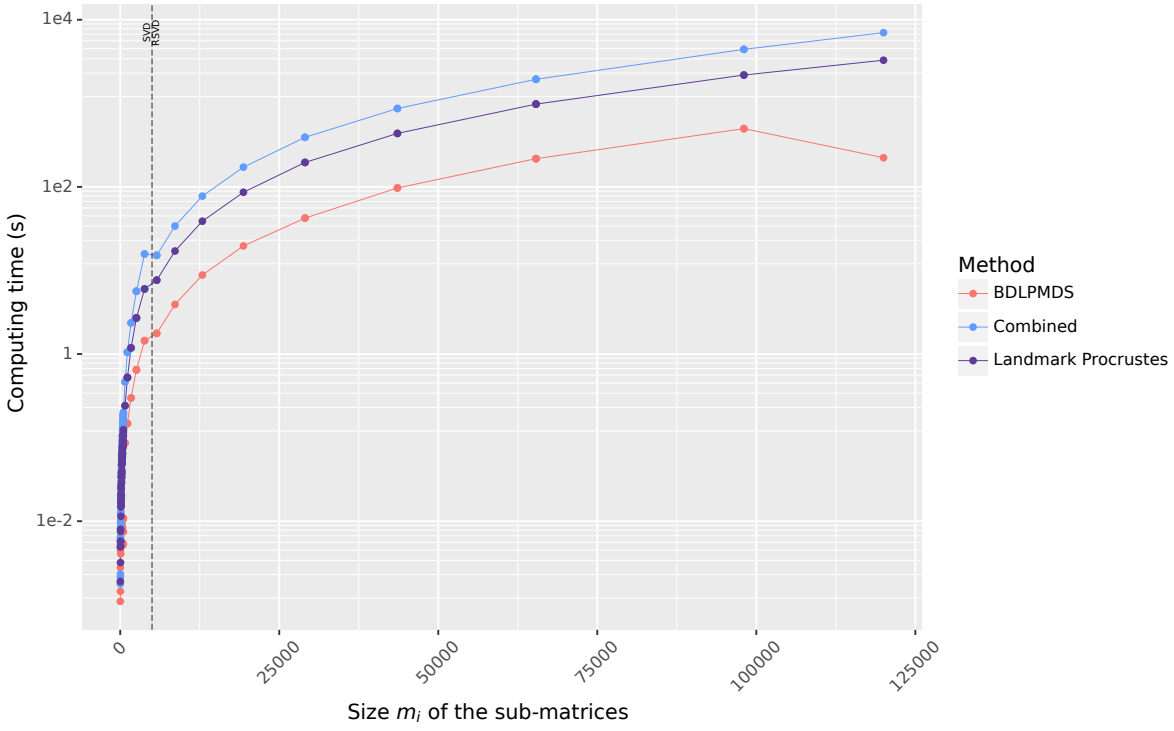


(a) Nonlog x-scale.

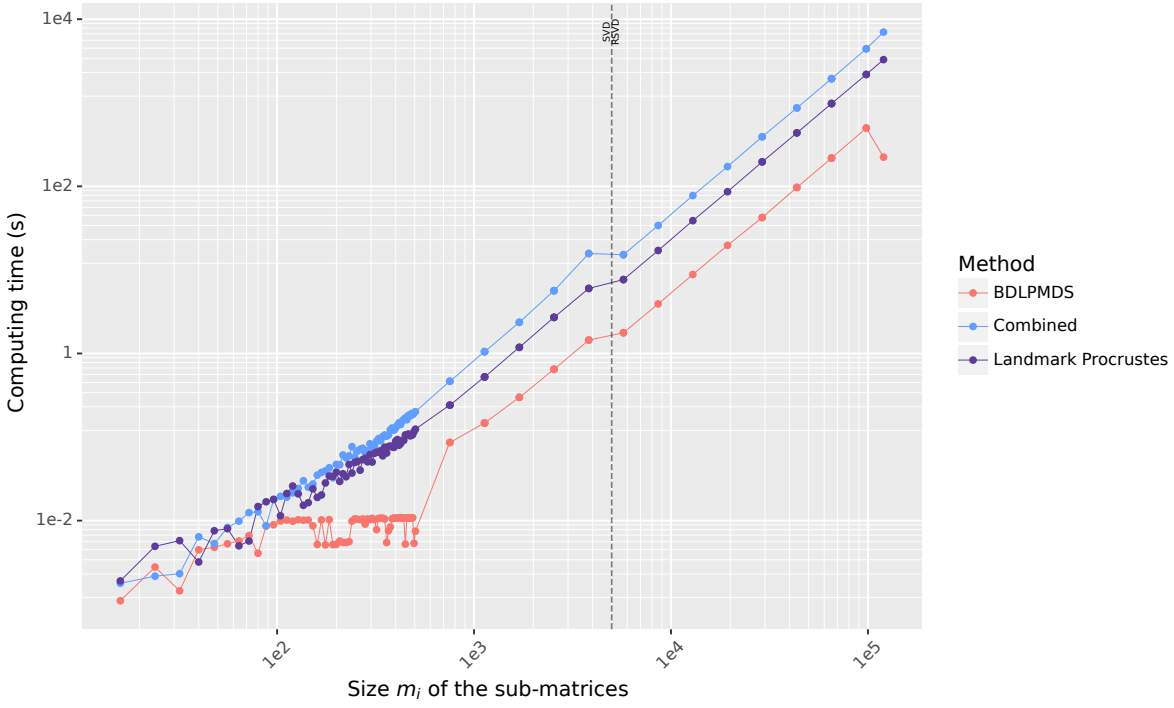


(b) Log x-scale.

Figure 4.26 – Computation time for Combined, Landmark Procrustes and BDLPMDS on dataset Long Reads A.



(a) Nonlog x-scale.



(b) Log x-scale.

Figure 4.27 – Computation time for Combined, Landmark Procrustes and BDLPMDS on dataset Lref.

the reference point cloud (Q1) or between each other (Q2). The most important observation is that the results show a regularly decreasing median distance with iterations for both relative Modified Hausdorff (MHD_r) and relative Squared Modified Hausdorff ($H_{2,r}$) on all datasets. They can therefore be used as *robust a posteriori* estimators for answering both Questions Q1 and Q2. Furthermore, the new estimate associated with Question Q2 is able to attribute an *intrinsic* value to a sample. As a result, we are able to evaluate the full samples of our datasets.

Based on these results, we proposed iterative MDS algorithms that use a stopping criterion based on the distance between the resampled point clouds at each iteration. We proposed four algorithms, each following the same resampling and distance computation step and differing only in how they compute the MDS at a given iteration. First, we proposed two conservative algorithms that compute each iteration without relying on the previous results, the Combined and Landmark Procrustes algorithms. We showed that the Combined algorithm reliably gave results very close to the reference method while the Landmark Procrustes algorithm had a slightly higher error. Then, we presented two progressive algorithms based on LMDS and BDLPMDS, that aim to build the next iteration by extending the result of the previous one, and both show promising results.

Regarding future work, while we have pointed out the limitations of the BDLPMDS algorithm, it still performed correctly in 3 of the 4 test cases. We expect that a revised version of this algorithm will provide a fast and reliable method for computing iterative MDS.

As we mentioned earlier, the two questions Q1 and Q2 can be linked. We plan to design an algorithm that can decide when to stop scanning the data (in the sense of Question Q2) using uniform sampling in order to obtain an intermediate reference sample, and then to compress that data further. This additional step corresponds to answering Question Q1 with the intermediate reference sample as input. Since we know that Question Q1 is generally best answered with importance sampling, we could perform the second step with importance sampling.

Chapter 5

Approximately coincident input distance datasets

5.1 Introduction

The landmark technique used in chapters 3 and 4 requires the computation of some off-diagonal elements in addition to large diagonal blocks. In this chapter, we investigate whether it is possible to construct a reduced MDS of a sample, without having to access any off-(block-)diagonal elements at all.

Here, we only consider point clouds that are approximately coincident, *i.e.* we expect these point clouds to already be close because they were sampled from the same dataset. In this case, we may want to validate how close they are using the Hausdorff distances presented in Section 3.2.1 (p. 68) in order to verify how well they approximate the global sample in a similar way to the methods described in Chapter 4.

When the point clouds become very close, the non-alignment problems presented in Chapter 3 begin to be corrected by the sampling: as long as the eigenvalues of the reference dataset are disjoint, which is shown to be the case for the data we consider in Section 1.8 (p. 19), then the sEVD will capture the same principal axes and there will be no axis permutation or rotation, solving **issues 1 and 2**. In the same way, a sampling of sufficient quality will statistically give point clouds that are well centered on each other, solving **issue 3**. In this case, only **issue 4** (reflection around the principal axes) remains. This last problem is caused by the non-unicity of the sEVD: given Q, Λ the sEVD of a matrix, it is possible to construct more solutions by changing the signs of the eigenvectors. These different solutions, in the context of MDS, result in point clouds that are identical up to a certain number reflections around any number of principal axes.

If the only transformations we need to consider to match our point clouds are these reflections, it may be possible to consider every possible combination, compute the Hausdorff distance, and keep the lowest one. This would have the advantage of not having to access any off-(block-)diagonal elements at all, and allow the complete matrix to be reconstructed using the data access pattern presented in Figure 5.1. This may be desirable when the off-(block-)diagonal elements are not available, e.g. in the cases where they are too expensive to compute.

This method is effective in a small number of dimensions (less than 10) before it becomes computationally prohibitive. The complexity of such an approach is exponential in the number

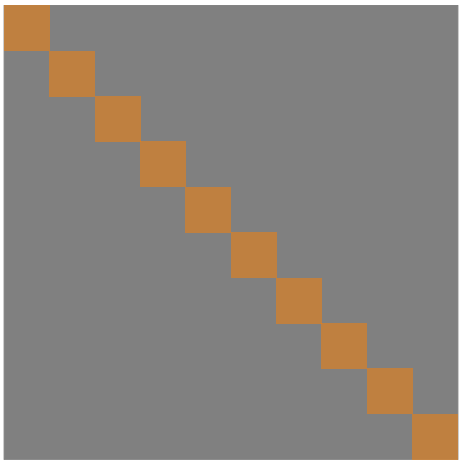


Figure 5.1 – MDS high-level data access pattern (up to a symmetric permutation) to the dissimilarity matrix D (in grey) Only orange blocks need to be accessed.

of dimensions to consider: there are 2^n possibilities for n dimensions.

In this chapter, we propose two strategies for comparing point clouds under these conditions. In Section 5.2, we introduce the concept of exploring the possibility of finding the exact set of reflections to apply to one point cloud in order to align it on the other. We consider a number of heuristics to speed up this computation, at the cost of introducing the possibility of **false negatives**, *i.e.* results where the heuristic has made a mistake and has not found the optimal transformation, making the point clouds appear to have a greater distance than they actually have.

In Section 5.3, we introduce a strategy we call folding. By removing the signs from all coordinates of a point cloud, we obtain a result that is independent of any reflection around the axes, and which makes the point clouds appear folded around the principal axes as illustrated in Figure 5.7, hence the name. There are two drawbacks to this approach: first, it is destructive, *i.e.* it can be used as a preliminary step before computing the distance, but we lose information about the initial point clouds so this method cannot be used to rebuild point clouds, only during the evaluation of the resampling step. The second issue is that this method may introduce **false positives**, since the folding of point clouds will always reduce the distance between point clouds, although we show in Section 5.3.4 that this effect can be mitigated by using a relative distance.

5.2 Flip method

In this section, we propose a method for aligning point clouds derived from approximately coincident input distance datasets. These point clouds are obtained by sampling a larger dataset, as this method relies on the idea that both point clouds are projected onto the same space through the sEVD step of MDS. In such cases, the non-unicity of the sEVD is the only factor that prevents alignment of the point clouds, as it can produce reflections around any of the principal axes. The goal of this section is to discuss algorithms that are designed to identify the optimal set of reflections to apply to a point cloud in order to align it onto another point cloud.

Consider a point cloud $X \in \mathbb{R}^{m \times k}$ of m points in $\mathbb{R}^k$. In this representation, each row of the matrix corresponds to the coordinates of a point in $\mathbb{R}^k$, while each column represents the coordinate along the corresponding dimension. To perform a reflection around the i^{th} principal

axis, we multiply the i^{th} columns of X by -1. We define a `flip` f as a vector in $\{-1,1\}^k$ that represents a set of reflections around the principal axes. If the i^{th} element of f is -1 then it indicates that we should make a reflection around the i^{th} principal axis. We define $F = Diag(f)$ as the matrix associated with the flip f, where the diagonal entries are 1 or -1 depending on the corresponding entry in f.

To apply a flip f to a point cloud X, for all elements of f equal to -1, we multiply the corresponding column of X by -1. We can note that applying a flip f is equivalent to multiplying the point cloud by the diagonal matrix F associated with f. The problem of aligning two point clouds X_1 and X_2 using flips only is summarized in Equation 5.1 with H standing for any of the Hausdorff distance variations considered in Section 3.2 (p. 67).

$$\min_{F \in Diag(\{1,-1\}^k)} H(X_1, X_2 F) \tag{5.1}$$

Such a minimum is always reachable, since the number of possible flips for a given dimension is finite being 2^k for dimensions k. Because the number of possible flips grows exponentially with the dimension, performing a comprehensive search by trying to evaluate all possible distances associated with all flips can become prohibitively expensive for values of k starting around 10. Therefore, sections 5.2.1 and 5.2.2 are dedicated to alternative heuristics of finding the best solution.

5.2.1 Heuristics

Algorithm 16: Baseline of Perdim algorithm.

Input: A, B: point clouds, k: dimension, $dist$: distance method
Output: $flip$, lists of best flips computed

```
flip ← []
for i ← 0 to k do
    d0 ← dist(A[i], B[i])
    d1 ← dist(−A[i], B[i])
    if d0 < d1 then
        flip[i] ← 1
    else
        flip[i] ← −1
return flip
```

In this section, we propose three heuristics for calculating the best flip to align point clouds. In Algorithm 16, we compute the best reflection for each dimension independently. In this Algorithm, $A[i]$ corresponds to the i^{th} column of A. We call this algorithm **Perdim**. The idea comes from the observations of figures 3.32 and 3.33 (p. 97-98), which seem to show that the symmetries of the coordinates can be inferred independently on each dimension. By computing only one distance per dimension, the overall complexity of this algorithm becomes linear in the number of dimensions. Experimental results show that this method actually performs poorly compared to the others, which means that the information about the coordinates in other dimensions can affect the distance result of a flip.

Algorithm 17: Baseline of Perdim-τ algorithm.

Input: A, B: point clouds, k: dimension, ϵ: threshold, $dist$: distance method
Output: $flip$, lists of best flips computed

```
flip ← []
for i ← 0 to k do
    d0 ← dist(A[i], B[i])
    d1 ← dist(−A[i], B[i])
    if min(d0, d1)/max(d0, d1) > ϵ then
        flip[i] ←?
    else
        if d0 < d1 then
            flip[i] ← 1
        else
            flip[i] ← −1
    Perform comprehensive search for all i such that flip[i] =?
return flip
```

It is possible that the **Perdim** method may fail in cases where there is no clear solution to find out the best reflection around a particular axis. Therefore, we propose a variant of this method, which we call Perdim-τ. The idea, presented in Algorithm 17 is to compute for each dimension the ratio between the smaller and the greater distance computed, if this ratio exceeds a certain threshold, we consider both reflections as providing distances that are too close to each other, and we add this index to the list of indices that need to be computed using a comprehensive search. This algorithm can therefore discriminate the well distinct cases while leaving less work for the subsequent comprehensive search.

Algorithm 18: Baseline of Accumulate algorithm.

Input: A, B: point clouds, k: dimension, ϵ: threshold, $dist$: distance method
Output: $flip$, lists of best flips computed

```
flip ← []
for i ← 0 to k do
    d0 ← dist(A[: i], B[: i])
    A[i] ← −A[i]
    d1 ← dist(A[: i], B[: i])
    if d0 < d1 then
        flip[i] ← 1
        A[i] ← −A[i]
    else
        flip[i] ← −1
return flip
```

The final algorithm we propose involves computing both reflections for each dimension, while considering all the previous dimensions in the optimal solution found so far. This means that for the first dimension we try out both possibilities and keep the best one, then for the next dimension we compute the two dimensional distance taking into account the fact that the first dimension is already aligned. This algorithm, which we call **Accumulate** is described in Algorithm 18. In this algorithm $A[: i]$ represents the first i columns of the matrix A, corresponding to the first i dimensions of the point cloud. At each iteration we compare the distance for both reflection around the i^{th} principal axis, while having the first $(i-1)^{th}$ axis already oriented as

best as possible.

5.2.2 Genetic algorithms

Genetic algorithms are a type of optimization algorithm that mimic the principle of evolution. In genetic algorithms, the solution is viewed as a genome that can be mutated to achieve the desired result. In our problem, the genome is represented by a vector of ones and zeroes indicating which dimensions need to be flipped. The algorithm simulates a number of generations, evaluates them using the fitness function (in our case, the distance measurements), and creates the next generation using variations of the best solutions from the previous generation. Several modifiers are applied, each rooted in the principle of evolution: mutation, where one sequence is changed by a certain amount from the previous generation, and genetic crossover, which mixes different individuals from the previous generation. For this study, we used the Python genetic algorithm library.

5.2.3 Experiments

5.2.3.1 Comparing the different heuristics

In Figure 5.2, we compare the effectiveness of the different heuristics presented in section 5.2, both in terms of accuracy and time to solution. Both Perdim-τ and Accumulate find the same solution as the comprehensive search, at a greatly reduced cost. The Perdim approach found the correct results up to dimension 3 and not beyond. Since its computation is completely independent dimension by dimension, an error in any dimension will impact the rest of the computation. It found the wrong solution in both dimensions 4 and 9. The Perdim-τ approach used a comprehensive search for those two dimensions, since the ratio of the distances was `0.85` and `0.95` for these two dimensions respectively, higher than the `0.8` threshold we used, confirming our hypothesis that this approach only fails when the solution is ambiguous.

The genetic algorithm as we implemented it does not seem to be suitable for this task, as it is both slow to run, and not as accurate as the other methods. However, its execution time does not seem to be affected by the number of dimensions as much as the other methods, so it might become competitive for comparisons with more dimensions than what we consider here. It is also possible that better fine-tuning of the parameters of the algorithm would lead to stronger results.

5.2.3.2 Numerical evaluation of the flip strategy

To evaluate the flip strategy, we perform the same experiments as in Section 4.5 (p. 114), using the same four datasets: Atlas Guyane, 10V-RbcL and Long Reads A from Section 1.8 (p. 19) and Lref (see Section 4.5, p. 114) instead of **S5**. At iteration l, we extract two different submatrices which we align using the flip strategy and construct a point cloud X_l for this iteration. This corresponds to Figure 3.16 (p. 80). We then use resampling by splitting the point cloud in 8 and estimate the quality of the approximation of the true point cloud X by X_l using the distances between the subdivided point cloud.

For each dataset, we studied the evolution of the distance obtained using either the flip strategy with the accumulate heuristic or the **full** method which consists in extracting the point clouds directly from the complete MDS solution, in terms of the size of the submatrices used, the distance used, the number of dimensions used in the distance comparisons for 8 submatrices between each other as well as between the submatrices and the full matrix. For submatrices

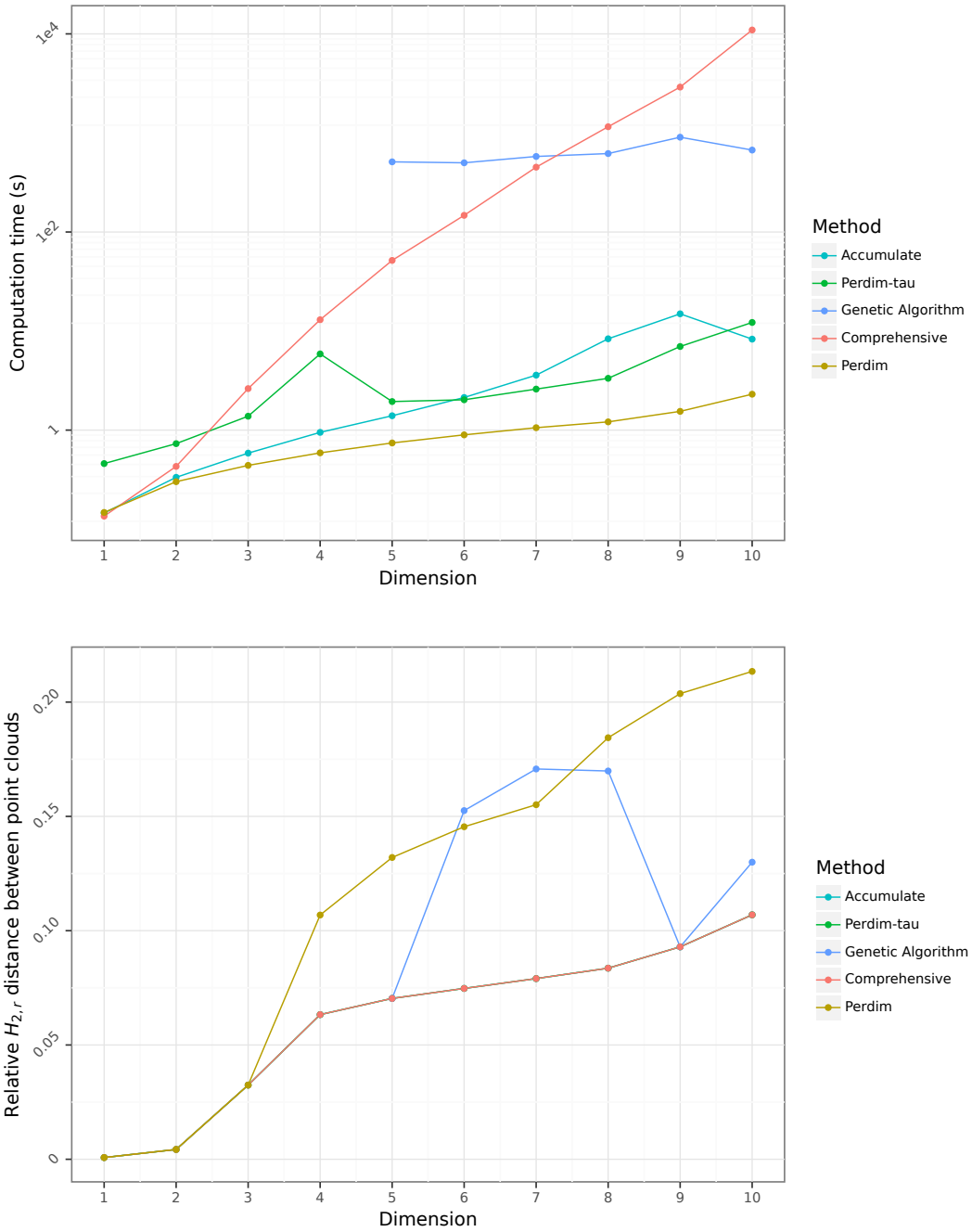


Figure 5.2 – Computation time (top) and relative $H_{2,r}$ distance (bottom) obatined with the different heurisics of flips between point clouds Long Reads B and Long Reads C for dimensions ranging from 1 to 10. The genetic algorithm was only used in dimensions 5 and beyond.

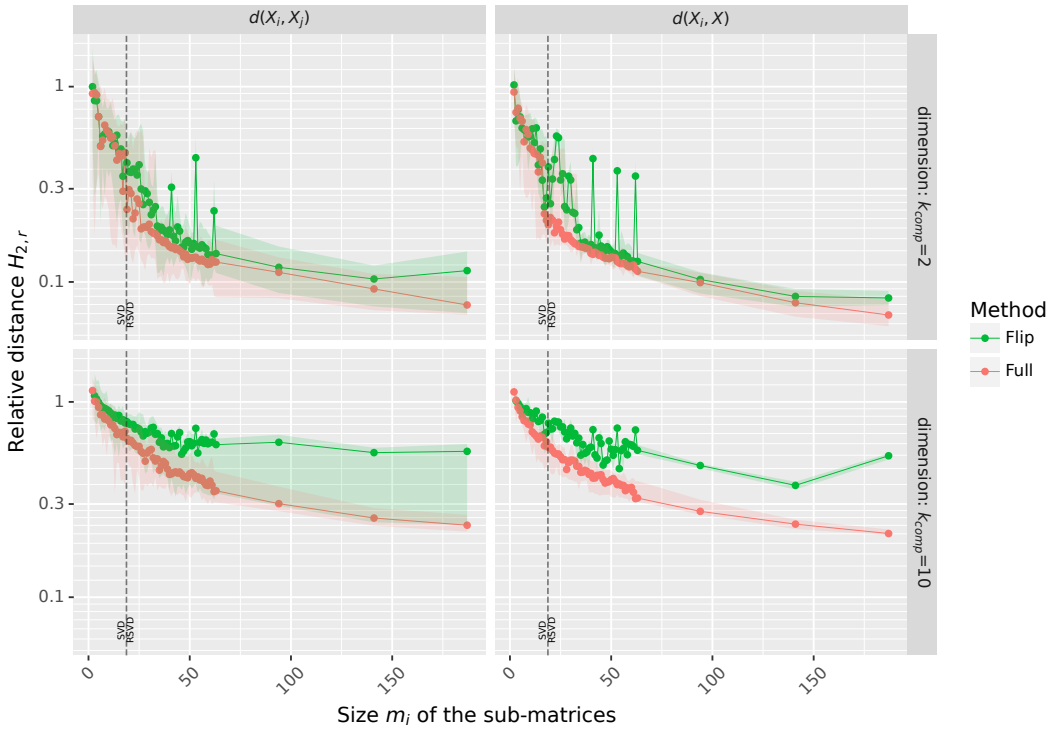


Figure 5.3 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original Atlas Guyane sample (right) aligned using the Flip strategy. Median value is displayed in solid line.

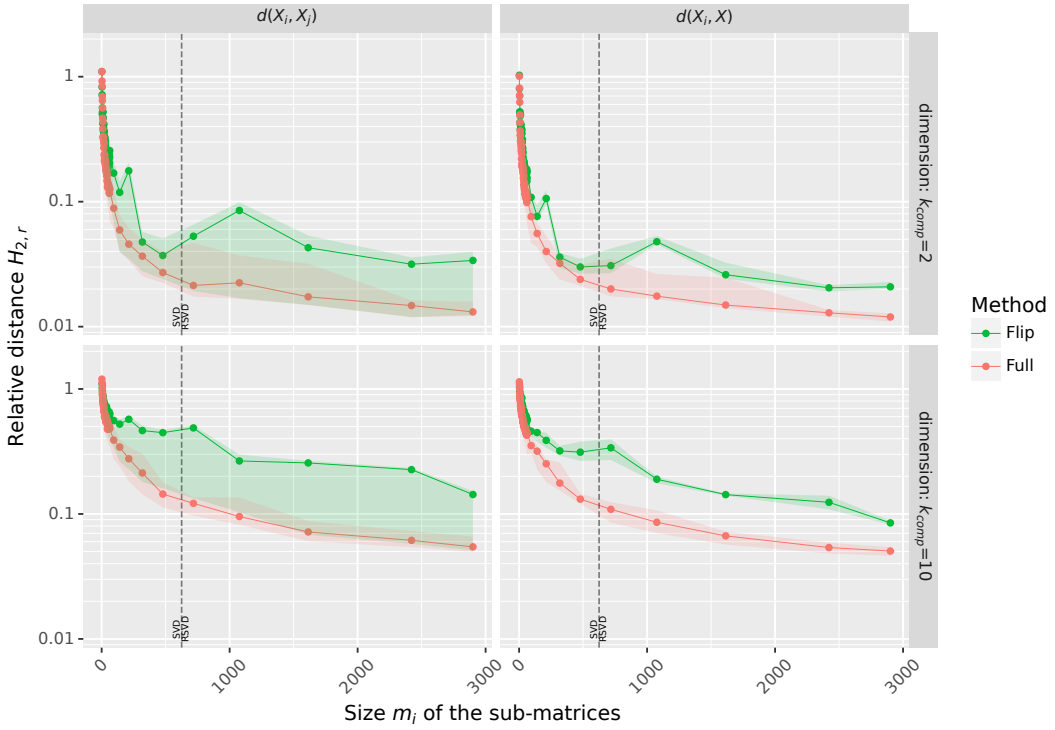


Figure 5.4 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original 10V-RbcL sample (right) aligned using the Flip strategy. Median value is displayed in solid line.

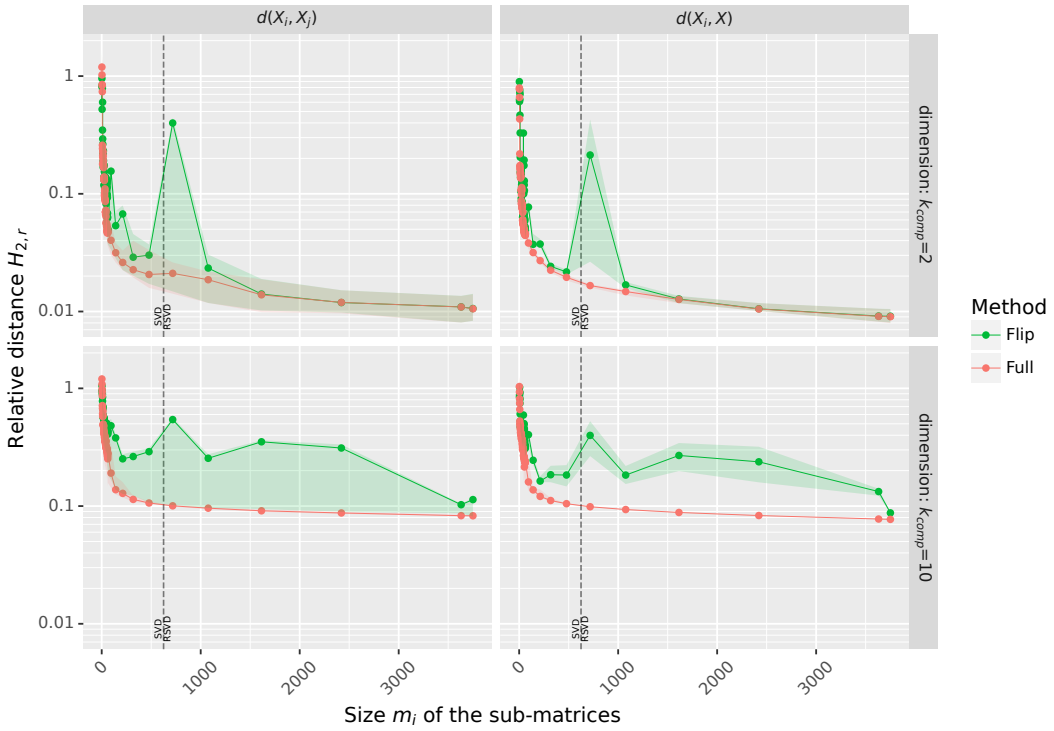


Figure 5.5 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original Long Reads A sample (right) aligned using the Flip strategy. Median value is displayed in solid line.

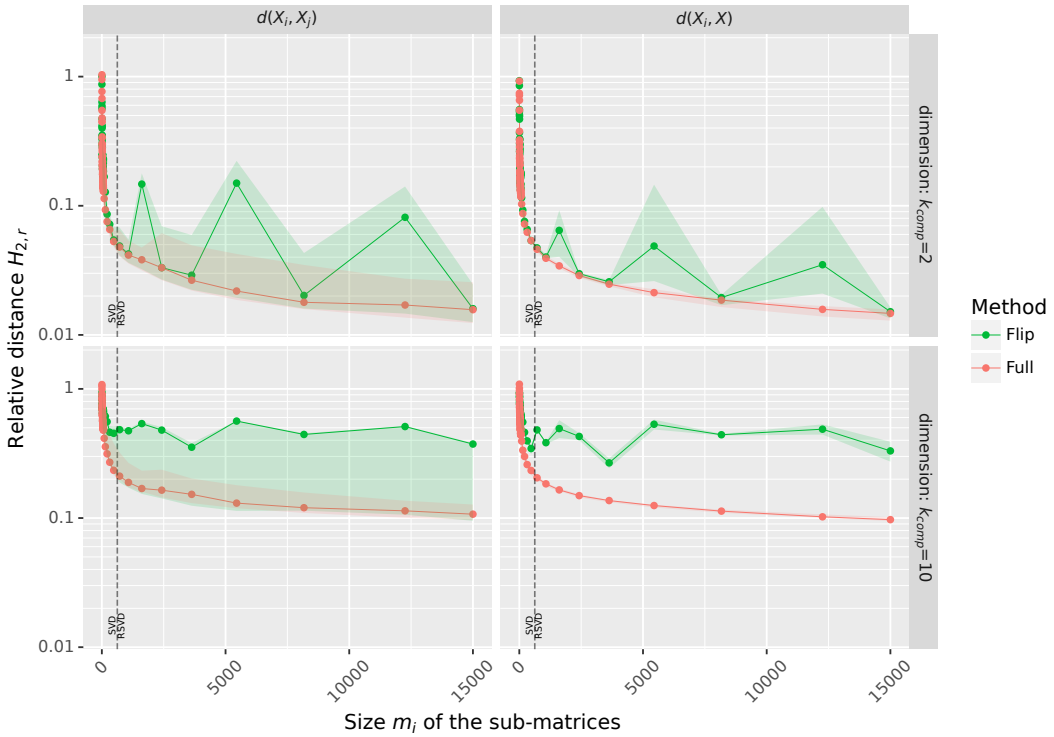


Figure 5.6 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original Lref sample (right) aligned using the Flip strategy. Median value is displayed in solid line.

smaller than 5,000 (150 for Atlas Guyane) we use a full SVD for the MDS, while for larger matrices we use RSVD. Following the results presented in Section 4.5 (p. 114), we only present results using the $H_{2,r}$ distance.

The results are shown in figures 5.3, 5.4, 5.5 and 5.6. We first comment on the results in 2 dimensions before moving on to the results in 10 dimensions. Figure 5.3 shows the result for the dataset Atlas Guyane. We can see that the distance we find is generally the same as the expected distance obtained using the **full** method, except in a few cases where the distance is higher for a particular size. This is most likely the result of the accumulate heuristic not finding the correct flip to align the point clouds. The results for the 10V-RbcL dataset in Figure 5.4, show that the computed distance is always higher than the expected distance, with more variation when looking at the distances between the extracted point clouds. The distance between these point clouds and the full one is still higher than the expected one, but it shows less variation. For both datasets Long Reads A and Lref, we can see that the distance varies between the expected distance and a large error on the distance. This makes us think of errors in the heuristics that find the correct orientation for the point clouds.

For all datasets, the behavior of the flip strategy in 10 dimensions is similar, with the distance obtained being relatively stable, although always significantly higher than the expected results. This may be due to a problem with the flip method, but we saw similar behavior for most of the resampling methods in Section 4.5 (p. 114) so this could also be due to a problem with high dimensional comparisons.

5.3 Folding method

Our idea was to look for a transformation of the point clouds that does not depend on the orientation of the point clouds, in order to be able to compute the Hausdorff distance on these transformed version of the point clouds while preserving their shapes as best as possible, i.e. we want to respect the distance between different points, this would eliminate the need for the costly alignment step. In other words, if A and B are two point clouds where $A = S(B)$ and S is a symmetric transformation, we look for a transformation T such that

$$A = S(B) \implies T(A) = T(B) \tag{5.2}$$

or to put it in another way, we want to find a transformation T on the point clouds that satisfies

$$H(A, S(B)) = H(T(A), T(B)) \tag{5.3}$$

without knowing S. Considering that we do not need to check for all possible symmetries, but only for reflections around the axes of the SVD, and that these symmetries are caused by sign changes in the singular vectors, we can use the absolute value of the coordinates of the point clouds.

Now let us give a more formal definition. First we define the absolute value $|.|$ operator on a vector $a = (a_1, a_2, \ldots, a_k)$ as $|a| = (|a_1|, |a_2|, \ldots, |a_k|)$. By extension, the absolute value $|A|$ of the point cloud A is the set of the absolute values of all the points that make up the cloud. Applying this transformation to a point cloud makes it look like the points are being "folded" along the axes, hence the name we decided to give to this method.

Figure 5.7 shows how the folding algorithm transforms point clouds into 2 dimensions.

On the left there are two point clouds obtained with MDS on two subsets of a single dataset, which we therefore expect to be similar. While by simply looking at them we can see that they

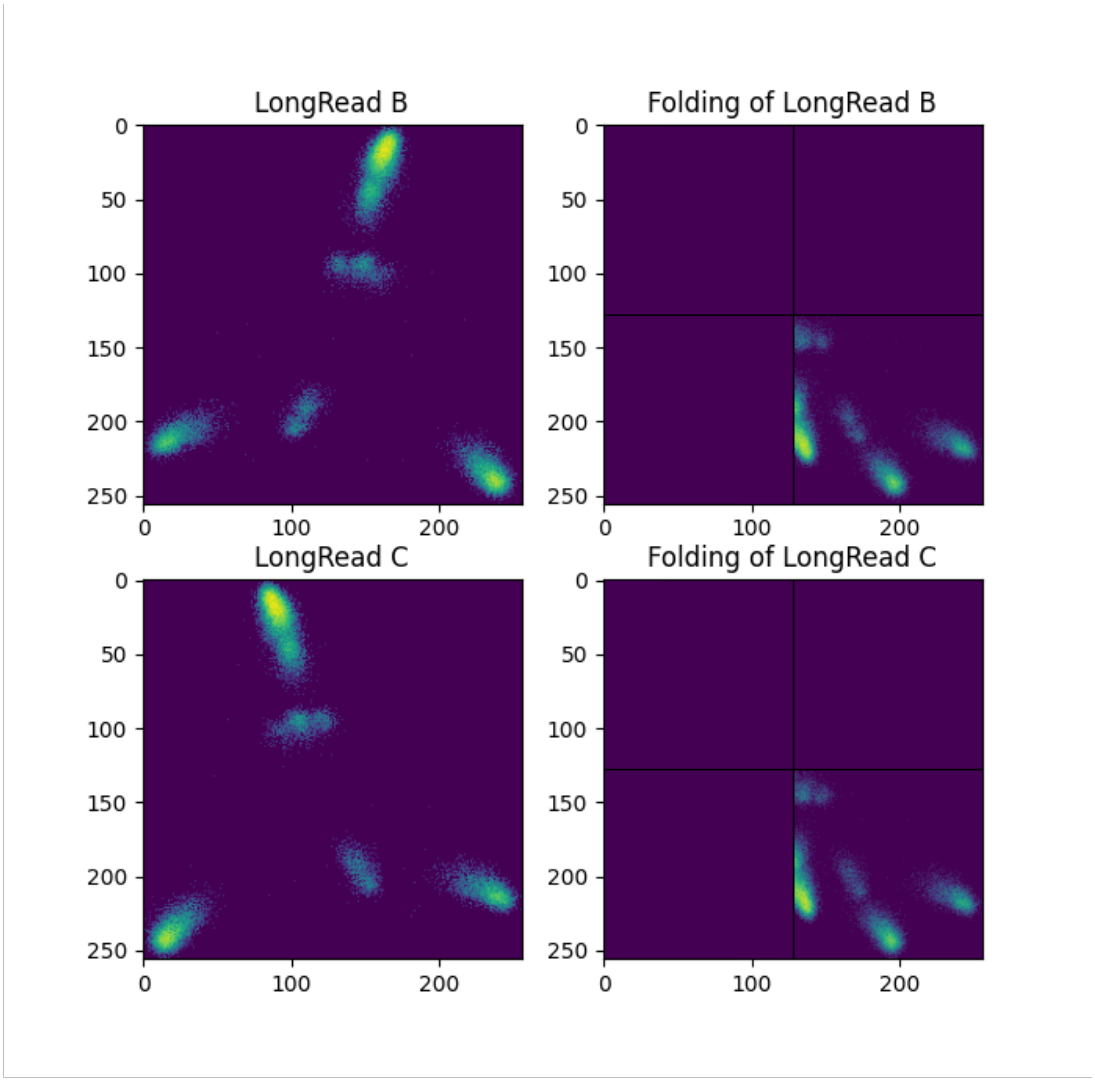


Figure 5.7 – Example of how the folding of 2 point clouds presenting symmetries can create similar outputs.

have the same shape, the fact that they have a symmetry around the vertical axis means that if we were to compute the Hausdorff distance between them we would find that they are far apart. On the left, we show what these point clouds look like after going through our folding method.

In Sections 5.3.1 and 5.3.2, we will express bounds on the folding algorithm to try to determine if it statisfies the constraints we gave in Equation 5.3. We can show that the distance after folding, can only be less than or equal to the real distance. This will allow us to understand how using this method will affect the distances we are working with. We also want to make sure that this step does not transform the data in such a way that the comparison becomes irrelevant, for example, if this transformation affects the shapes in such a way that it becomes impossible to distinguish between point clouds that are similar and those that are not.

5.3.1 Upper Bound

Lemma 3. *Let two clouds of points A and B and $|A|$ and $|B|$ be the folded clouds then*

$$H(|A|,|B|) \leq H(A,B)$$

Proof. Let d be the directed Hausdorff distance between A and B. By definition of the directed Hausdorff distance

$$h(A,B) = \max_{a\in A} \min_{b\in B} ||a-b|| = d \tag{5.4}$$

Which is equivalent to

$$\forall a \in A, \exists b \in B, ||a-b|| \leq d \tag{5.5}$$

We can apply the reverse triangle equality to this expression, giving us

$$\forall a \in A, \exists b \in B, |||a|-|b||| \leq ||a-b|| \leq d \tag{5.6}$$

And then we have

$$h(|A|,|B|) = \max_{a\in A} \min_{b\in B} |||a|-|b||| \leq d = h(A,B) \tag{5.7}$$

Applying the same argument to $h(|B|,|A|)$, we can conclude that the following holds $H(|A|,|B|) \leq H(A,B)$ □

We can also show this holds true for the MHD.

Lemma 4. *For two point clouds A and B, we have $MHD(|A|,|B|) \leq MHD(A,B)$*

Proof. Let us take the directed distance of the MHD between A and B to show that

$$h_{MHD}(|A|,|B|) = \frac{1}{N_A}\sum_{a\in A} d(|a|,|B|) \leq h_{MHD}(A,B) = \frac{1}{N_A}\sum_{a\in A} d(a,B) = d \tag{5.8}$$

Now let us consider a in A. We know that there exists a b' in B such that $||a-b'||$ is minimal. Then by applying the reverse triangular inequality to this expression, we get that $|||a|-|b'||| \leq ||a-b'||$ and by extensions, $\min_{b\in B} |||a|-|b||| \leq ||a-b'|| = \min_{b\in B} ||a-b||$. Since this is true for all $a \in A$, then by extension it is also true for the sum of all a and we have our proof.

□

5.3.2 Lower Bound

Unfortunately, there is no lower bound, since it is possible to create point clouds with an arbitrarily large Hausdorff distance that have a distance of 0 when folded onto one another. Consider an example in one dimension. First, we choose a point cloud consisting of n elements all positive, this will be our first point cloud A. Then we can create a second point cloud $B = A \cup \{-a, a \in A\}$ (actually, even a subset of A would suffice). Since we constructed A to only have only positive points, $|A| = A$. We also have $|B| = A$. Therefore, while these two point clouds are different, they have the same Hausdorff distance after folding. However, in this case, A is not centered, which is the case for the point clouds we expect from the MDS. We can work around this problem by finding the center of gravity of A (let's call it g) and adding n points in $-g$ to make A centered (assuming g itself is already a point of A, otherwise we have to add g to A as well).

However, the tests we perform in the next section show that this may not be a problem, as the cases described here require very specific point clouds that are most likely not representative of real data. For instance, in our examples, we need to place a large number of points at the exact same coordinates in order to maintain the centering we expect from point clouds generated with MDS.

5.3.3 Experiments on synthetic datasets

5.3.3.1 Synthetic point cloud generation

We expect the point clouds we are working with to have some clusters around which points tend to gravitate and want to take this into account when generating point clouds. In our tests, we start by randomly placing some clusters with a uniform distribution. Then we place points with a Gaussian distribution around them. By generating pairs of point clouds we can use the same clusters and different positions to generate "close" point clouds, while different clusters will generate "far" point clouds. We can then play with the parameters to try and simulate different possibilities. The possible parameters to consider are:

— d: the dimension of the space
— s: The number of clusters
— s2: The number of clusters of the second point cloud (if not specified is equal to s)

The goal of these experiments is to test out 5 different scenarios:

— Low-dimensional Close (d: 2, s: 3)
— Low-dimensional Far (d: 2, s: 3, s2: 4)
— High-dimensional Close (d: 10, s: 3)
— High-dimensional Far (d: 10, s: 3, s2: 4)
— Very High-dimensional Far (d: 50, s: 3, s2: 4)

Most importantly, this will help us to verify that our method can discriminate between point clouds that are close, and those that are not. We also want to study the effect of taking into account more dimensions.

5.3.3.2 Results

All of the tests are performed over 5 000 repetitions and distances are calculated using the MHD. We chose to use this method because it is a good way to deal with outliers that do not introduce additional parameters [40].

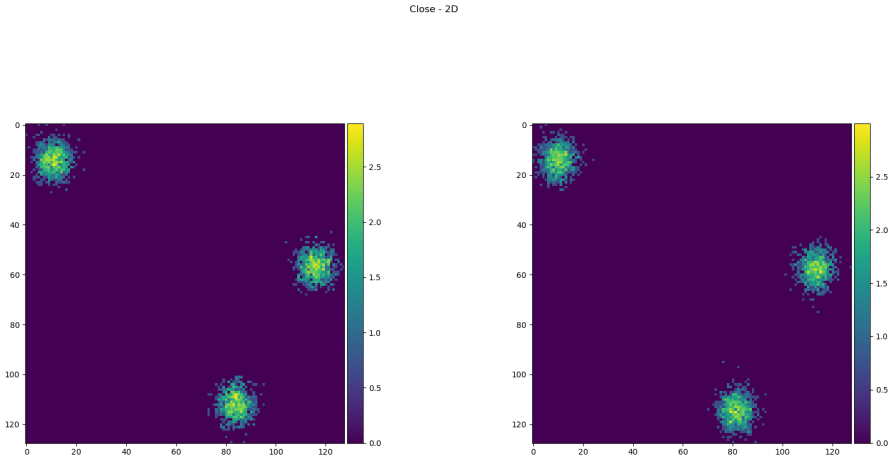


Figure 5.8 – Heatmap for the Close point clouds.

Low-dimensional Close. Figure 5.8 shows us an example of a pair of point clouds generated during our tests, where we can see very similar point clouds. Both show the 3 clusters very distinctly. Figure 5.9 shows us three histograms: on the top left is the true distance, which is quite low because the images are generated to be close. The upper right shows the distance after applying the folding scheme. As discussed in Section 5.3.1, these distances are always lower than (or equal to) the real ones. The bottom image shows the histogram of the ratio between the distance computed after folding and the expected Hausdorff distance. Since the distance after folding can only be smaller than the real one this ratio is always between 0 and 1, where 1 is better, since it means that the distance has been preserved by the transformation. In Figure 5.9 we can see that the ratio is mostly around 1, but there are some cases where this ratio drops to almost 0.5.

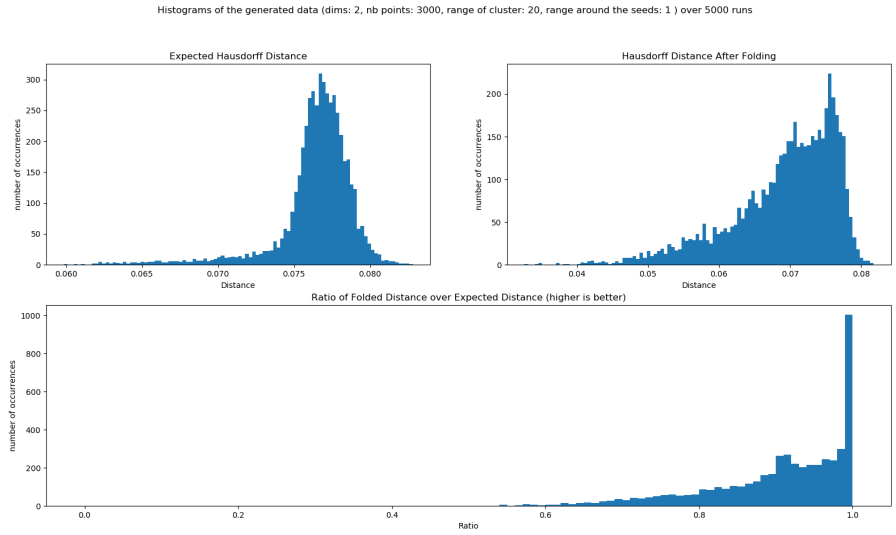


Figure 5.9 – Histogram of the distances for similar point clouds with 2 Dimensions.

High-dimensional Close Here, Figure 5.10 shows us that by considering more dimensions, the ratio between the real distance and the distance after folding stays much closer to 1 than

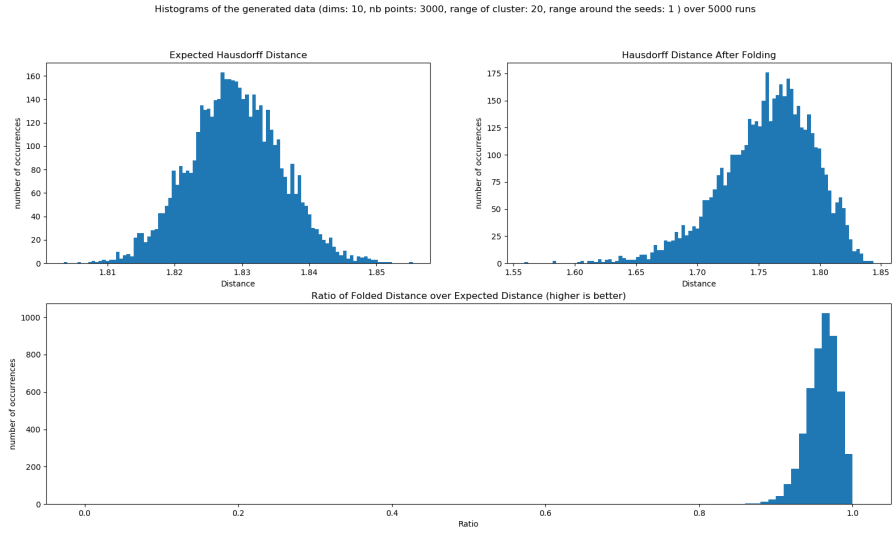


Figure 5.10 – Histogram of the distances for similar point clouds with 10 Dimensions.

before. In Figure 5.9 the ratios were between 0.5 and 1, while here they are mostly above 0.9.

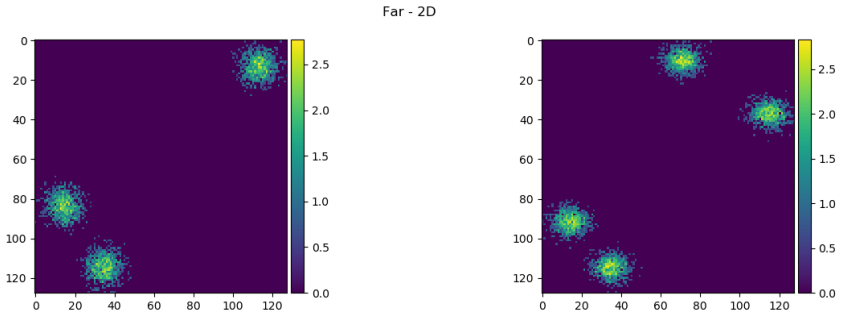


Figure 5.11 – Heatmap for the Far point clouds.

Low-dimensional Far. Figure 5.11 shows point clouds that are different because we added an extra cluster to the second one. Looking at the bottom graph in Figure 5.11 we can see that the ratio is not consistent at all, which means that the folding algorithm cannot be used in this case because we cannot guarantee that it is representative of the true distance.

High-dimensional Far. Figure 5.13 shows more concentrated results than those in Figure 5.12 however it is still quite spread and could lead to mistakes.

Very High-dimensional Far. Figure 5.14 shows us that when we select an even larger number of dimensions, the ratio starts to stabilize more, but does not reach 1, which is expected, since the folding makes the point clouds closer than they really are.

5.3.4 Numerical Study

To evaluate the folding strategy, we perform the same experiments as in Section 4.5 (p. 114), using the same four datasets: Atlas Guyane, 10V-RbcL and Long Reads A from Section 1.8 (p.

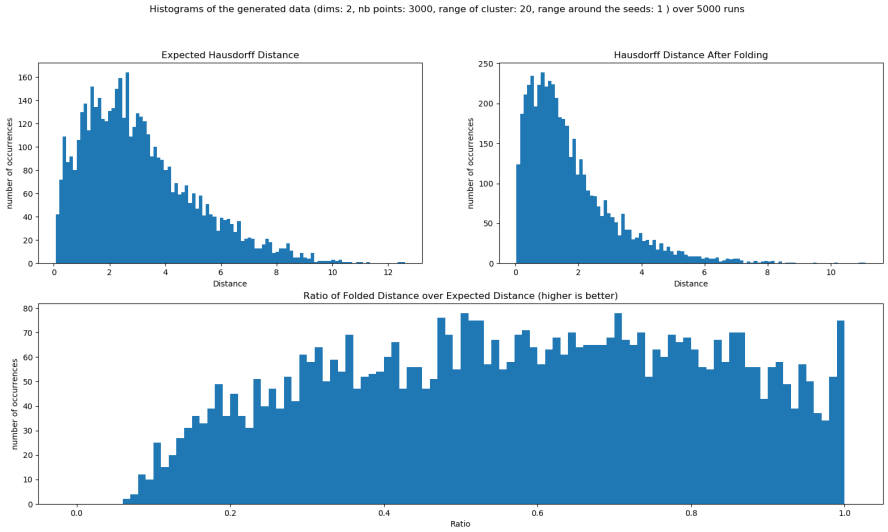


Figure 5.12 – Histogram of the distances for far point clouds with 2 Dimensions.

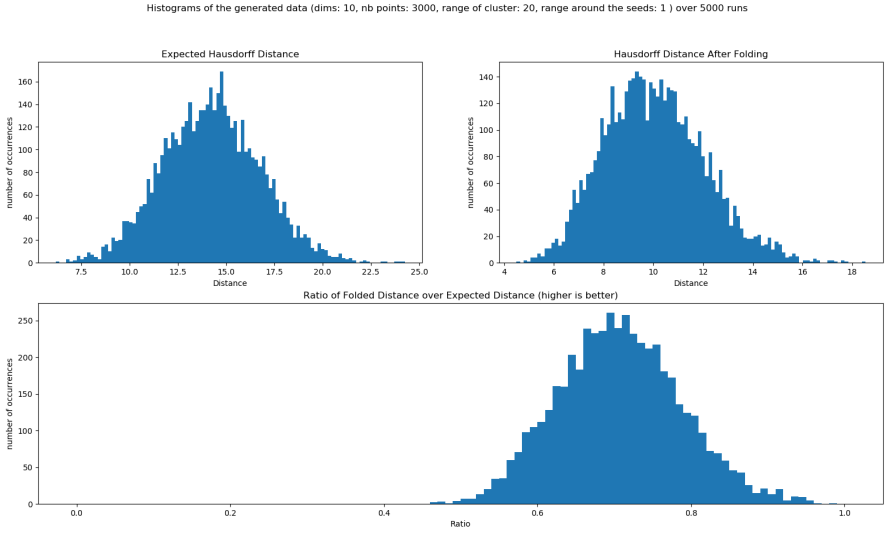


Figure 5.13 – Histogram of the distances for far point clouds with 10 Dimensions.

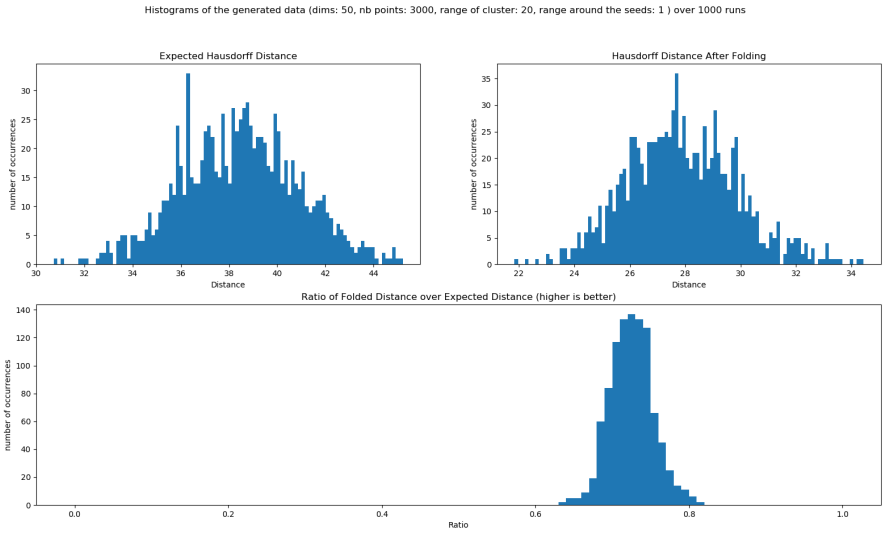


Figure 5.14 – Histogram of the distances for far point clouds with 50 Dimensions.

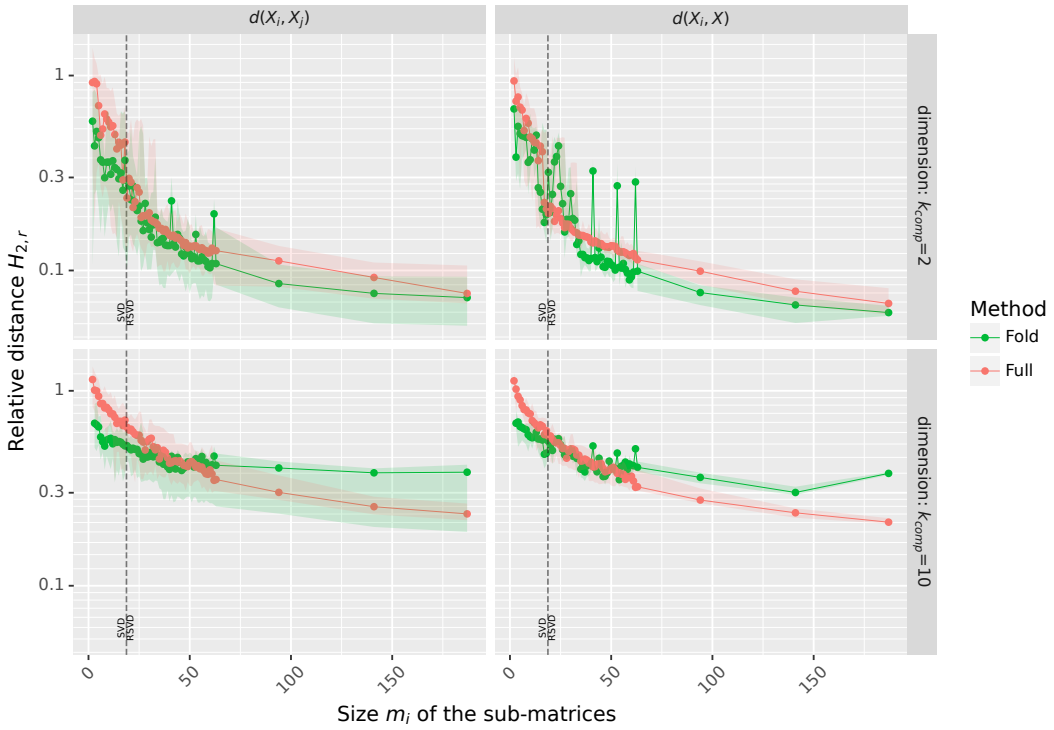


Figure 5.15 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original Atlas Guyane sample (right) aligned using the Fold strategy. Median value is displayed in solid line.

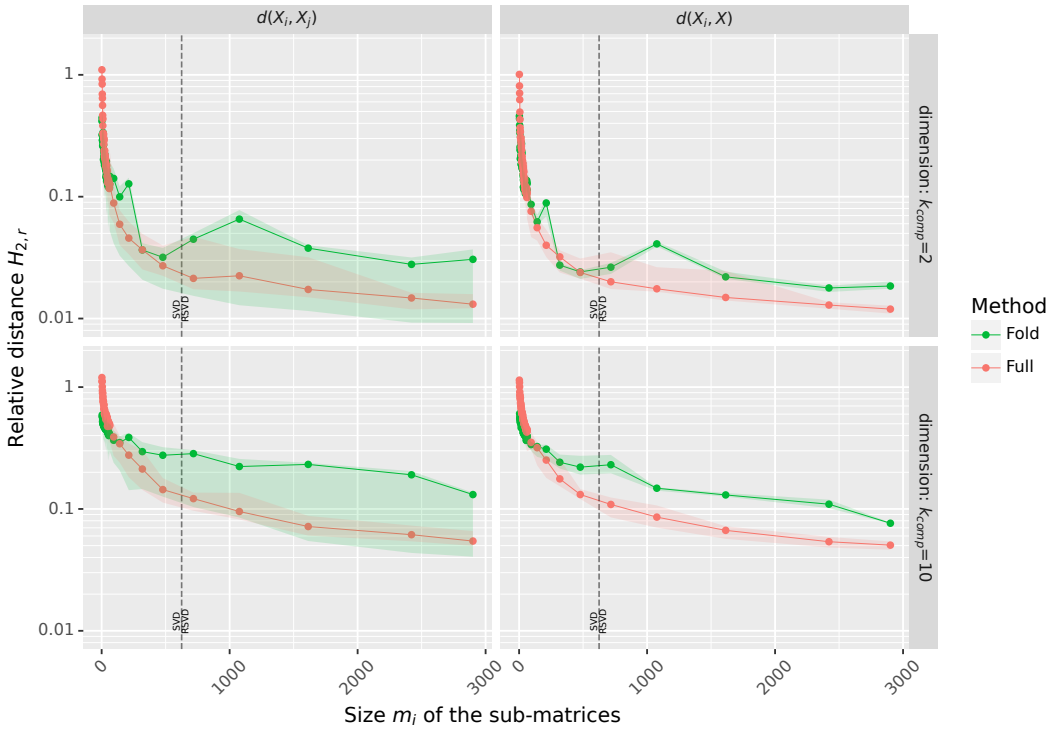


Figure 5.16 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original 10V-RbcL sample (right) aligned using the Fold strategy. Median value is displayed in solid line.

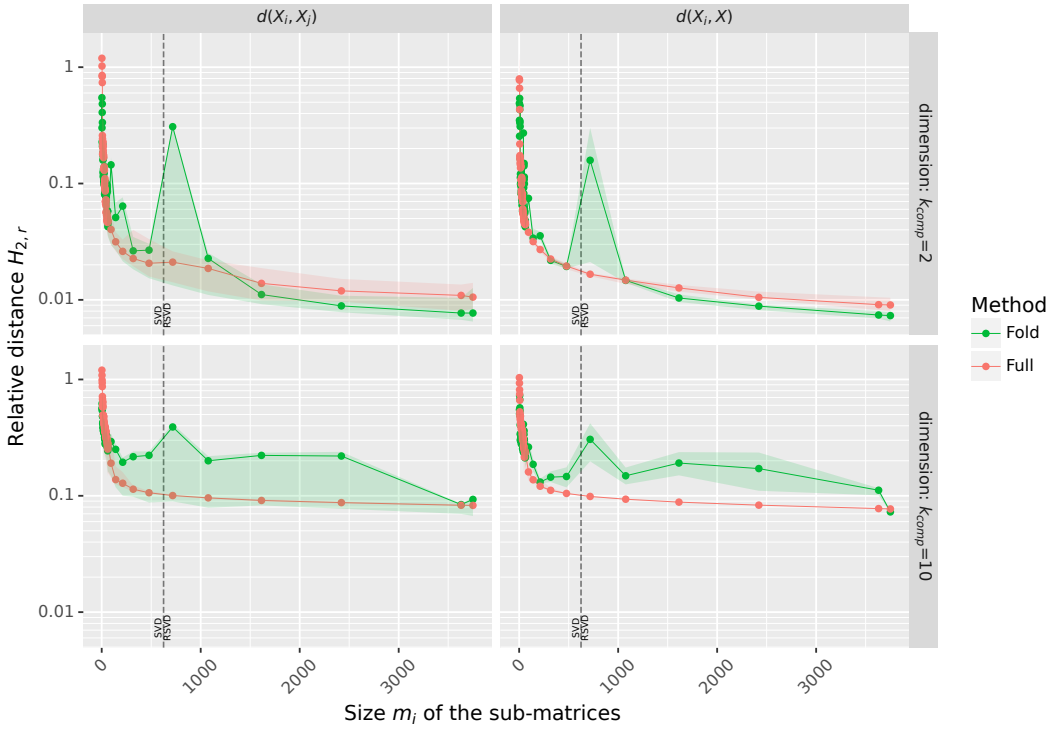


Figure 5.17 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original Long Reads A sample (right) aligned using the Fold strategy. Median value is displayed in solid line.

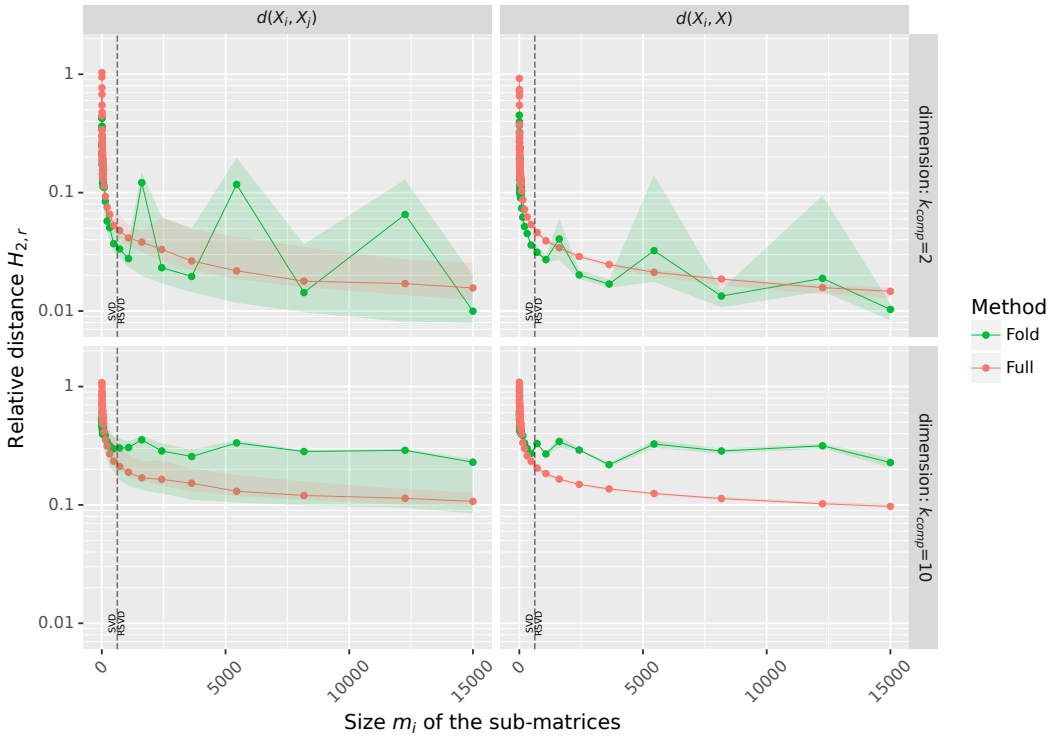


Figure 5.18 – Evaluation of the $H_{2,r}$ distance of 8 submatrices between each other (left) and between the submatrices and the original Lref sample (right) aligned using the Fold strategy. Median value is displayed in solid line.

19) and Lref (see Section 4.5,p. 114) instead of **S5**. At iteration l, we extract two different submatrices that we align by folding and construct a point cloud X_l for this iteration. We then use resampling by dividing the point cloud into 8 and estimate the quality of the approximation of the true point cloud X by X_l using the distances between the subdivided point clouds. Since we use the same experimental setup as in Sections 5.2.3.2 and 4.5, we will not describe it further here.

The results are shown in Figures 5.15, 5.16, 5.17 and 5.18. These results are very similar to those of the flip strategy presented in Section 5.2.3.2. This is due to two main factors: first, the submatrices considered are the same, since in the context of both experiments, we generated the submatrices once and the applied flip and fold successively to generate the X_l. The second reason is the fact that both methods work under the same constraint that they must be applied to point clouds that are misaligned only up to a reflection around the principal axes. The main difference is that the folding scheme sacrifices the actual shape of the point cloud for a faster result, while the flip strategy can achieve the exact transformation at the cost of more computation time.

When using heuristics for the flip such as the one we used in the experiment of Section 4.5, we may also find ourselves in a situation where we have a false negative, *i.e.* we do not find the right transformation and the resulting distance appears higher than it should. This problem does not occur with folding, but it can produce false positives by bringing together clouds that are far apart.

The tendency of folding to bring point clouds closer together than they actually are can be seen in Figures 5.15 and 5.18, where the folded distance is sometimes smaller than the reference. This does not happen for any other method of aligning point clouds. It is also remarkable that the relative distance after folding remains in a comparable order of magnitude to the actual

distance, even though we have no reason to believe that it would behave this way, sine folding is supposed to reduce distances.

5.4 Discussion on Performance

In this section, we compare the performance result of the Flip and Fold algorithms with the different methods presented in Chapter 4. Figures 5.19, 5.20, 5.21 and 5.22 show, for the Atlas Guyane, 10V-RbcL, Long Reads A and Lref datasets, respectively, the computation time for each method (Combined, Landmark Procrustes and BDLPMDS) with respect to the size of the extracted submatrices, presented in log-y scale with both non-log-x scale (top) and log-x scale (bottom). We present the flip and fold methods in a single curve, since they are computed in the same way with MDS on two disjoint submatrices. In practice the folding operation will be faster than using flips (even using heuristics). We do not report this here, since the dominant part of both methods remains the MDS.

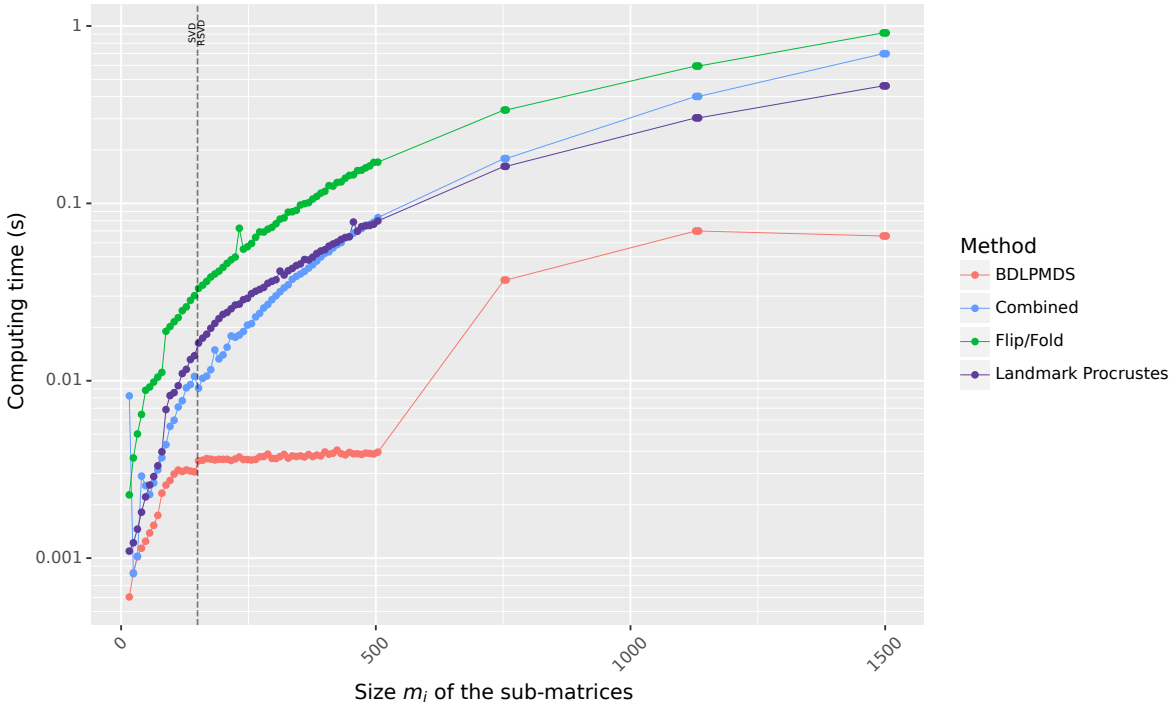
Except in Figure 5.19, where the computation time is so fast that the result is dominated by the various alignment algorithms instead of the MDS, we can see that the flip and fold schemes require about the same computing time as the Landmark Procrustes approach. This is to be expected, since both the Landmark Procrustes and the flip and fold methods are based on computing two MDS on matrices half the size of X_l at each iteration. BDLPMDS remains the fastest approach due to its progressive nature.

5.5 Application

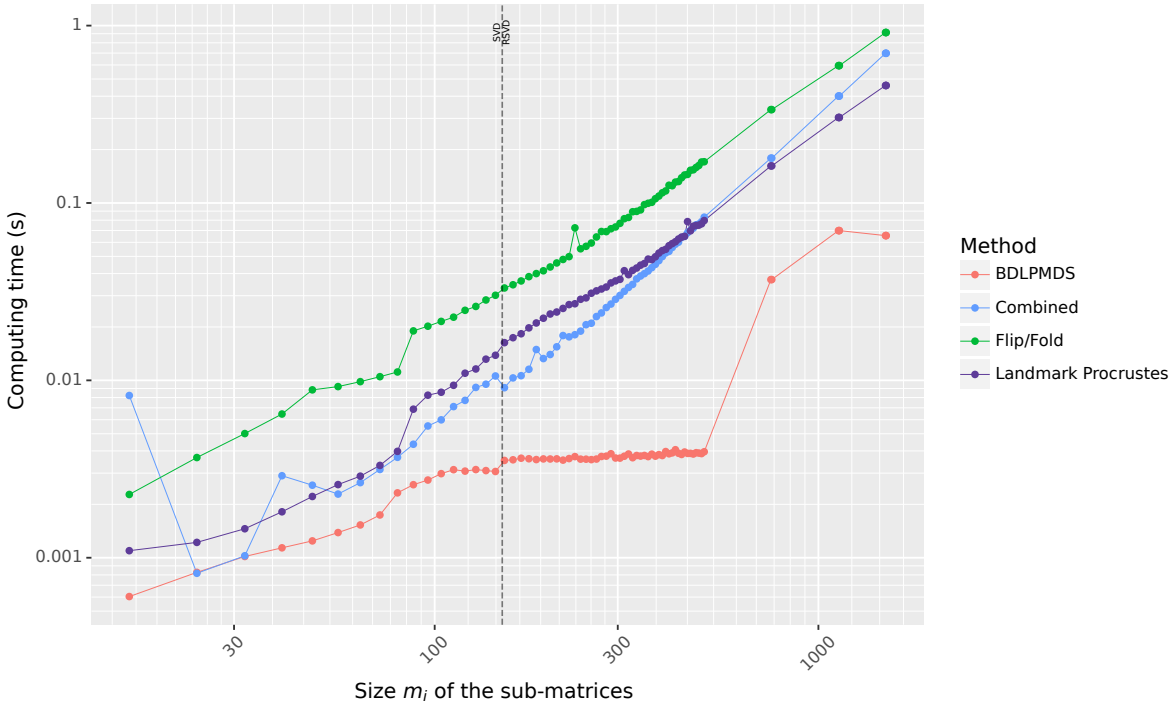
In this section, we present the Flip and Fold heuristic in two cases. First, in Section 5.5.1 we return to the **S5** sample and compute the distance between the results we obtained by RsEVD-MDS and RSVD-MDS. We compare the alignment using flip and fold with the more rigorous result from Section 3.8.1 and confirm that the result we find using either a flip or fold strategy is consistent with what we obtained using Procrustean transformations.

In Section 5.6, we use the flip and fold heuristics to compare the point clouds from the Long Reads sample presented in Section 1.8.2 (p. 21). Since the Long Reads datasets have been sampled in an independent and identically distributed manner, we expect them to have the same principal axes, and therefore the first three issues of Chapter 3 (axis permutations, rotation and translation) are therefore be solved by the sEVD step, and that all we have to do to align them is to consider the reflections around the principal axes. After validating that this approach gives correct results in terms of the distance between the point clouds, we reconstruct the full ABCD dataset (similar to how we rebuilt the full **S5** dataset in Section 3.8.4, p. 92) using only the Flip strategy. The main difference in this case is that the Flip strategy does not require the use of any off-diagonal-block elements, unlike the BDLPMDS which requires the use of landmarks to align all point clouds.

Finally, in Section 5.5.3 we show an attempt to use the flip and fold method to compare the different point clouds that make up the **S5** sample, even though each L_i is sufficiently different because they were sampled at different times of the year. As such, the L_i samples are not independent and identically distributed and we show how applying flip or fold in a case where issues 1 through 3 of Chapter 3 are present can lead to catastrophic failure.

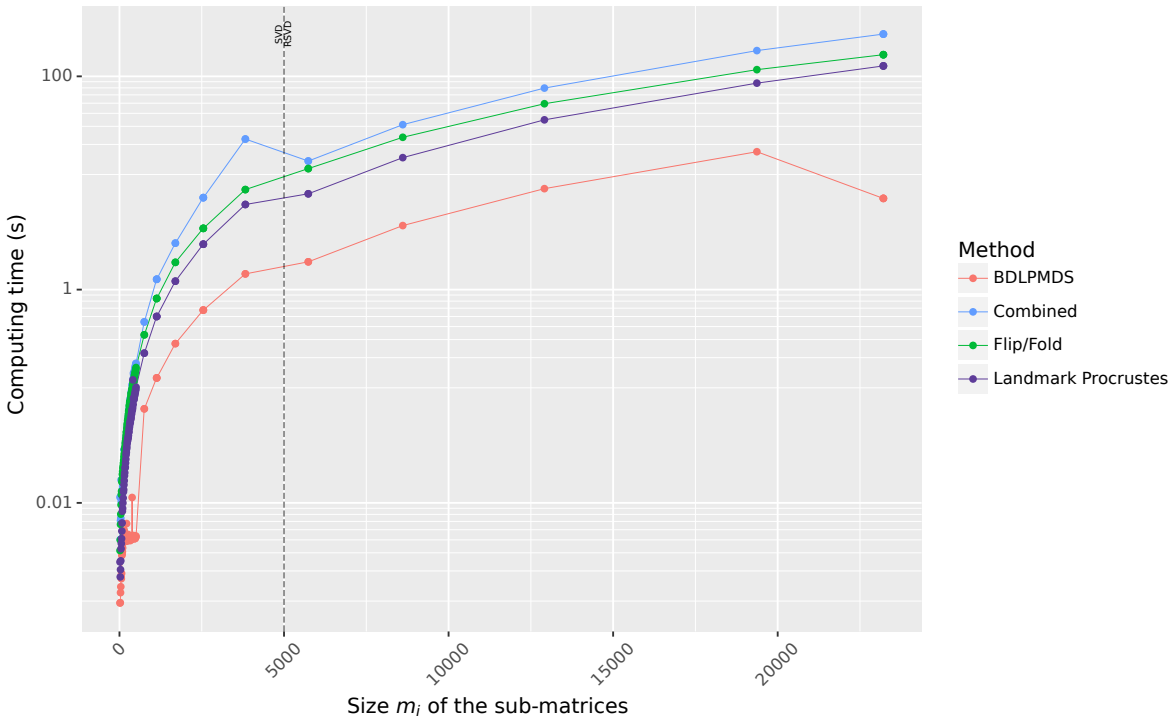


(a) Nonlog x-scale.

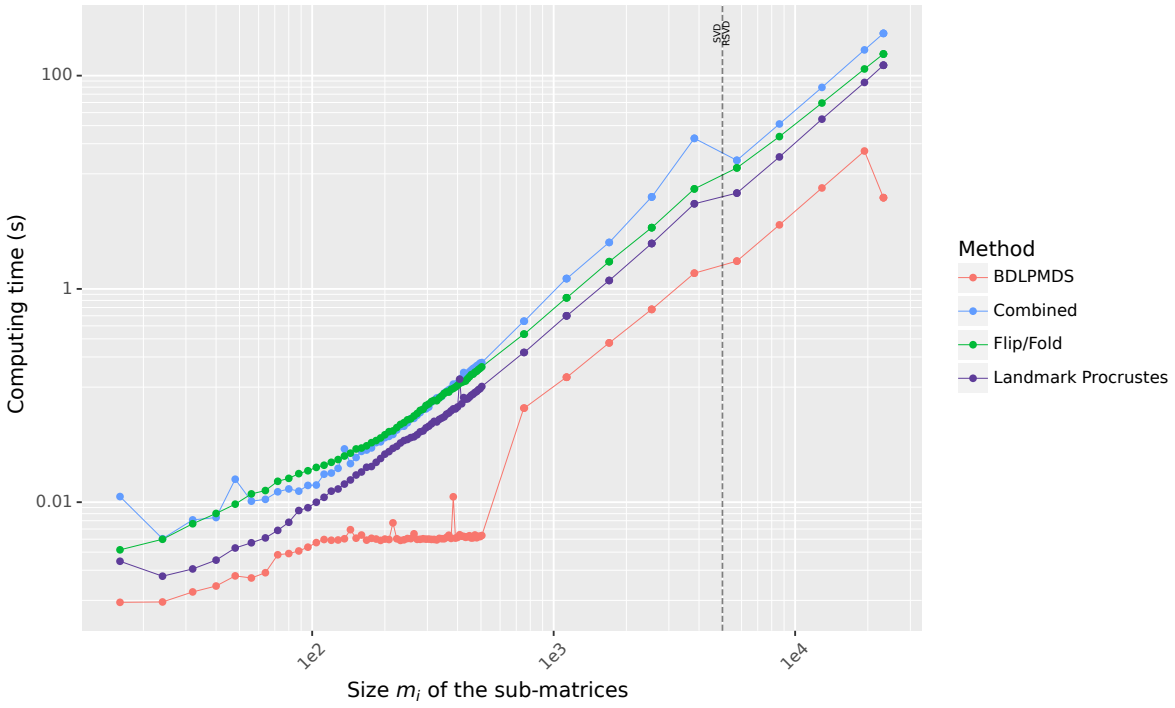


(b) Log x-scale.

Figure 5.19 – Computation time for Flip and Fold algorithms compared with the other heuristics from Chapter 4 for dataset Atlas Guyane.

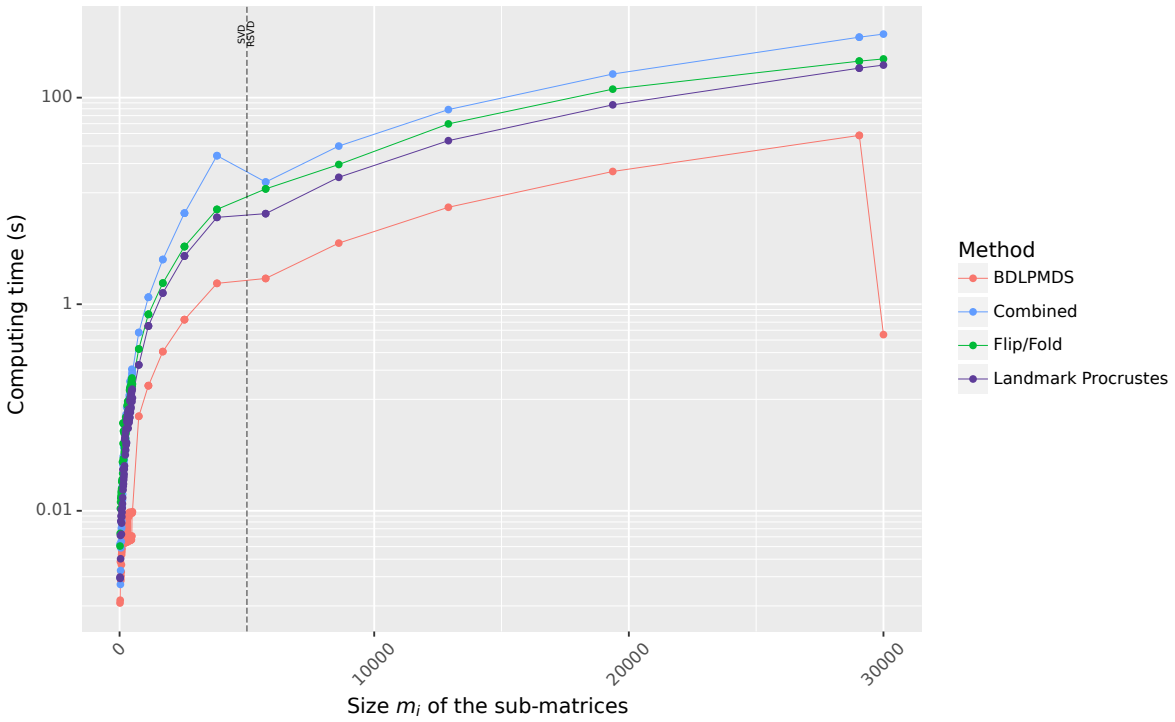


(a) Nonlog x-scale.

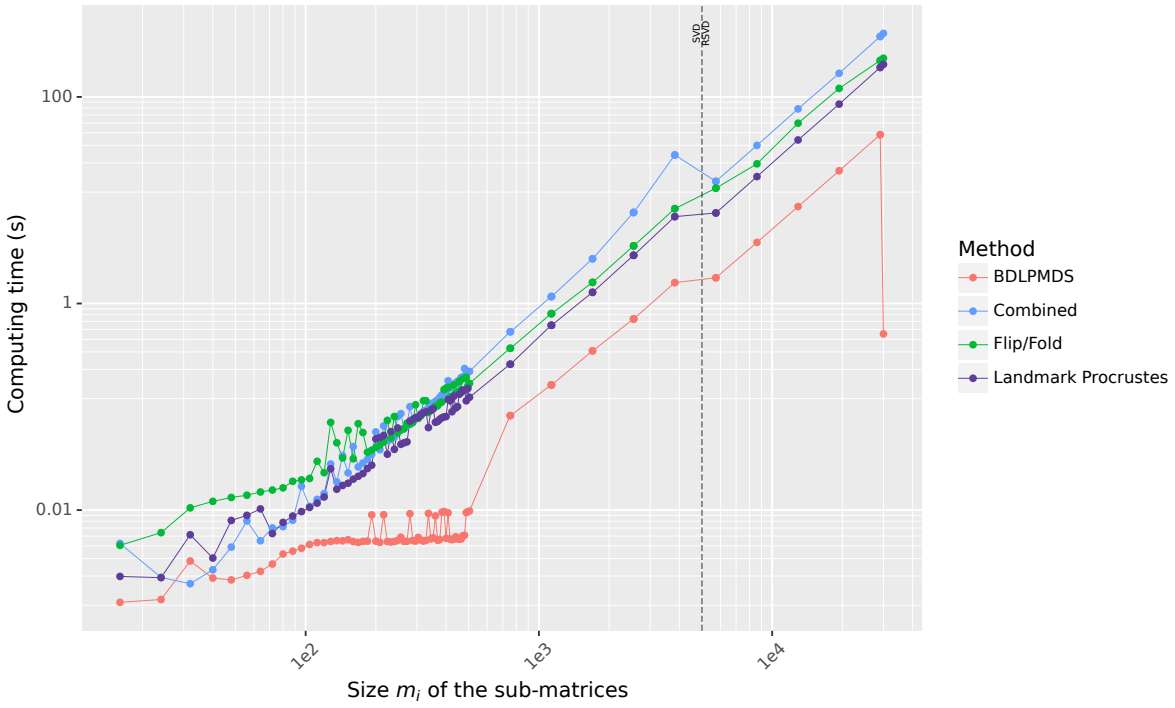


(b) Log x-scale.

Figure 5.20 – Computation time for Flip and Fold algorithms compared with the other heuristics from Chapter 4 for dataset 10V-RbcL.

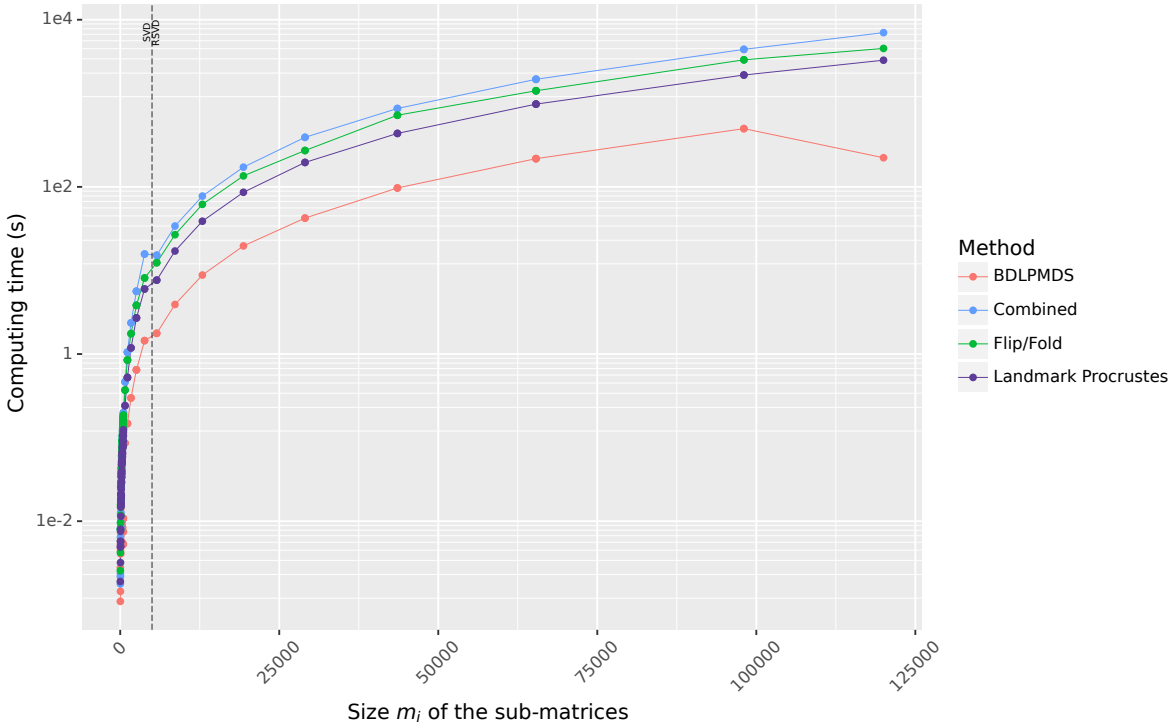


(a) Nonlog x-scale.

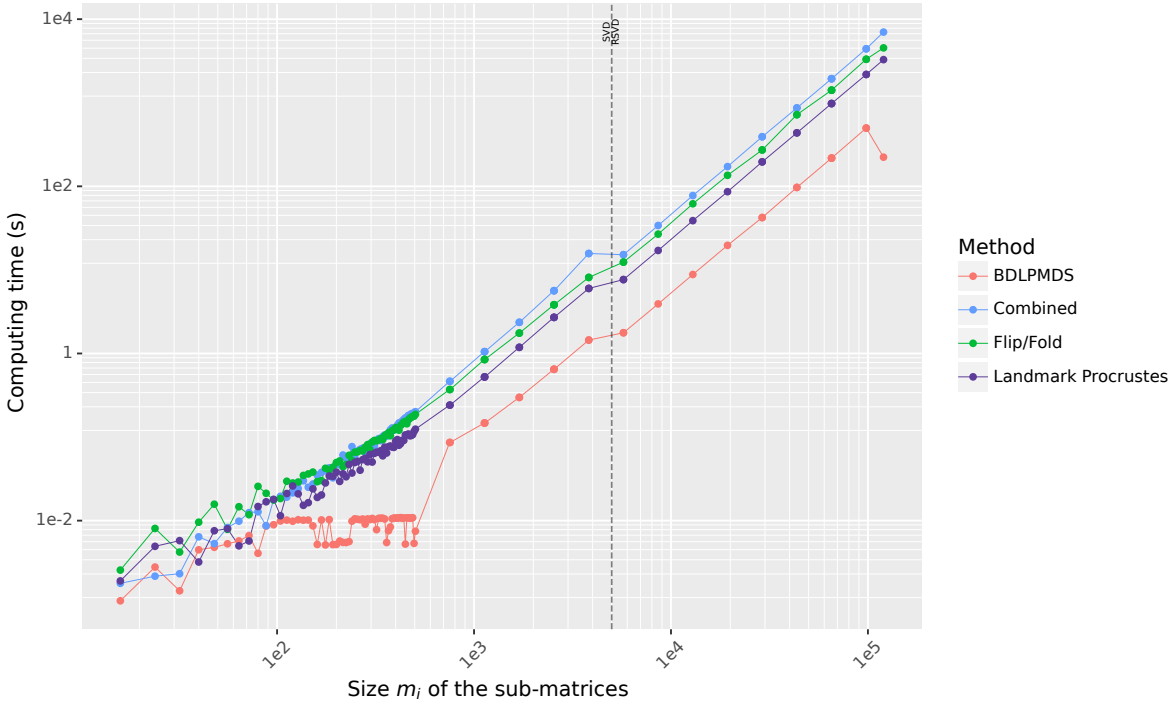


(b) Log x-scale.

Figure 5.21 – Computation time for Flip and Fold algorithms compared with the other heuristics from Chapter 4 for dataset Long Reads A.



(a) Nonlog x-scale.



(b) Log x-scale.

Figure 5.22 – Computation time for Flip and Fold algorithms compared with the other heuristics from Chapter 4 for dataset Lref.

5.5.1 Assessing the closeness of S5 (L1-L10) point clouds generated by RSVD and RsEVD

In this section, we will revisit the comparison between the RsEVD-MDS (Figure 2.19, p. 60) and the RSVD-MDS (Figure 1.13, p. 26). We want to compare the Flip and Fold schemes with the results we obtained in Section 3.8.1 using Procrustean transformations. Since both these results were obtained using the same input matrix, with only a few variations due to the random nature of the algorithms and the number of randomized linear embeddings, we can assume that we are in the conditions where the results of that chapter apply. In this case, issues **1 to 3** of Section 3.3 (axis permutations, rotation and translation) are solved and we should be able to align the point clouds using only the reflections around the axes. The resulting distance is shown in Table 5.1. Since the Flip and Fold heuristics do not benefit from considering more dimensions than are included in the final comparison, we only considered the same number of dimensions for the Procrustes alignment as well. In the leftmost column of this table, we can see the distance obtained after aligning the point clouds using the Procrustes transformations which will be our baseline. We can see that in most cases, the distance obtained using the flip or fold method is the same as the Procrustes result, except when using the flips in 4 dimensions. This illustrates one of the possible problem we can encounter with this method, which is a false negative. In this case it is due to the fact that the accumulate heuristic found the wrong orientation for the 4^{th} dimension. We can correct this error using an exhaustive approach at the cost of significant computation time. The result we then obtain is **3.3607725847857384e-06**, which confirms the false negative hypothesis.

Dimension	Procrustes	Flip	Fold
2	1.8248895282299271e-06	1.8249147286464697e-06	1.824906847434291e-06
3	2.2191719647848152e-06	2.513734093895583e-06	2.513734093895583e-06
4	3.1035338219322256e-06	6.870188605886184e-02	3.3607698553003758e-06

Table 5.1 – $H_{2,r}$ distance between point cloud of the $S5$ sample obtained through RsEVD-MDS and RSVD-MDS aligned using either Procrustes, flip (with the accumulate heuristic) or fold.

5.5.2 Confirming the closeness of the four Long Reads samples

In this section we study the four Long Reads samples from Section 1.8.2 (p. 21). These four samples are extracted from a larger matrix in an assumed independent and identically distributed manner. The samples are named from A to D and the complete dataset is named ABCD. Thus, we expect that after running the MDS on each of these samples, only issue 4 of Chapter 3 (reflections around the principal axes) will remain. Issue 4 is due to the non-uniqueness of the sEVD (and SVD) and will therefore always be present.

In Table 5.2, we compare the distances between the individual Long Reads samples using the relative H_2 distance ($H_{2,r}$). We performed each comparison using the Procrustes transformation to align each sample to the complete ABCD sample in the same way that we aligned all L_i on the full sample in Section 3.8.3. This gives us a reference distance against which to evaluate our other heuristics. We then compared them using either Flip (to Long Reads A) or Fold. We can see that all the results are very close to the theoretical result, confirming that the Flip and Fold strategies are effective for comparing these datasets. The visualization of the Flip and Fold schemes on point clouds is shown in Figure 5.23, where the leftmost column is the original point clouds as obtained by MDS, the middle column shows the point clouds after aligning them to Long Reads A using Flips, and the rightmost column is the result of the Folding heuristic. We can already see from the original point clouds obtained from MDS that the different point clouds are indeed very similar, and differing only up to a reflection around the principal

axes. Once we use either heuristic, we see that the point clouds are aligned. We should also point out that folding is destructive to the original shape of the point cloud, but is still a good way of aligning point clouds when only closeness matters.

	A			B			C			D		
	Proc.	Flip	Fold	Proc.	Flip	Fold	Proc.	Flip	Fold	Proc.	Flip	Fold
A	0	0	0	4.9e-3	4.9e-3	3.3e-3	5.0e-3	5.2e-3	3.7e-3	4.3e-3	4.4e-3	3.5e-3
B	4.9e-3	4.9e-3	3.3e-3	0	0	0	4.2e-3	4.3e-3	3.2e-3	4.3e-3	4.5e-3	3.3e-3
C	5.0e-3	5.2e-3	3.7e-3	4.2e-3	4.3e-3	3.2e-3	0	0	0	4.4e-3	4.6e-3	3.5e-3
D	4.3e-3	4.4e-3	3.5e-3	4.3e-3	4.5e-3	3.3e-3	4.4e-3	4.6e-3	3.5e-3	0	0	0

Table 5.2 – $H_{r,2}$ distance between each pair of Long Reads datasets with alignment being done either with Procrustes transformation, Flip (towards Long Reads A) or Fold.

Once we have validated that both the Flip and the Fold approaches provide satisfactory alignment of the Long Reads sample, we want to, similarly to what is presented in Section 3.8.4 (p. 92), rebuild the entire ABCD point cloud using only the Flip strategy. The resulting point clouds are shown in Figure 5.24. In this figure, the top row represents the full ABCD point cloud as obtained by a RSVD-MDS on the full $120,000 \times 120,000$ matrix. Each of the following row represents one point cloud between A, B, C, D aligned on the full ABCD. The bottom row represents the reconstucted ABCD obtained by combining these point clouds. A summary of the distances between these point clouds and the original one is given in Table 5.3. As we can see, the point cloud rebuilt with Flip is very close to the original one, which confirms our intuition that under certain conditions, we can rebuild the entire point cloud without having to access any of the off-diagonal blocks.

Dataset	Procrustes	Flip	Fold
A	3.317e-03	3.434e-03	2.575e-03
B	3.761e-03	3.787e-03	2.682e-03
C	3.942e-03	4.190e-03	3.150e-03
D	3.663e-03	3.892e-03	3.051e-03
Rebuilt	5.693e-04	1.183e-03	2.575e-03

Table 5.3 – $H_{2,r}(., ABCD)$ distance between the full ABCD sample and each Long Reads sample aligned to ABCD, the bottom row is the distance between the rebuilt ABCD and the computed ABCD.

5.5.3 Limits of the Approach

Finally, we present here the limitations of both the Flip and Fold heuristics, and illustrate what happens when we try to use this approach to compare point clouds that do not come from independent submatrices extracted from a dataset. To illustrate this, we use the $L_1, \ldots, L_{10}$ datasets from Section 1.8.4 (p. 21). As explained in that section, the L_i datasets are datasets extracted from the same place at different times of the year. As such, the evolution of the biodiversity throughout the year will affect the structure of these matrices in relation to each other. This means that the principal axes of these datasets are not necessarily confounded and that more treatment is needed to align them than simply considering reflections around the principal axes. This behavior is described in more details in Section 3.8.4 (p. 92).

In Figure 5.25, we present the result of the attempt to apply Flips and Folds to the L_i samples. Each row contains one of the L_i, and each column represents a transformation applied to the point clouds. The leftmost column represents the point clouds as obtained from the output of MDS, the second one corresponds on the point clouds transformed by the Procrustean to align

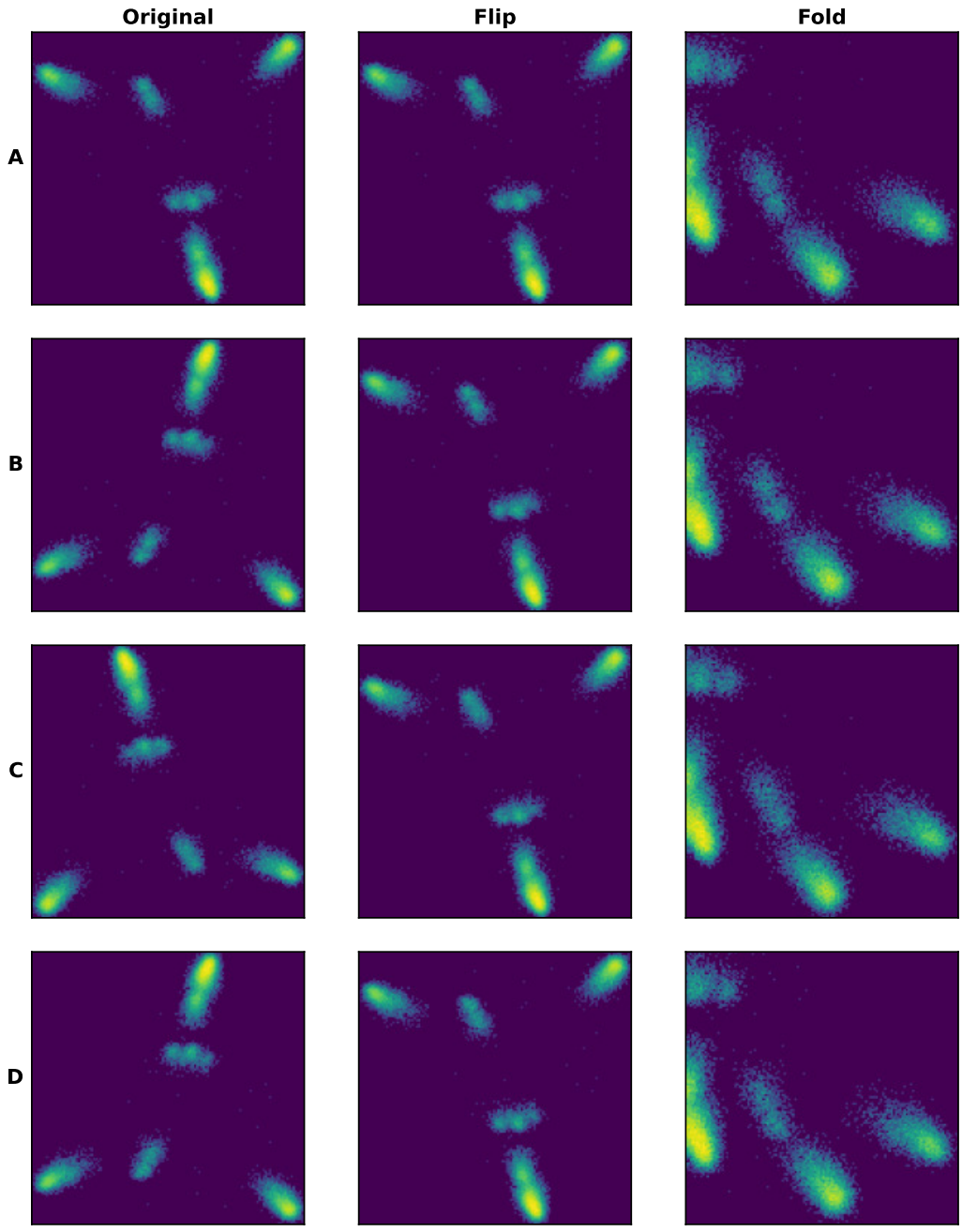


Figure 5.23 – Long Reads point clouds obtained from MDS (left) aligned to A using flips (middle) and folded (right).

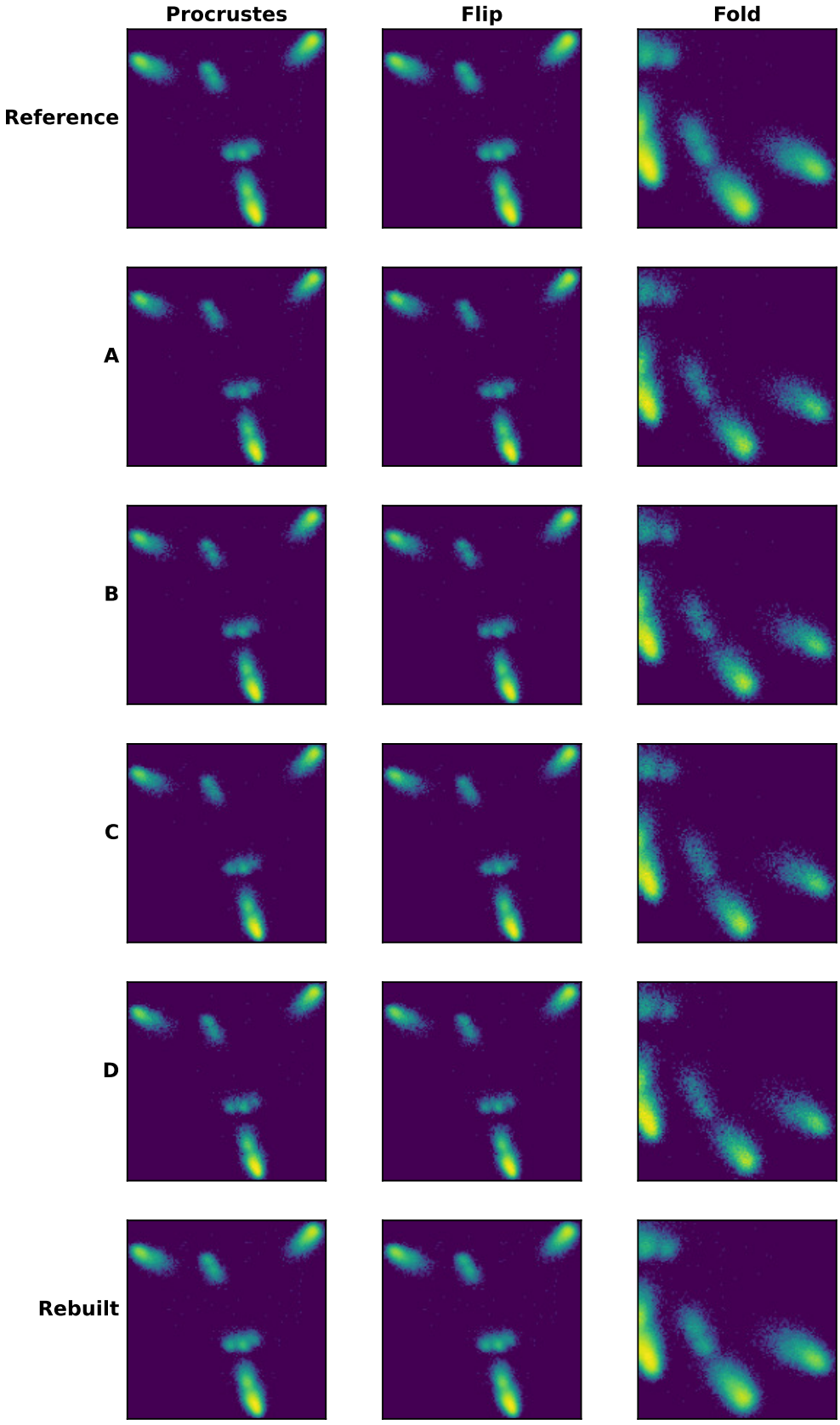


Figure 5.24 – Long Reads ABCD (top row) and LongReads aligned to it using Procrustes analysis (left), flips (middle) and folding (right). Bottom row shows the rebuilt ABCD sample obtained by merging each version of alignment.

them all on the point cloud obtained from RSVD-MDS. This is the same transformation that was applied to the first column in Figure 3.30. The third column represents the point clouds obtained using flips. As we assume that this method is used in the cases we don't have access to off-diagonal blocks, we decided to align every point cloud to $L1$ instead of **S5**. The last two columns represent the results obtained with the fold scheme. The fourth column represents the point clouds as they are directly folded so the folded version of the first column, while the last column represents the folded version of the point clouds aligned on **S5** using the Procrustean transformations of the second column.

In other words, if the method was correct, the third column should look like the second, and the fifth should look like the fourth. As we can see it is not the case, which is the expected result since the different L_i samples are too different and the issues 1 to 3 of Chapter 3 are not solved by the sEVD, which means that a flip or fold strategy cannot correctly align them. The results of the relative H_2 distance between $L1$ and the rest of these point clouds for each alignment method are presented in Table 5.4. Results for all the other samples are presented in Appendix D.

Sample	Procrustes	Flip	Fold	Procrustes Fold
L2	9.287e-03	5.653e-02	1.627e-02	4.529e-03
L3	6.931e-03	2.571e-02	1.914e-02	4.502e-03
L4	1.102e-02	9.280e-02	7.840e-02	6.226e-03
L5	1.453e-02	5.788e-02	5.333e-02	8.677e-03
L6	1.614e-02	6.265e-02	5.210e-02	9.408e-03
L7	1.720e-02	5.556e-02	4.610e-02	1.007e-02
L8	1.467e-02	1.091e-01	2.964e-02	1.026e-02
L9	1.918e-02	6.930e-02	5.454e-02	1.240e-02
L10	1.957e-02	3.412e-02	2.435e-02	1.248e-02

Table 5.4 – $H_{2,r}(L_1, L_j)_{j\in[\![2;10]\!]}$ for the different methods of aligning.

5.6 Conclusion

In this chapter, we explored the possibility of aligning the point clouds without the need for landmarks. We started from the observation that for point clouds extracted in an independent and identically distributed manner, all the problems of alignment are directly corrected by the sEVD, which correctly captures the eigenvectors, and out of the four issues presented in Chapter 3 only **issue 4** (reflections around principal axes) remains. This problem is due to the non-unicity of the sEVD, and as such, it is not possible to prevent it from occurring. However, if it the only cause of non-alignment, we can adapt our alignment methods so that landmarks are not required.

We have proposed two strategies for aligning such point clouds. The first, which we call flip, is to try all possible reflections and find the best one based on the distance presented in Chapter 3. Since this method can become computationally expensive when the number of dimensions considered increases, we have explored heuristics that allow a faster computation at the cost of some possible false negatives: cases where we do not identify the best transformation and where we assume that the distance between the point clouds is greater than it actually is. The second strategy we propose is called folding. It consists of removing the sign from the coordinates of all the points in the cloud. While this method is destructive in the sense that it irreversibly transforms the point cloud, it allows to map all possible reflections around all axes into the same point cloud. Thus, point clouds that are identical except for reflections around

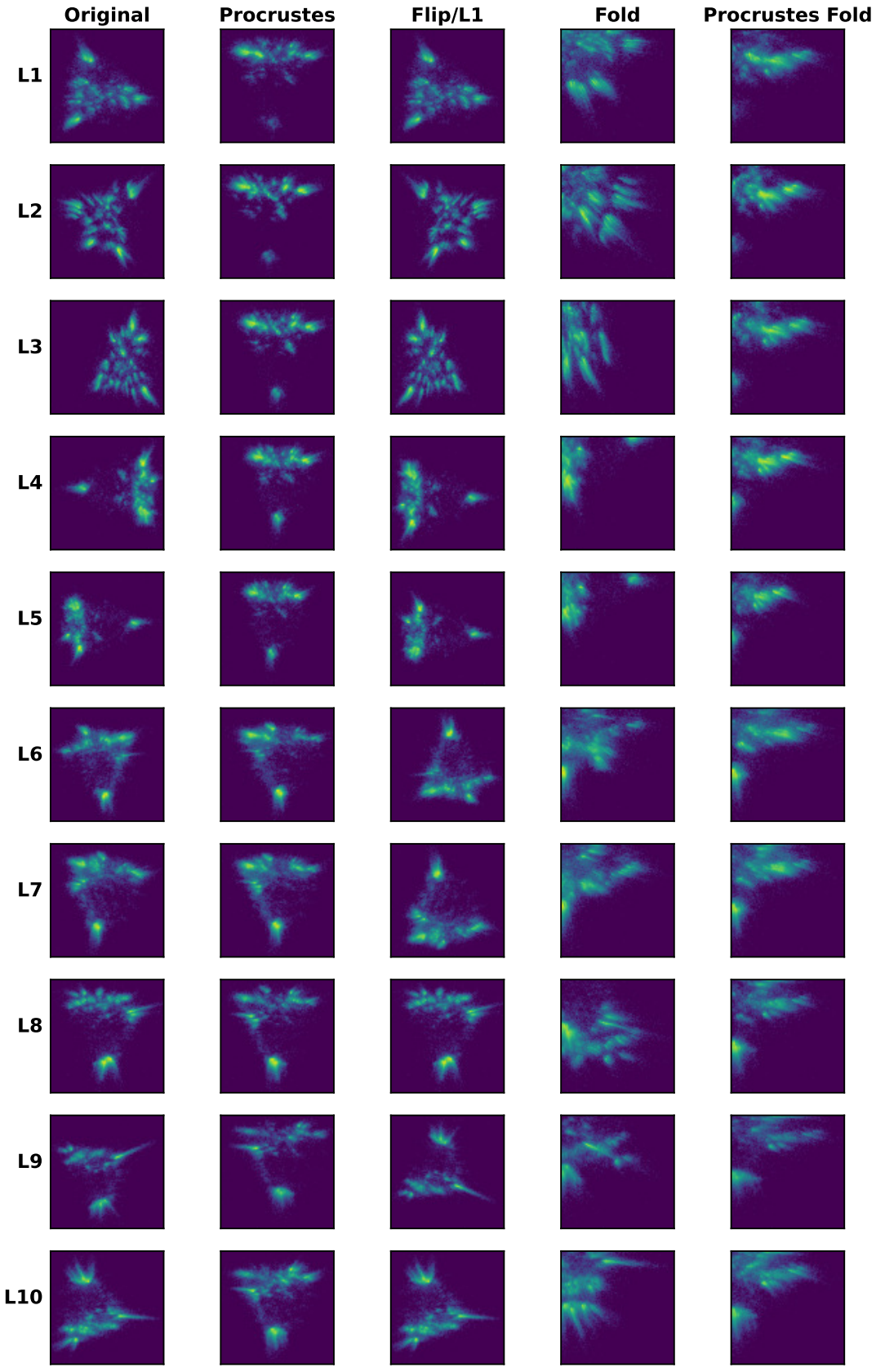


Figure 5.25 – S5 Sample aligned using various methods.

one or more principal axes will appear identical after folding. This method is also very fast to compute compared to the flip method, even using heuristics.

We performed experiments showing that these approaches can be used to approximate a matrix using uniform sampling without the need to access the off-(block-)diagonal elements, which may not be available in practice. We compared the point clouds of the large **S5** dataset computed using RSVD-MDS and RsEVD-MDS using only flips. We validated that by using only flips, we can reconstruct the complete Long Reads ABCD sample from its diagonal blocks alone.

In a third experiment, we demonstrated the limitations of the flip and fold approach when the dataset under consideration is not uniformly distributed by attempting to rebuild the full **S5** sample using only its diagonal blocks. Since these samples are not independent because they were extracted at different times of the year, the underlying structure of each point cloud is not the same, as we showed in Chapter 3. Thus, attempting to use Flip or Fold on this dataset will produce irrelevant results because we are not comparing data projected on the same principal axes.

Conclusion

The first part of this thesis, presented in Chapter 2 revisited the numerical and HPC design of randomized linear embedding based MDS summarized in Chapter 1. We showed that both RSVD-MDS and RsEVD-MDS achieve an excellent numerical performance when the numerical rank is well captured. When it is not well captured, there is a numerical trade-off. On the one hand, the RSVD approximates G better than the RsEVD due to the fact that it performs only a single projection ($G \approx QQ^TG$) whereas the RsEVD performs two projections ($G \approx QQ^TGQQ^T$). On the other hand, the RSVD-MDS can be further penalised by the loss of symmetry. The study has revealed that in practice this additional penalty is not dramatic as long as the approximated Gram matrix QQ^TG remains relatively close to G, as in this case its skew-symmetry ratio remains low.

As a consequence, in a general-purpose MDS library, it may be worthwhile to include both the RSVD-MDS and RsEVD-MDS variants. While we had an RSVD-MDS (presented in the background material in Chapter 1), we have implemented an HPC version of the RsEVD-MDS variant. From a numerical point of view, it corresponds to the variant discussed in Section 2.2. From an HPC point of view, we have followed the task-based design from Section 1.6.

Both RSVD-MDS and RsEVD-MDS are dominated by matrix-matrix multiplications (MM). The performance and memory consumption of MM are thus keys for that of the overall MDS based on such randomized linear embedding algorithms, which motivated a study on the improvement of distributed-memory matrix multiplication. We experimentally confirmed that reference distributed-memory libraries achieve a lower performance with SYMM than with GEMM. We showed that an efficient design of the communication schemes can significantly alleviate this gap. Still, we showed that part of the gap is explained by a lower AI of 2D BC SYMM compared to 2D GEMM (by a factor of 2). We considered two alternative data distributions, SBC and TBC. SBC is a direct adaptation to the matrix multiplication case of a study of the Cholesky decomposition [14]. TBC is a distributed-memory (and even a parallel) adaptation of the ideas behind TBS [15], a sequential out-of-core algorithm. We proved that SBC and TBC improve the AI of SYMM by a factor of $\sqrt{2}$ and 2, respectively, thus in particular equaling that of 2D BC GEMM for the latter one. In the case where we allow SYMM to store an amount of memory equivalent to a full matrix as 2D BC GEMM does, we furthermore showed that 2.5D TBC with $s = 2$ slices achieves a higher AI than 2D BC GEMM by a factor of $\sqrt{2}$. Our experimental study showed that the improvement of the AI translates into a compelling performance enhancement, up to the point of roughly matching GEMM performance. However, the highest AI does not always translate into the best performance.

The resulting code has been integrated in the randomized linear embedding algorithms used in the MDS framework of Section 1.6 in place of the two dense matrix multiplications representing the main computational step of the algorithm. While one had to trade-off [7] between performance, with GEMM, or memory, with SYMM, we showed that, altogether, the proposed STF design and the new TBC distribution now achieve a performance competitive with GEMM. This study also showed that algorithms involving very irregular data and task distributions can

now be implemented with a code easy to write, read and maintain thanks to the latest developments on the scalability of the STF model [5], while ensuring a competitive performance.

As a result of this work, we now have access to a versatile high-performance MDS library that provides an implementation of both RSVD-MDS and RsEVD-MDS in a distributed-memory context. These algorithms are based on a state-of-the-art STF implementation of SYMM which makes no compromise between performance and memory footprint.

The second part of this thesis dealt with the comparison of point clouds, either as an end in itself (chapters 3 and 5) or as a means to design new iterative randomized sampling algorithms (Chapter 4), with a special interest in the data access patterns of the dissimilarity matrix, which is expensive to compute in practice.

In Chapter 3, we explored the comparison of point cloud arising from MDS. Our first goal was to be able to compare the different $(L_1, \ldots, L_{10})$ point clouds that make up the S5 dataset of Section 1.8.4 (p. 21). We have shown that computing each L_i independently leads to point clouds that are not directly comparable and must first be realigned.

We proposed to rely on the Hausdorff distance to evaluate the distance between point clouds based on their shapes. We presented a new variation of the Hausdorff distance which we related to the Frobenius distance and also proposed a definition for a relative Hausdorff distance.

We identified four issues (**1: Axes permutation, 2: Rotation, 3: Translation and 4: Reflections**) that can affect the alignment of point clouds and showed how to solve them using orthogonal Procrustes analysis and centering. We showed that when comparing point clouds that have no correspondence between their points, it is possible to use landmarks to act as a reference and find the transformation to align these point clouds. Such landmarks can be obtained either by augmenting one point cloud with points from the other, or by selecting points in both point clouds to compute a smaller reference MDS.

Aligning point clouds for comparison also allows one to superimpose the aligned point clouds and reconstruct the full point cloud. This can be thought of as a different view of FastMDS [130], originally motivated by the comparison of the intermediate point clouds rather than the direct calculation of MDS. We call this method Block Diagonal Landmark Procrustes MDS (BDLPMDS).

We applied the BDLPMDS algorithm to the L_i blocks and were able to reconstruct the full S5 point cloud using a single computing node, whereas previously it had only been possible using a distributed-memory MDS over 100 nodes.

In chapter 4, we discussed and implemented methods to construct a reduced MDS, i.e. a subset of the points in the cloud using randomized *uniform sampling*. We discussed two questions that arise from sampling the distance matrix in the context of MDS:

— Question Q1 - How to approximate a reference input sample ?
— Question Q2 - How to estimate the intrinsic quality of a sample ?

Q1 can be seen as the issue of iterative uniform sampling with a reference sample while Q2 corresponds to iterative uniform sampling without a reference sample. We showed that both relative Modified Hausdorff (MHD_r) and relative Squared Modified Hausdorff ($H_{2,r}$) can be used as *robust a posteriori* estimators to answer both questions.

Based on these results, we proposed iterative MDS algorithms that use a stopping criterion based on the distance between the resampled point clouds at each iteration. We proposed four algorithms, each following the same resampling and distance computation step and differing only in how they compute the MDS at a given iteration. First, we proposed two conservative algorithms that compute each iteration without relying on the previous results and two progressive algorithms based on LMDS and BDLPMDS, that aim to build the next iteration by extending the result of the previous one, and both show promising results.

In the final chapter of this thesis, we explored the possibility of aligning the point clouds without the need for landmarks when dealing with point clouds extracted in an independent and identically distributed manner from a larger sample. We showed that when this condition is respected, the point clouds represent the same population and the MDS captures it correctly up to any number of reflections around principal axes. This problem is due to the non-unicity of the sEVD, and as such, it is not possible to prevent it from occurring.

We proposed two strategies for aligning such point clouds. The first, which we call flip, is to try all possible reflections and find the best one based on the distance presented in Chapter 3. Since this method can become computationally expensive when the number of dimensions considered increases, we have explored heuristics. The second strategy we proposed is called folding. It consists of removing the sign from the coordinates of all the points in the cloud. While this method is destructive in the sense that it irreversibly transforms the point cloud, it allows to map all possible reflections around all axes into the same point cloud. This method is very fast to compute compared to the flip method, even using heuristics.

We performed experiments showing that these approaches can be used to approximate a matrix using uniform sampling without the need to access the off-(block-)diagonal elements, which may not be available in practice. We compared the point clouds of the large S5 dataset computed using RSVD-MDS and RsEVD-MDS using only flips. We validated that by using only flips, we can reconstruct the complete Long Reads ABCD sample from its diagonal blocks alone.

Regarding the perspective of the thesis, we showed that in the context of distributed-memory MM, the highest AI does not always translate into the best performance, in particular for the 2.5D variant of MM using our new irregular data distribution, that struggled to achieve higher performance than classical 2D algorithms. A preliminary analysis hints that it is due to the scheduling of the communications that certainly must be reconsidered with the proposed new irregular data distributions.

Although we have pointed out the limitations of the BDLPMDS algorithm in the context of iterative MDS, it still performed well in 3 of the 4 test cases. We expect that a revised version of this algorithm will provide a fast and reliable method for computing iterative MDS.

As we mentioned earlier, the two questions Q1 and Q2 can be linked. We plan to design an algorithm that can decide when to stop scanning the data (in the sense of Question Q2) using uniform sampling in order to obtain an intermediate reference sample, and then to compress that data further. This additional step corresponds to answering Question Q1 with the intermediate reference sample as input. Since we know that Question Q1 is generally best answered with importance sampling, we could perform the second step with importance sampling.

We also plan to propose implementations of the fast methods of computing MDS (LMDS, BDLPMDS, iterative MDS) presented in the second part of the thesis in our high-performance MDS library, in addition to the already available randomized linear embedding algorithms.

Bibliography

[1] Mohamed Anwar Abouabdallah, Nathalie Peyrard, and Alain Franc. « Does clustering of DNA barcodes agree with botanical classification directly at high taxonomic levels? Trees in French Guiana as a case study ». In: *Molecular Ecology Resources* 22.5 (2022), pp. 1746–1761.

[2] Ramesh C Agarwal, Susanne M Balle, Fred G Gustavson, Mahesh Joshi, and Prasad Palkar. « A three-dimensional approach to parallel matrix multiplication ». In: *IBM Journal of Research and Development* 39.5 (1995), pp. 575–582.

[3] Emmanuel Agullo, Cédric Augonnet, Jack Dongarra, Hatem Ltaief, Raymond Namyst, Samuel Thibault, and Stanimire Tomov. « A hybridization methodology for high-performance linear algebra software for GPUs ». In: *GPU Computing Gems Jade Edition*. Elsevier, 2012, pp. 473–484.

[4] Emmanuel Agullo, Olivier Aumage, Mathieu Faverge, Nathalie Furmento, Florent Pruvost, Marc Sergent, and Samuel Paul Thibault. « Achieving high performance on supercomputers with a sequential task-based programming model ». In: *IEEE Transactions on Parallel and Distributed Systems* (2017).

[5] Emmanuel Agullo, Alfredo Buttari, Abdou Guermouche, Julien Herrmann, and Antoine Jego. *Task-Based Parallel Programming for Scalable Algorithms: Application to Matrix Multiplication*. Research Report RR-9461. Inria Bordeaux - Sud-Ouest, 2022, p. 26.

[6] Emmanuel Agullo, Alfredo Buttari, Abdou Guermouche, and Florent Lopez. « Implementing multifrontal sparse solvers for multicore architectures with sequential task flow runtime systems ». In: *ACM Transactions on Mathematical Software* (2016).

[7] Emmanuel Agullo, Olivier Coulaud, Alexandre Denis, Mathieu Faverge, Alain A. Franc, Jean-Marc Frigerio, Nathalie Furmento, Samuel Thibault, Adrien Guilbaud, Emmanuel Jeannot, Romain Peressoni, and Florent Pruvost. *Task-based randomized singular value decomposition and multidimensional scaling*. Research Report 9482. Inria Bordeaux - Sud Ouest ; Inrae - BioGeCo, Sept. 2022, p. 37.

[8] Noga Alon, Yossi Matias, and Mario Szegedy. « The space complexity of approximating the frequency moments ». In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 20–29.

[9] Isabelle Auby, Claire Meteigner, Myriam Rumebe, Emilie Chancerel, Franck Salin, Christelle Aluome, Frédéric Barraquand, Laure Carassou, Yolanda Del Amo, Vona Meleder, Alexandra Petit, Coralie Picoche, Jean-Marc Frigerio, and Alain Franc. *Malabar project: datasets of pairwise distances between reads per sample for rbcL marker.* Version V1. 2022. URL: https://doi.org/10.57745/BAMEWX.

[10] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. « StarPU: a unified platform for task scheduling on heterogeneous multicore architectures ». In: *Concurrency and Computation: Practice and Experience* 23.2 (2011), pp. 187–198.

[11] Seung-Hee Bae. « Parallel multidimensional scaling performance on multicore systems ». In: *2008 IEEE Fourth International Conference on eScience*. IEEE. 2008, pp. 695–702.

[12] Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. « High performance multidimensional scaling for large high-dimensional data visualization ». In: *IEEE Transaction of Parallel and Distributed System* (2012).

[13] Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. « Communication-Optimal Parallel Algorithm for Strassen's Matrix Multiplication ». In: *Proceedings of the Twenty-Fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA '12. Pittsburgh, Pennsylvania, USA, 2012, pp. 193–204. ISBN: 9781450312134.

[14] Olivier Beaumont, Philippe Duchon, Lionel Eyraud-Dubois, Julien Langou, and Mathieu Vérité. « Symmetric Block-Cyclic Distribution: Fewer Communications Leads to Faster Dense Cholesky Factorization ». In: *Supercomputing*. 2022.

[15] Olivier Beaumont, Lionel Eyraud-Dubois, Mathieu Vérité, and Julien Langou. « I/O-Optimal Algorithms for Symmetric Linear Algebra Kernels ». In: *arXiv preprint arXiv:2202.10217* (2022).

[16] Mario Bebendorf. « Adaptive cross approximation of multivariate functions ». In: *Constructive approximation* 34 (2011), pp. 149–179.

[17] E Beltrami. In: *Giornale di Matematiche ad Uso degli Studenti Delle Universita* (1873).

[18] L. Susan Blackford, Jaeyoung Choi, Andrew J. Cleary, Eduardo F. D'Azevedo, James Demmel, Inderjit S. Dhillon, Jack J. Dongarra, Sven Hammarling, Greg Henry, Antoine Petitet, Ken Stanley, David W. Walker, and R. Clinton Whaley. « ScaLAPACK: A Linear Algebra Library for Message-Passing Computers ». In: *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, PPSC 1997, Hyatt Regency Minneapolis on Nicollel Mall Hotel, Minneapolis, Minnesota, USA, March 14-17, 1997*. SIAM, 1997.

[19] Pierre Blanchard. « Fast hierarchical algorithms for the low-rank approximation of matrices, with applications to materials physics, geostatistics and data analysis. » PhD thesis. Université de Bordeaux, 2017. URL: `https://tel.archives-ouvertes.fr/tel-01534930`.

[20] Pierre Blanchard, Philippe Chaumeil, Jean-Marc Frigerio, Frédéric Rimet, Franck Salin, Sylvie Thérond, Olivier Coulaud, and Alain Franc. *A geometric view of Biodiversity: scaling to metagenomics*. 2018.

[21] Pierre Blanchard, Olivier Coulaud, Eric Darve, and Alain Franc. « FMR: Fast randomized algorithms for covariance matrix computations ». In: *Platform for Advanced Scientific Computing (PASC)*. 2016.

[22] Ulrik Brandes and Christian Pich. « Eigensolver methods for progressive multidimensional scaling of large data ». In: *Graph Drawing: 14th International Symposium, GD 2006, Karlsruhe, Germany, September 18-20, 2006. Revised Papers 14*. Springer. 2007, pp. 42–53.

[23] Rasmus Bro and Age K Smilde. « Centering and scaling in component analysis ». In: *Journal of chemometrics* 17.1 (2003), pp. 16–33.

[24] Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra. « Parallel tiled QR factorization for multicore architectures ». In: *Concurrency and Computation: Practice and Experience* 20.13 (2008), pp. 1573–1590.

[25] Henri Caron, Jean-François Molino, Daniel Sabatier, Patrick Léger, Philippe Chaumeil, Caroline Scotti-Saintagne, Jean-Marc Frigério, Ivan Scotti, Alain Franc, and Rémy J Petit. « Chloroplast DNA variation in a hyperdiverse tropical tree community ». In: *Ecology and Evolution* 9.8 (2019), pp. 4897–4905.

[26] Tony F Chan. « An improved algorithm for computing the singular value decomposition ». In: *ACM Transactions on Mathematical Software* 8.1 (1982), pp. 72–83.

[27] Tony F Chan and Wing Lok Wan. « Analysis of projection methods for solving linear systems with multiple right-hand sides ». In: *SIAM Journal on Scientific Computing* 18.6 (1997), pp. 1698–1721.

[28] Jong Youl Choi, Seung-Hee Bae, Xiaohong Qiu, and Geoffrey Fox. « High performance dimension reduction and visualization for large high-dimensional data analysis ». In: *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE. 2010, pp. 331–340.

[29] T.F. Cox and M. A. A. Cox. *Multidimensional Scaling - Second edition*. Vol. 88. Monographs on Statistics and Applied Probability. Chapman & al., 2001.

[30] Jane Cullum and William E Donath. « A block Lanczos algorithm for computing the q algebraically largest eigenvalues and a corresponding eigenspace of large, sparse, real symmetric matrices ». In: *1974 IEEE Conference on Decision and Control including the 13th Symposium on Adaptive Processes*. IEEE. 1974, pp. 505–509.

[31] Vin De Silva and Joshua B Tenenbaum. *Sparse multidimensional scaling using landmark points*. Tech. rep. technical report, Stanford University, 2004.

[32] Vahid Dehdari and Clayton V Deutsch. « Applications of randomized methods for decomposing and simulating from large covariance matrices ». In: *Geostatistics Oslo 2012*. Springer, 2012, pp. 15–26.

[33] James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou. « Communication-optimal parallel and sequential QR and LU factorizations: theory and practice ». In: *arXiv preprint arXiv:0806.2159* (2008).

[34] James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou. « Communication-optimal parallel and sequential QR and LU factorizations ». In: *SIAM Journal on Scientific Computing* 34.1 (2012), A206–A239.

[35] Alexandre Denis. « pioman: a pthread-based Multithreaded Communication Engine ». In: *Euromicro International Conference on Parallel, Distributed and Network-based Processing*. Turku, Finland, Mar. 2015. URL: https://hal.inria.fr/hal-01087775.

[36] Alexandre Denis, Emmanuel Jeannot, Philippe Swartvagher, and Samuel Thibault. « Using Dynamic Broadcasts to Improve Task-Based Runtime Performances ». In: *Euro-Par 2020: Parallel Processing*. Ed. by Maciej Malawski and Krzysztof Rzadca. Cham: Springer International Publishing, 2020, pp. 443–457. ISBN: 978-3-030-57675-2.

[37] Ivan Dokmanic, Reza Parhizkar, Juri Ranieri, and Martin Vetterli. « Euclidean distance matrices: essential theory, algorithms, and applications ». In: *IEEE Signal Processing Magazine* 32.6 (2015), pp. 12–30.

[38] Jack Dongarra, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, and Ichitaro Yamazaki. « The singular value decomposition: Anatomy of optimizing an algorithm for extreme scale ». In: *SIAM review* 60.4 (2018), pp. 808–865.

[39] Petros Drineas, Michael W Mahoney, and Nello Cristianini. « On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning. » In: *journal of machine learning research* 6.12 (2005).

[40] M.-P. Dubuisson and A.K. Jain. « A modified Hausdorff distance for object matching ». In: *Proceedings of 12th International Conference on Pattern Recognition*. - 1994. URL: https://doi.org/10.1109/icpr.1994.576361.

[41] Alejandro Duran, Eduard Ayguadé, Rosa M Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, and Judit Planas. « Ompss: a proposal for programming heterogeneous multi-core architectures ». In: *Parallel processing letters* (2011).

[42] Carl Eckart and Gale Young. « The Approximation of One Matrix By Another of Lower Rank ». In: *Psychometrika* 1.3 (1936), pp. 211–218. URL: https://doi.org/10.1007/bf02288367.

[43] Jocelyne Erhel and Frédéric Guyomarc'h. « An augmented conjugate gradient method for solving consecutive symmetric positive definite linear systems ». In: *SIAM Journal on Matrix Analysis and Applications* 21.4 (2000), pp. 1279–1299.

[44] Christos Faloutsos and King-Ip Lin. « FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets ». In: *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*. 1995, pp. 163–174.

[45] Philippe Flajolet and G Nigel Martin. « Probabilistic counting ». In: *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*. IEEE. 1983, pp. 76–82.

[46] Robin Floyd, Eyualem Abebe, Artemis Papert, and Mark Blaxter. « Molecular barcodes for soil nematode identification ». In: *Molecular ecology* 11.4 (2002), pp. 839–850.

[47] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard – version 4.0*. June 2021.

[48] Alain Franc, Jean-Marc Frigerio, Emilie Chancerel, Franck Salin, Sylvie Thérond, Frédéric Rimet, and Agnès Bouchez. *Reads and pairwise distances from 10 samples of diatoms in Geneva lake*. Version V1. 2023. URL: https://doi.org/10.57745/NKTRHO.

[49] Jean-Marc Frigerio, Henri Caron, Daniel Sabatier, Jean-François Molino, and Alain Franc. *Guiana Trees*. Version V2. 2021. URL: https://doi.org/10.15454/XSJ079.

[50] Mark Gates, Jakub Kurzak, Ali Charara, Asim YarKhan, and Jack Dongarra. « SLATE: Design of a Modern Distributed and Accelerated Linear Algebra Library ». In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '19. Denver, Colorado: Association for Computing Machinery, 2019. ISBN: 9781450362290.

[51] Robert van de Geijn and Jerrell Watts. « SUMMA: scalable universal matrix multiplication algorithm ». In: *CONCURRENCY: PRACTICE AND EXPERIENCE* 9.4 (1997), pp. 255–274. URL: http://www.netlib.org/lapack/lawnspdf/lawn96.pdf.

[52] Gene Golub and William Kahan. « Calculating the singular values and pseudo-inverse of a matrix ». In: *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis* 2.2 (1965), pp. 205–224.

[53] Gene H Golub and Christian Reinsch. « Singular value decomposition and least squares solutions ». In: *Linear algebra*. Springer, 1971, pp. 134–151.

[54] Gene H Golub, R Underwood, and James H Wilkinson. « The Lanczos algorithm for the symmetric Ax= λBx problem ». In: *Techn. Rep. STAN-CS-72–270, Stanford University* (1972).

[55] Gene H Golub and Richard Underwood. « The block Lanczos method for computing eigenvalues ». In: *Mathematical software*. Elsevier, 1977, pp. 361–377.

[56] Gene H Golub and Charles F Van Loan. *Matrix computations*. Vol. 3. JHU press, 2013.

[57] Gene Howard Golub. « Least squares, singular values and matrix approximations ». In: *Aplikace matematiky* 13.1 (1968), pp. 44–51.

[58] John C Gower. « Generalized procrustes analysis ». In: *Psychometrika* 40 (1975), pp. 33–51.

[59] John Clifford Gower. « Adding a point to vector diagrams in multivariate analysis ». In: *Biometrika* 55.3 (1968), pp. 582–585.

[60] Edouard Grave, Armand Joulin, and Quentin Berthet. « Unsupervised alignment of embeddings with wasserstein procrustes ». In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR. 2019, pp. 1880–1890.

[61] Laura Grigori, Sophie Moufawad, and Frédéric Nataf. « Enlarged Krylov subspace conjugate gradient methods for reducing communication ». In: *SIAM Journal on Matrix Analysis and Applications* 37.2 (2016), pp. 744–773.

[62] Mikhael Gromov, Misha Katz, Pierre Pansu, and Stephen Semmes. *Metric structures for Riemannian and non-Riemannian spaces*. Vol. 152. Springer, 1999.

[63] Bilel Hadri, Hatem Ltaief, Emmanuel Agullo, and Jack Dongarra. « Tile QR factorization with parallel panel processing for multicore architectures ». In: *2010 IEEE International Symposium on Parallel & Distributed Processing*. IEEE. 2010, pp. 1–10.

[64] N. Halko, P. G. Martinsson, and J. A. Tropp. « Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions ». In: *SIAM Review* 53.2 (2011), pp. 217–288. eprint: https://doi.org/10.1137/090771806. URL: https://doi.org/10.1137/090771806.

[65] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York, 2009, pp. 539–541. URL: https://doi.org/10.1007/978-0-387-84858-7.

[66] Paul DN Hebert, Sujeevan Ratnasingham, and Jeremy R De Waard. « Barcoding animal life: cytochrome c oxidase subunit 1 divergences among closely related species ». In: *Proceedings of the Royal Society of London. Series B: Biological Sciences* 270.suppl_1 (2003), S96–S99.

[67] Thomas Herault, Yves Robert, George Bosilca, and Jack Dongarra. « Generic Matrix Multiplication for Multi-GPU Accelerated Distributed-Memory Platforms over PaRSEC ». In: *ScalA 2019 - IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*. Denver, United States: IEEE, Nov. 2019, pp. 33–41.

[68] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[69] Harold Hotelling. « Analysis of a complex of statistical variables with principal components ». In: *J. Educ. Psy.* 24 (1933), pp. 498–520.

[70] D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. « Comparing Images Using the Hausdorff Distance ». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.9 (1993), pp. 850–863. URL: https://doi.org/10.1109/34.232073.

[71] A. J. Izenman. *Modern Multivariate Statistical Techniques*. NY: Springer, 2008, p. 731.

[72] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[73] Hao Ji and Yaohang Li. « GPU accelerated randomized singular value decomposition and its application in image compression ». In: *Proc. of MSVESCC* (2014), pp. 39–45.

[74] Ian Jolliffe. « Principal component analysis ». In: *Encyclopedia of statistics in behavioral science* (2005).

[75] Camille Jordan. « Mémoire sur les formes bilinéaires. » In: *Journal de mathématiques pures et appliquées* 19 (1874), pp. 35–54.

[76] Suamporn Ketpreechasawat and Odest Chadwicke Jenkins. « Hierarchical landmark charting ». In: *Master's Thesis, Brown University* (2006).

[77] Jaeyeon Kim, Binh-Son Hua, Thanh Nguyen, and Sai-Kit Yeung. « Minimal adversarial examples for deep learning on 3d point clouds ». In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 7797–7806.

[78] Mirza Klimenta. « Extending the usability of multidimensional scaling for graph drawing ». PhD thesis. Citeseer, 2012.

[79] Donald E Knuth. « The art of computer programming, vol. 3: Searching and sorting ». In: *Reading MA: Addison-Wisley* (1973), pp. 543–583.

[80] Mario Köppen. « The curse of dimensionality ». In: *5th online world conference on soft computing in industrial applications (WSC5)*. Vol. 1. 2000, pp. 4–8.

[81] Joseph B Kruskal. « Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis ». In: *Psychometrika* 29.1 (1964), pp. 1–27.

[82] Grzegorz Kwasniewski, Marko Kabić, Maciej Besta, Joost VandeVondele, Raffaele Solcà, and Torsten Hoefler. « Red-Blue Pebbling Revisited: Near Optimal Parallel Matrix-Matrix Multiplication ». In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '19. Denver, Colorado, 2019. ISBN: 9781450362290.

[83] Charles L Lawson and Richard J Hanson. « Solving least squares problems ». In: *Prentice-Hall Series in Automatic Computation* (1974).

[84] John A Lee and Michel Verleysen. *Nonlinear dimensionality reduction*. Vol. 1. Springer, 2007.

[85] Seunghak Lee and Seungjin Choi. « Landmark MDS ensemble ». In: *Pattern recognition* 42.9 (2009), pp. 2045–2053.

[86] Jan de Leeuw. « Applications of Convex Analysis to Multidimensional Scaling ». In: 2000.

[87] U-Wai Lok, Pengfei Song, Joshua D Trzasko, Eric A Borisch, Ron Daigle, and Shigao Chen. « Parallel implementation of randomized singular value decomposition and randomized spatial downsampling for real time ultrafast microvessel imaging on a multicore cpus architecture ». In: *2018 IEEE International Ultrasonics Symposium (IUS)*. IEEE. 2018, pp. 1–4.

[88] Yue Lu, Chew Lim Tan, Weihua Huang, and Liying Fan. « An approach to word image matching based on weighted Hausdorff distance ». In: *Proceedings of Sixth International Conference on Document Analysis and Recognition*. 2001, pp. 921–925. URL: https://doi.org/10.1109/icdar.2001.953920.

[89] Yuechao Lu, Ichitaro Yamazaki, Fumihiko Ino, Yasuyuki Matsushita, Stanimire Tomov, and Jack Dongarra. « Reducing the amount of out-of-core data access for GPU-accelerated randomized SVD ». In: *Concurrency and Computation: Practice and Experience* 32.19 (2020), e5754.

[90] K. V. Mardia, J.T. Kent, and J. M. Bibby. *Multivariate Analysis*. Probability and Mathematical Statistics. Academic Press, 1979.

[91] Martinsson, Per Gunnar, Vladimir Rokhlin, and Mark Tygert. « A randomized algorithm for the decomposition of matrices ». In: *Applied and Computational Harmonic Analysis* 30.1 (2011), pp. 47–68.

[92] Per-Gunnar Martinsson. *Randomized methods for matrix computations*. 2016. URL: https://arxiv.org/abs/1607.01649.

[93] Per-Gunnar Martinsson and Joel A Tropp. « Randomized numerical linear algebra: Foundations and algorithms ». In: *Acta Numerica* 29 (2020), pp. 403–572.

[94] Facundo Mémoli and Guillermo Sapiro. « Comparing point clouds ». In: *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing - SGP '04*. - 2004, nil. URL: https://doi.org/10.1145/1057432.1057436.

[95] J Ian Munro and Mike S Paterson. « Selection and sorting with limited storage ». In: *Theoretical computer science* 12.3 (1980), pp. 315–323.

[96] Shanmugavelayutham Muthukrishnan et al. « Data streams: Algorithms and applications ». In: *Foundations and Trends® in Theoretical Computer Science* 1.2 (2005), pp. 117–236.

[97] Andy A Nikishin and A Yu Yeremin. « Variable block CG algorithms for solving large sparse symmetric positive definite linear systems on parallel computers, I: General iterative scheme ». In: *SIAM journal on matrix analysis and applications* 16.4 (1995), pp. 1135–1153.

[98] Dianne P O'Leary. « The block conjugate gradient algorithm and related methods ». In: *Linear algebra and its applications* 29 (1980), pp. 293–322.

[99] Emmanuel Paradis. « Multidimensional scaling with very large datasets ». In: *Journal of Computational and Graphical Statistics* 27.4 (2018), pp. 935–939.

[100] Emmanuel Paradis. « Reduced multidimensional scaling ». In: *Computational Statistics* 37.1 (2022), pp. 91–105.

[101] Beresford N Parlett. « A new look at the Lanczos algorithm for solving symmetric systems of linear equations ». In: *Linear algebra and its applications* 29 (1980), pp. 323–346.

[102] Beresford N Parlett. *The symmetric eigenvalue problem*. SIAM, 1998.

[103] J. Paumard and E. Aubourg. « Adjusting astronomical images using a censored Hausdorff distance ». In: *Proceedings of International Conference on Image Processing*. 1997, 232–235 vol.3. URL: https://doi.org/10.1109/icip.1997.632069.

[104] Piotr Pawliczek and Witold Dzwinel. « Parallel implementation of multidimensional scaling algorithm based on particle dynamics ». In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6067 LNCS.PART 1 (2010), pp. 312–321.

[105] Karl Pearson. « LIII. On lines and planes of closest fit to systems of points in space ». In: *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2.11 (1901), pp. 559–572.

[106] John Platt. « Fastmap, metricmap, and landmark mds are all nyström algorithms ». In: *International Workshop on Artificial Intelligence and Statistics*. PMLR. 2005, pp. 261–268.

[107] François Pompanon, Eric Coissac, and Pierre Taberlet. « Metabarcoding, une nouvelle façon d'analyser la biodiversité. » In: *Biofutur* 319 (2011), pp. 30–32.

[108] Jack Poulson, Bryan Marker, Robert A. van de Geijn, Jeff R. Hammond, and Nichols A. Romero. « Elemental: A New Framework for Distributed Memory Dense Matrix Computations ». In: *ACM Trans. Math. Softw.* 39.2 (Feb. 2013). ISSN: 0098-3500.

[109] Marion Webster Richardson. « Multidimensional psychophysics ». In: *Psychological Bulletin* 35 (1938), pp. 659–660.

[110] Vladimir Rokhlin, Arthur Szlam, and Mark Tygert. « A Randomized Algorithm for Principal Component Analysis ». In: *SIAM Journal on Matrix Analysis and Applications* 31.3 (Jan. 2010), pp. 1100–1124. ISSN: 0895-4798.

[111] Axel Ruhe. « Implementation aspects of band Lanczos algorithms for computation of eigenvalues of large sparse symmetric matrices ». In: *Mathematics of Computation* 33.146 (1979), pp. 680–687.

[112] Youcef Saad. « On the Lanczos method for solving symmetric linear systems with several right-hand sides ». In: *Mathematics of computation* 48.178 (1987), pp. 651–662.

[113] Yousef Saad. « Analysis of augmented Krylov subspace methods ». In: *SIAM Journal on Matrix Analysis and Applications* 18.2 (1997), pp. 435–449.

[114] Yousef Saad, Manshung Yeung, Jocelyne Erhel, and Frédéric Guyomarc'h. « A deflated version of the conjugate gradient algorithm ». In: *SIAM Journal on Scientific Computing* 21.5 (2000), pp. 1909–1926.

[115] Martin D. Schatz, Robert A. van de Geijn, and Jack Poulson. « Parallel matrix multiplication: a systematic journey ». In: *SIAM Journal on Scientific Computing* 38.6 (2016), pp. 748–781.

[116] Erhard Schmidt. « Zur Theorie der linearen und nichtlinearen Integralgleichungen. III. Teil ». In: *Mathematische Annalen* 65.3 (1908), pp. 370–399.

[117] Peter H Schönemann. « A generalized solution of the orthogonal procrustes problem ». In: *Psychometrika* 31.1 (1966), pp. 1–10.

[118] Vin Silva and Joshua Tenenbaum. « Global versus local methods in nonlinear dimensionality reduction ». In: *Advances in neural information processing systems* 15 (2002).

[119] Charles F Smith, Andrew F Peterson, and Raj Mittra. « A conjugate gradient algorithm for the treatment of multiple incident electromagnetic fields ». In: *IEEE Transactions on Antennas and Propagation* 37.11 (1989), pp. 1490–1493.

[120] Temple F Smith and Michael S Waterman. « Identification of common molecular subsequences ». In: *Journal of molecular biology* 147.1 (1981), pp. 195–197.

[121] Edgar Solomonik and James Demmel. « Communication-optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms ». In: *Proceedings of the 17th International Conference on Parallel Processing - Volume Part II*. Euro-Par'11. Bordeaux, France: Springer-Verlag, 2011, pp. 90–109. ISBN: 978-3-642-23396-8.

[122] Gilbert W Stewart. « On the early history of the singular value decomposition ». In: *SIAM review* 35.4 (1993), pp. 551–566.

[123] Louis L Thurstone. « Psychophysical analysis ». In: *The American journal of psychology* 38.3 (1927), pp. 368–389.

[124] Louis Leon Thurstone. « Theory of attitude measurement. » In: *Psychological review* 36.3 (1929), p. 222.

[125] Warren S. Torgerson. « Multidimensional scaling: I. Theory and method ». In: *Psychometrika* 17.4 (Dec. 1952), pp. 401–419. ISSN: 1860-0980. URL: `http://dx.doi.org/10.1007/bf02288916`.

[126] Francois Trahay, Alexandre Denis, Olivier Aumage, and Raymond Namyst. « Improving reactivity and communication overlap in MPI using a generic I/O manager ». In: *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer. 2007, pp. 170–177.

[127] Alice Valentini, François Pompanon, and Pierre Taberlet. « DNA barcoding for ecologists ». In: *Trends in ecology & evolution* 24.2 (2009), pp. 110–117.

[128] Alice Valentini, Pierre Taberlet, Claude Miaud, Raphaël Civade, Jelger Herder, Philip Francis Thomsen, Eva Bellemain, Aurélien Besnard, Eric Coissac, Frédéric Boyer, et al. « Next-generation monitoring of aquatic biodiversity using environmental DNA metabarcoding ». In: *Molecular ecology* 25.4 (2016), pp. 929–942.

[129] 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*. 1999, pp. 307–311.

[130] Tynia Yang, Jinze Liu, Leonard McMillan, and Wei Wang. « A fast approximation to multidimensional scaling ». In: *IEEE workshop on computation intensive methods for computer vision*. 2006.

[131] Asim YarKhan. « Dynamic task execution on shared and distributed memory architectures ». PhD thesis. University of Tennesee, 2012.

[132] Gale Young and A. S. Householder. « Discussion of a Set of Points in Terms of Their Mutual Distances ». In: *Psychometrika* 3.1 (1938), pp. 19–22. URL: https://doi.org/10.1007/bf02287916.

[133] Chong Ho Yu. « Resampling methods: concepts, applications, and justification ». In: *Practical Assessment, Research, and Evaluation* 8.1 (2002), p. 19.

[134] Harald Ziegelwanger, Piotr Majdak, and Wolfgang Kreuzer. « Numerical calculation of listener-specific head-related transfer functions and sound localization: Microphone model and mesh discretization ». In: *The Journal of the Acoustical Society of America* 138.1 (2015), pp. 208–222.

[135] Antanas Žilinskas and Julius Žilinskas. « Parallel genetic algorithm: Assessment of performance in multidimensional scaling ». In: *Proceedings of GECCO 2007: Genetic and Evolutionary Computation Conference* January 2007 (2007), pp. 1492–1501.

Appendices

Appendix A

Extra results from chapter 2

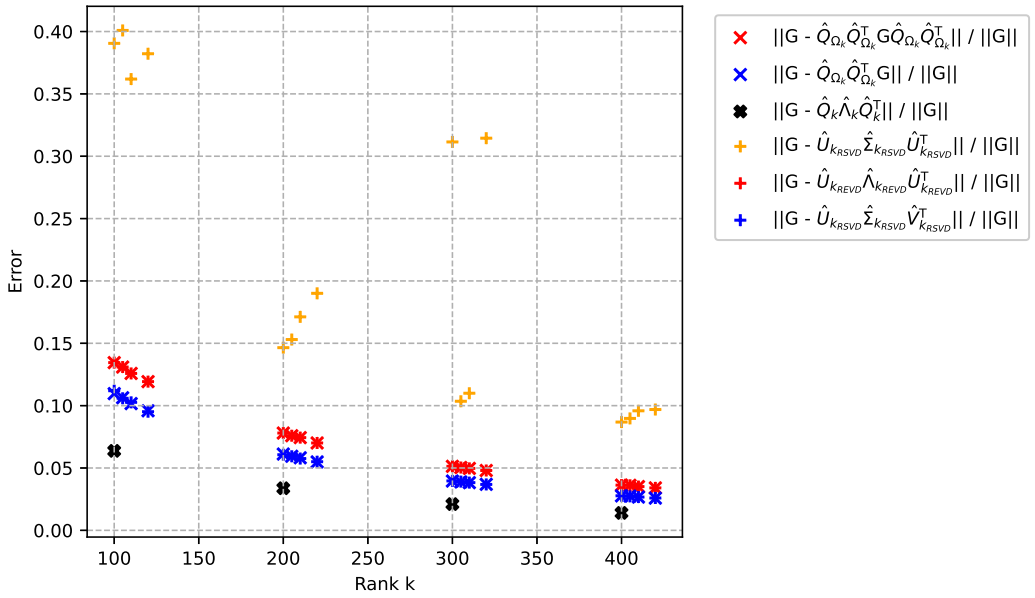


Figure A.1 – Relative error made with regards to the original Atlas Guyane matrix for each method at rank k. In black the Truncated sEVD, the best rank-k approximation, in red the RsEVD and in blue the RSVD. The crosses represent low-rank approximation and the pluses the decompositions. In yellow w present the RSVD with assumed symmetry (*i.e.* assuming $V = U$).

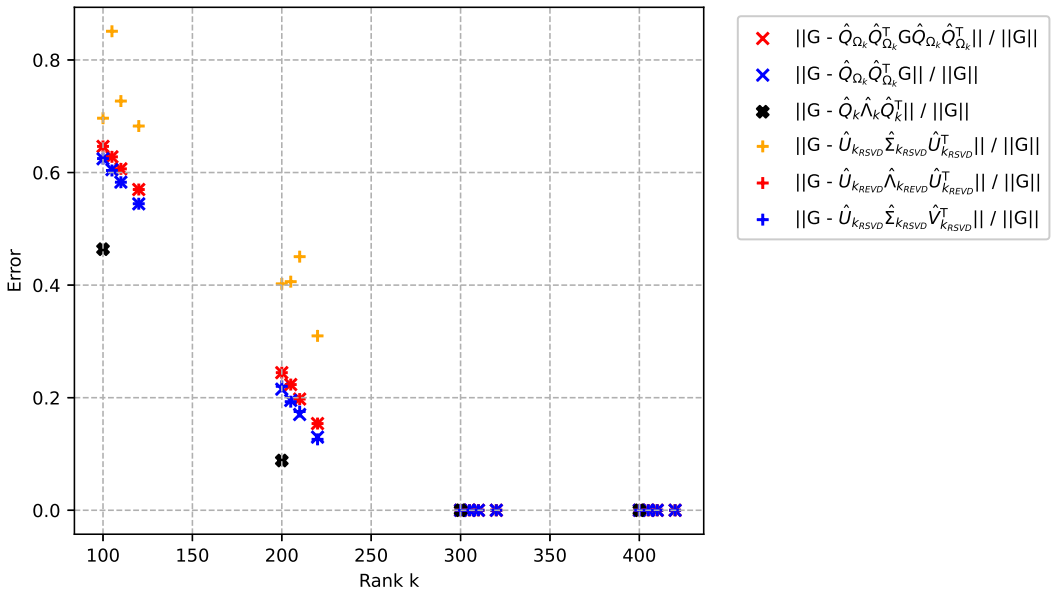


Figure A.2 – Relative error made with regards to the $1,000 \times 1,000$ synthetic matrix of rank 250 for each method at rank k. In black the Truncated sEVD, the best rank-k approximation, in red the RsEVD and in blue the RSVD. The crosses represent low-rank approximation and the pluses the decompositions. In yellow w present the RSVD with assumed symmetry (*i.e.* assuming $V = U$).

Appendix B

Extra results from chapter 3

B.1 Distance results between L_i/L_j pairs

	Direct extraction from full *S5*	Alignment with Xref =				
		S5	120,000	1,000	250	180
L2	2.49e-03	3.16e-03	5.69e-03	8.53e-03	1.44e-01	2.05e-02
L3	2.64e-03	3.32e-03	5.37e-03	7.61e-03	1.37e-01	1.73e-02
L4	2.95e-03	3.63e-03	4.34e-03	5.96e-03	5.88e-02	2.99e-01
L5	3.33e-03	4.02e-03	4.80e-03	6.19e-03	4.55e-02	2.57e-02
L6	5.85e-03	6.68e-03	8.94e-03	1.91e-02	7.16e-02	8.39e-02
L7	6.76e-03	7.63e-03	9.07e-03	1.89e-02	6.86e-02	3.54e-01
L8	6.47e-03	7.83e-03	1.26e-02	2.05e-02	1.16e-01	1.46e-01
L9	8.71e-03	1.10e-02	1.54e-02	2.50e-02	7.31e-02	8.00e-02
L10	8.12e-03	9.52e-03	1.48e-02	2.09e-02	2.04e-01	2.96e-02

Table B.1 – $MHD_r(L_1, L_j)_{j\in[\![2;10]\!]}$.

	Direct extraction from full *S5*	Alignment with Xref =				
		S5	120,000	1,000	250	180
L2	2.15e-01	2.64e-01	2.56e-01	2.66e-01	4.28e-01	3.48e-01
L3	1.27e-01	1.37e-01	1.81e-01	1.80e-01	4.20e-01	3.71e-01
L4	3.20e-01	2.97e-01	3.46e-01	3.71e-01	4.84e-01	4.40e-01
L5	5.21e-01	5.65e-01	5.48e-01	4.94e-01	6.17e-01	5.60e-01
L6	5.46e-01	4.73e-01	3.56e-01	4.89e-01	5.76e-01	5.84e-01
L7	4.15e-01	3.67e-01	4.24e-01	5.35e-01	6.39e-01	4.79e-01
L8	3.78e-01	3.98e-01	3.86e-01	5.37e-01	6.43e-01	5.97e-01
L9	4.74e-01	3.84e-01	3.82e-01	4.33e-01	4.86e-01	3.04e-01
L10	2.55e-01	2.62e-01	3.11e-01	3.26e-01	5.34e-01	2.60e-01

Table B.2 – $H_r(L_1, L_j)_{j\in[\![2;10]\!]}$.

B.2 Histogram of position for various datasets

	Direct extraction from full *S5*	Alignment with Xref =				
		S5	120,000	1,000	250	180
L2	1.49e-01	1.62e-01	2.97e-01	4.04e-01	4.21e+00	7.21e-01
L3	1.17e-01	1.22e-01	2.23e-01	4.19e-01	3.94e+00	6.97e-01
L4	1.98e-01	2.11e-01	2.04e-01	3.17e-01	1.82e+00	8.29e+00
L5	2.62e-01	2.85e-01	2.98e-01	3.65e-01	1.73e+00	6.68e-01
L6	3.35e-01	3.50e-01	4.01e-01	7.39e-01	2.32e+00	2.98e+00
L7	3.56e-01	3.65e-01	4.03e-01	7.09e-01	2.08e+00	1.70e+01
L8	3.20e-01	3.46e-01	5.20e-01	7.42e-01	3.71e+00	6.40e+00
L9	3.58e-01	3.97e-01	4.86e-01	8.05e-01	2.15e+00	2.45e+00
L10	3.81e-01	4.16e-01	5.66e-01	8.38e-01	7.68e+00	1.03e+00

Table B.3 – $H_2(L_1, L_j)_{j\in[\![2;10]\!]}$.

	Direct extraction from full *S5*	Alignment with Xref =				
		S5	120,000	1,000	250	180
L2	4.17e-02	5.29e-02	9.24e-02	1.55e-01	3.21e+00	2.86e-01
L3	4.43e-02	5.57e-02	8.77e-02	1.38e-01	2.82e+00	2.34e-01
L4	5.46e-02	6.72e-02	7.82e-02	1.14e-01	1.11e+00	7.58e+00
L5	6.33e-02	7.63e-02	9.02e-02	1.22e-01	7.14e-01	4.08e-01
L6	1.20e-01	1.37e-01	1.76e-01	4.00e-01	1.52e+00	1.64e+00
L7	1.35e-01	1.52e-01	1.78e-01	3.80e-01	1.38e+00	1.54e+01
L8	1.44e-01	1.74e-01	2.79e-01	4.56e-01	2.57e+00	4.38e+00
L9	1.64e-01	2.07e-01	2.66e-01	4.84e-01	1.33e+00	1.42e+00
L10	1.59e-01	1.87e-01	2.92e-01	4.19e-01	4.61e+00	5.99e-01

Table B.4 – $MHD(L_1, L_j)_{j\in[\![2;10]\!]}$.

	Direct extraction from full *S5*	Alignment with Xref =				
		S5	120,000	1,000	250	180
L2	9.49e+00	1.09e+01	1.04e+01	1.10e+01	2.02e+01	1.61e+01
L3	5.36e+00	5.71e+00	7.05e+00	7.17e+00	1.64e+01	1.72e+01
L4	1.61e+01	1.50e+01	1.57e+01	1.97e+01	2.21e+01	2.48e+01
L5	2.61e+01	2.94e+01	3.07e+01	2.75e+01	3.42e+01	3.57e+01
L6	2.75e+01	2.34e+01	1.83e+01	2.39e+01	2.83e+01	2.70e+01
L7	2.19e+01	1.87e+01	2.25e+01	2.88e+01	3.56e+01	3.92e+01
L8	1.88e+01	1.95e+01	1.96e+01	2.79e+01	3.49e+01	3.43e+01
L9	2.60e+01	2.09e+01	1.61e+01	2.34e+01	2.51e+01	1.58e+01
L10	1.34e+01	1.38e+01	1.59e+01	1.68e+01	3.05e+01	1.51e+01

Table B.5 – $H(L_1, L_j)_{j\in[\![2;10]\!]}$.

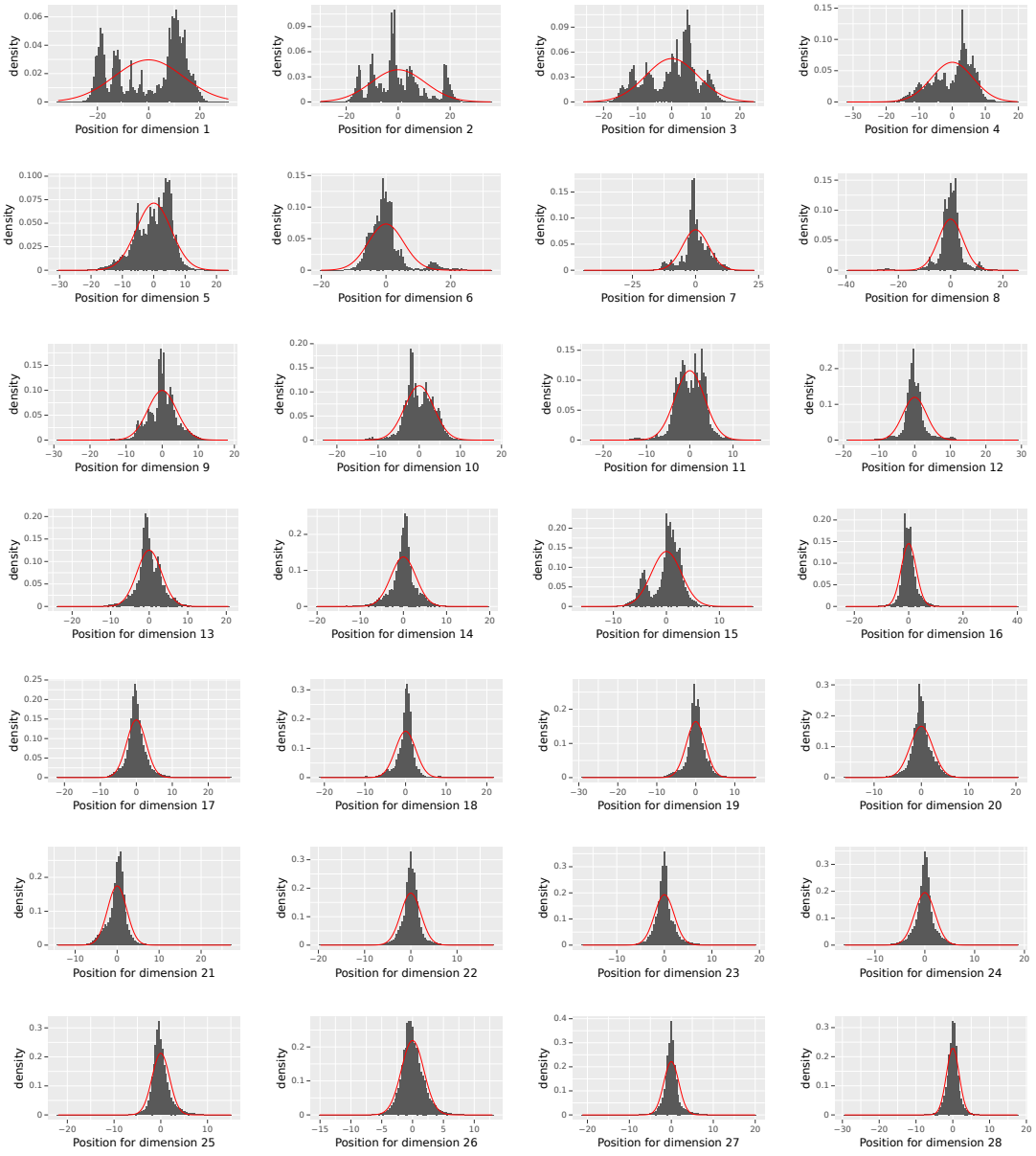


Figure B.1 – Histogram of the distribution of the coordinates per dimension in L1L2 for dimensions varying from 1 to 28.

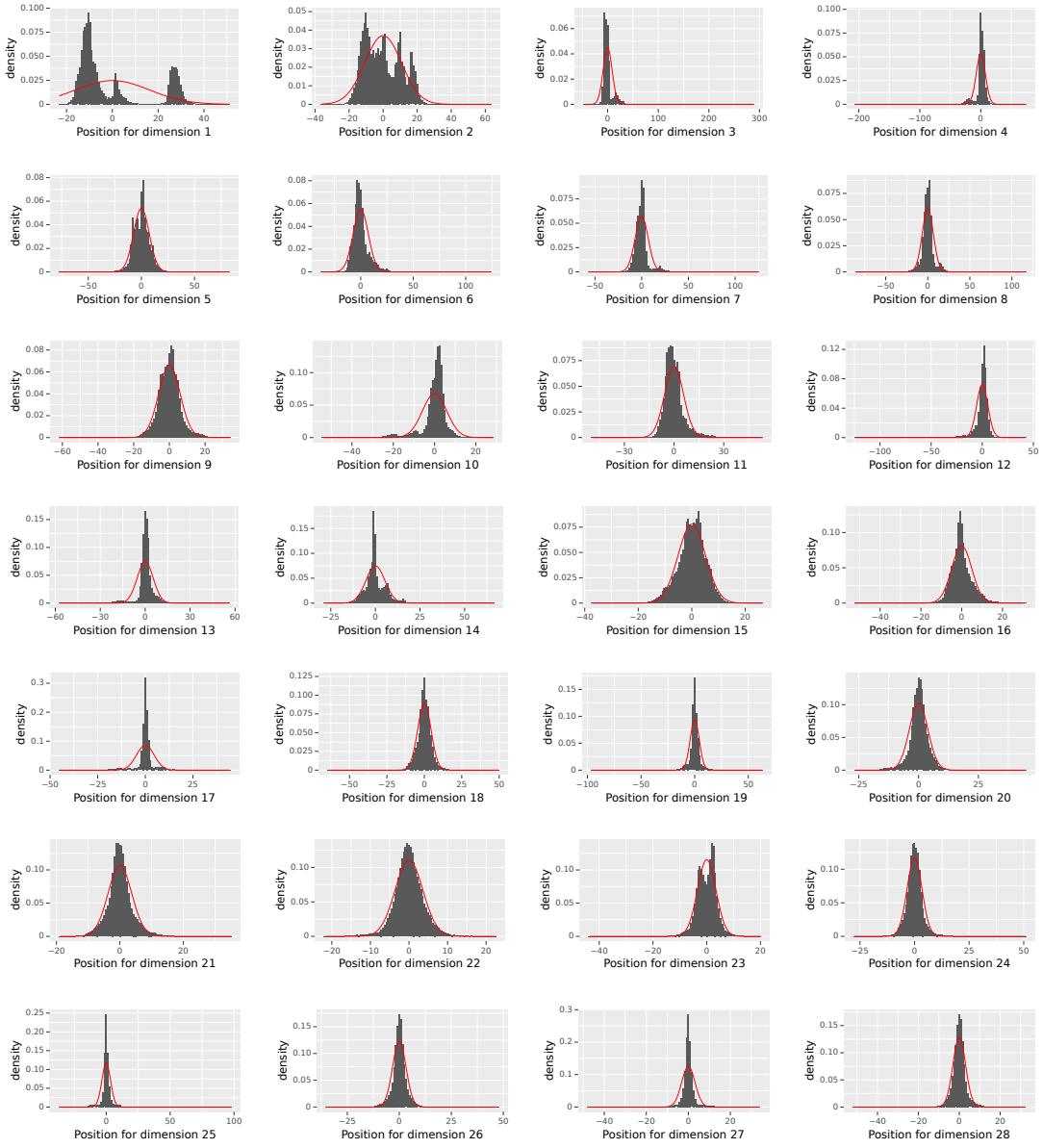


Figure B.2 – Histogram of the distribution of the coordinates per dimension in Lref for dimensions varying from 1 to 28.

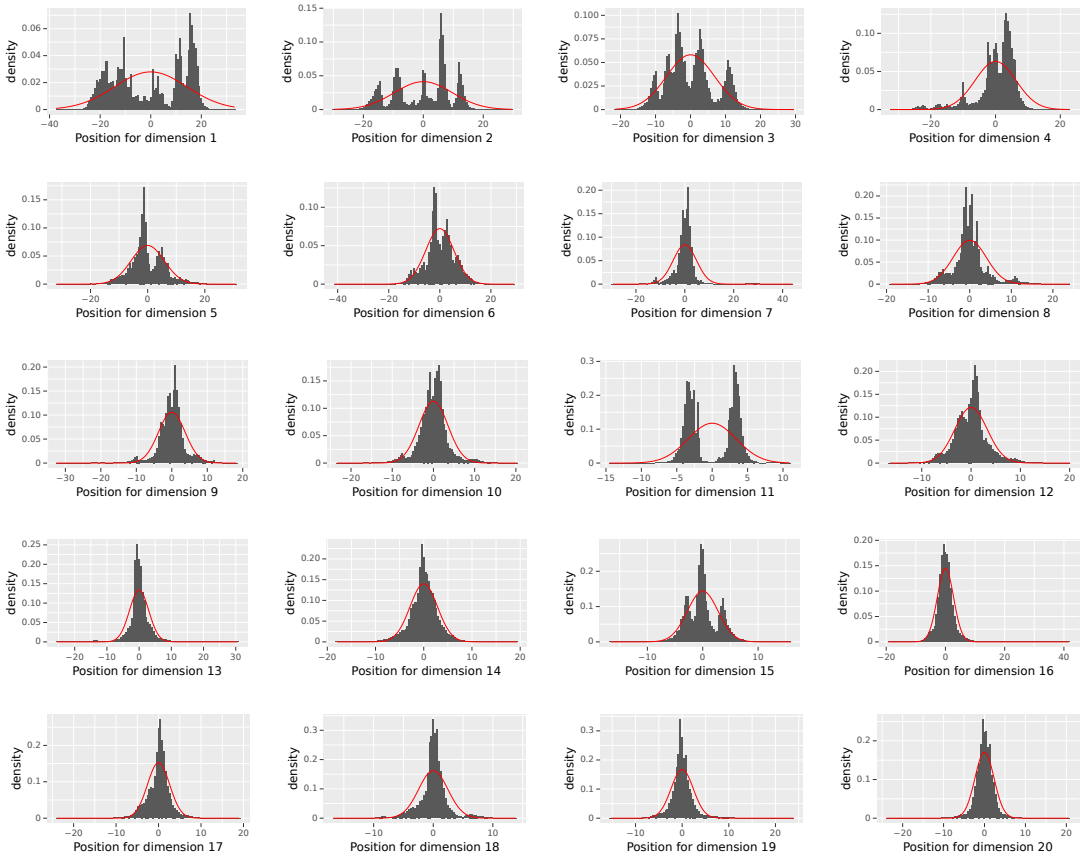


Figure B.3 – Histogram of the distribution of the coordinates per dimension in L2 for dimensions varying from 1 to 20.

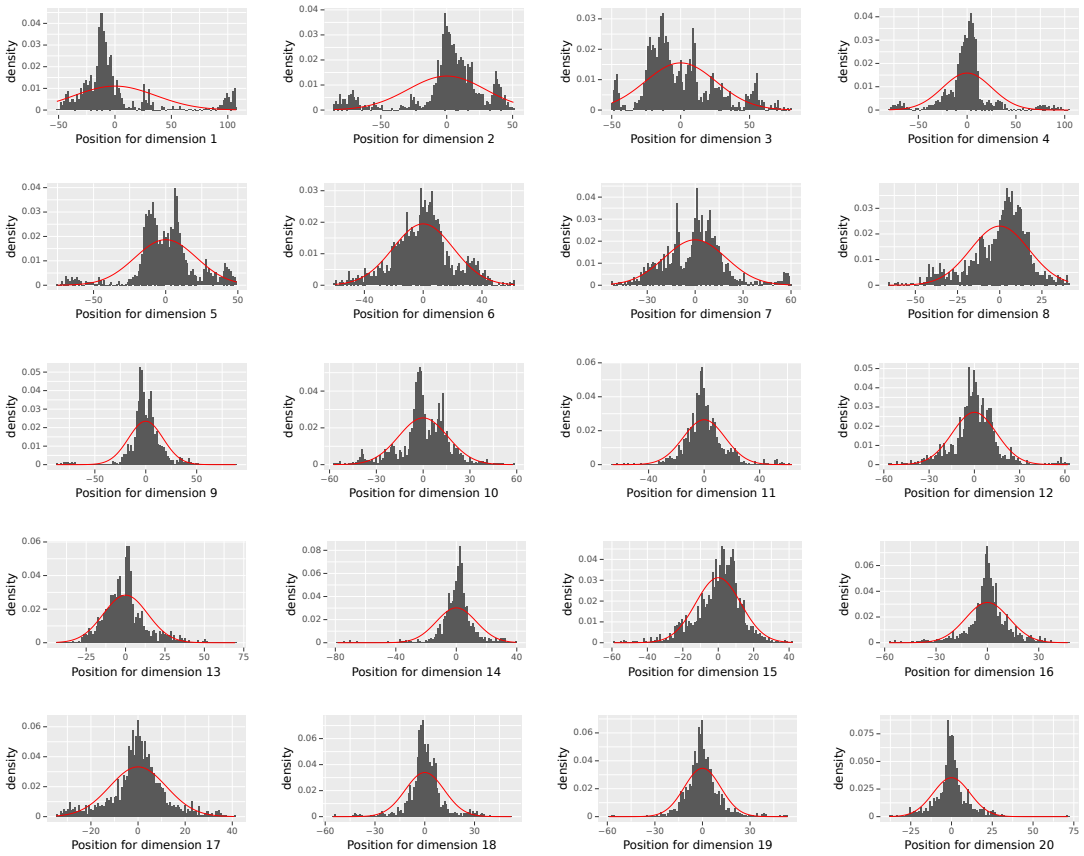


Figure B.4 – Histogram of the distribution of the coordinates per dimension in Atlas Guyane for dimensions varying from 1 to 20.

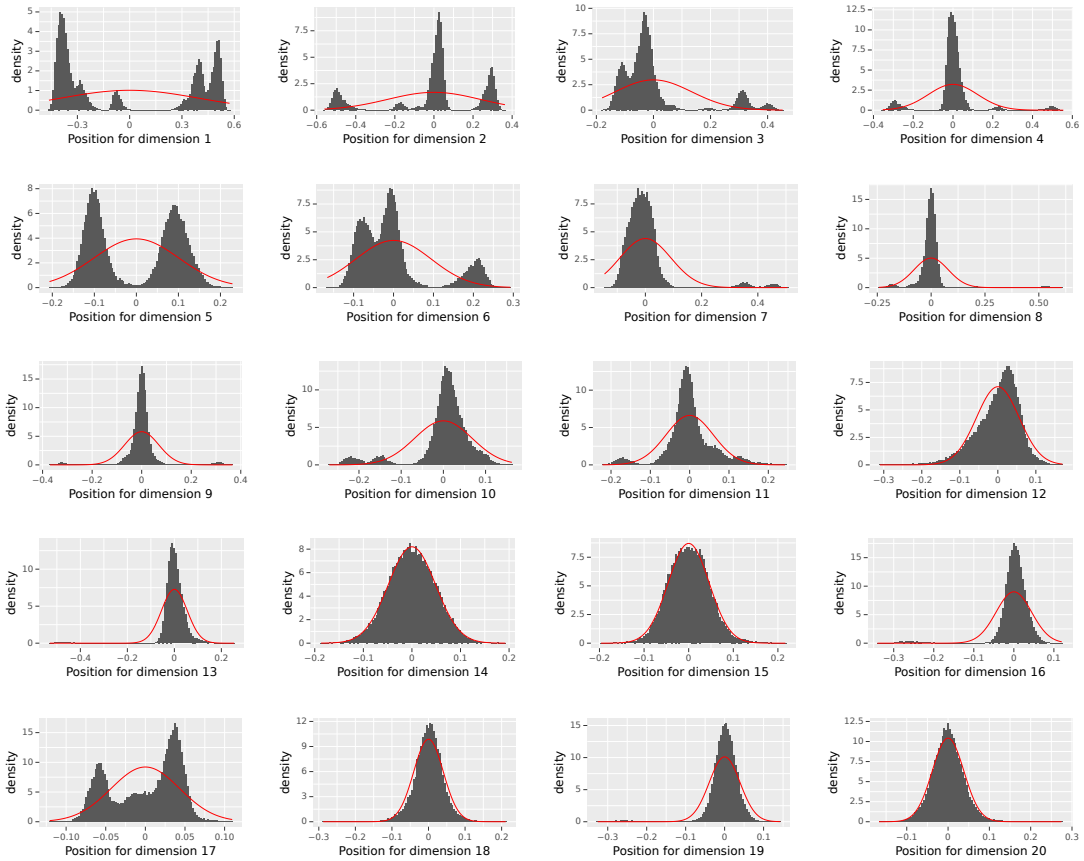


Figure B.5 – Histogram of the distribution of the coordinates per dimension in LR for dimensions varying from 1 to 20.

Appendix C

Extra results from chapter 4

C.1 Results using absolute distance

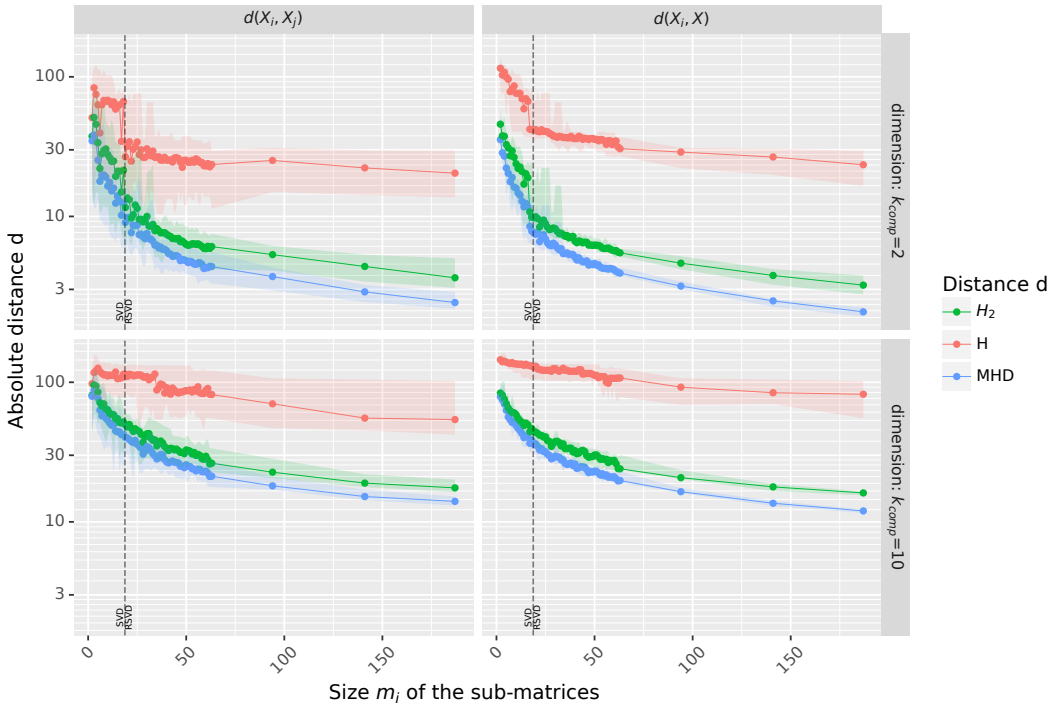


Figure C.1 – Evaluation of the H, MHD and H_2 distances of 8 submatrices between each other (left) and between the submatrices and the original Atlas Guyane sample (right). Median value is displayed in solid line.

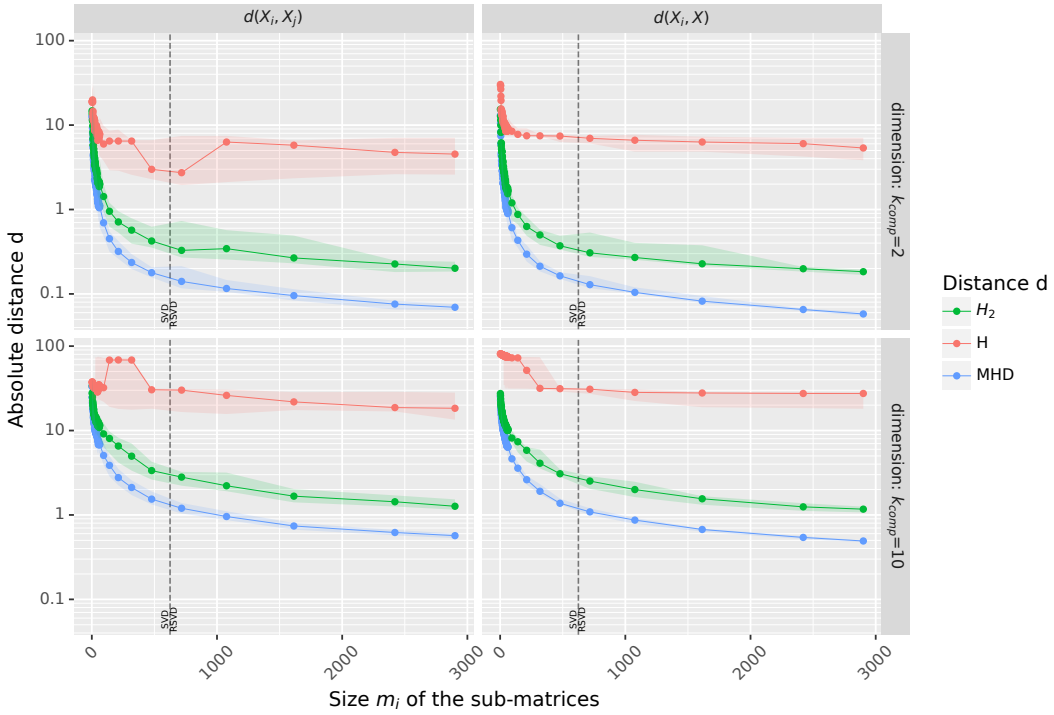


Figure C.2 – Evaluation of the H, MHD and H_2 distances of 8 submatrices between each other (left) and between the submatrices and the original 10V-RbcL sample (right). Median value is displayed in solid line.

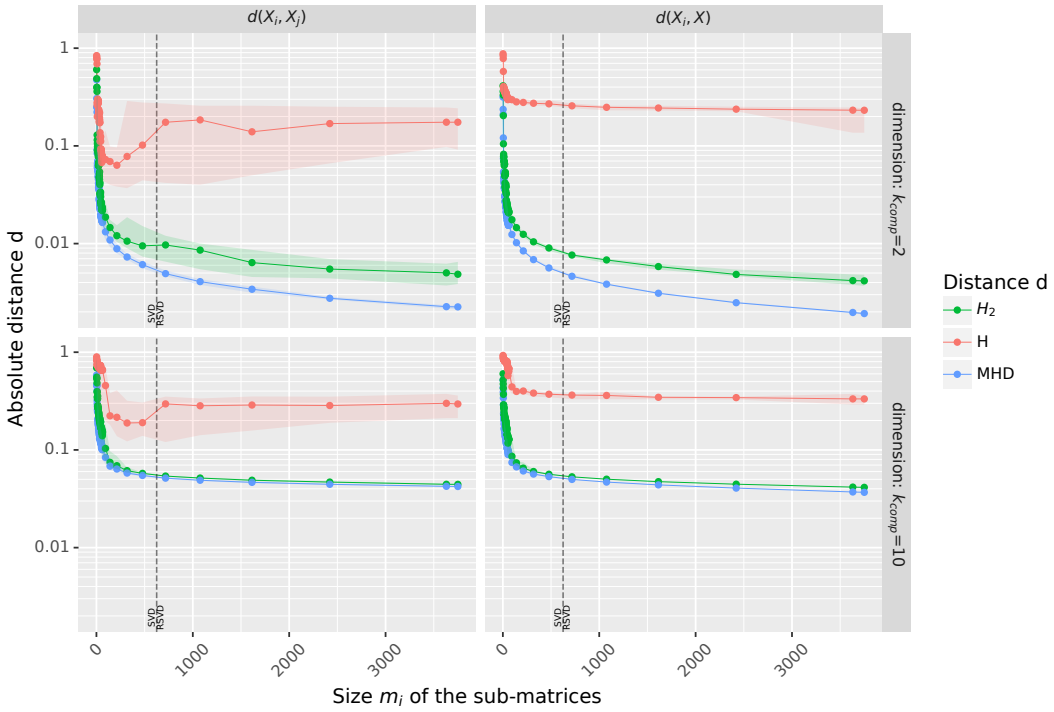


Figure C.3 – Evaluation of the H, MHD and H_2 distances of 8 submatrices between each other (left) and between the submatrices and the original Long Reads A sample (right). Median value is displayed in solid line.

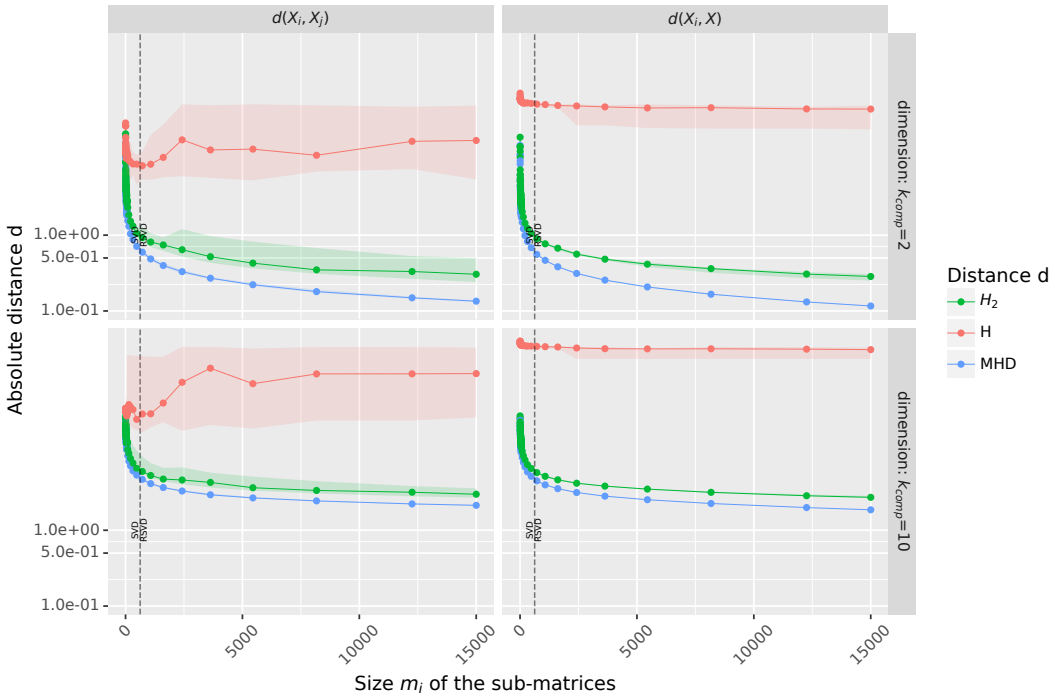


Figure C.4 – Evaluation of the H, MHD and H_2 distances of 8 submatrices between each other (left) and between the submatrices and the original Lref sample (right). Median value is displayed in solid line.

Appendix D

Extra results from chapter 5

D.1 Extra L_i/L_j distances

Sample	Procrustes	Flip	Fold	Procrustes Fold
L1	9.287e-03	5.653e-02	1.627e-02	4.529e-03
L3	9.116e-03	6.924e-02	8.950e-03	4.399e-03
L4	1.333e-02	1.818e-01	1.697e-01	5.547e-03
L5	1.814e-02	2.076e-01	1.476e-01	8.404e-03
L6	2.722e-02	8.492e-02	5.508e-02	9.207e-03
L7	2.951e-02	8.136e-02	4.841e-02	9.796e-03
L8	2.016e-02	4.766e-02	1.512e-02	9.483e-03
L9	2.929e-02	1.158e-01	3.129e-02	1.346e-02
L10	2.662e-02	6.563e-02	3.023e-02	1.216e-02

Table D.1 – $H_{2,r}(L_2, L_j)_{j\neq 2}$ for the different methods of aligning.

Sample	Procrustes	Flip	Fold	Procrustes Fold
L1	6.931e-03	2.571e-02	1.914e-02	4.502e-03
L2	9.116e-03	6.924e-02	8.950e-03	4.399e-03
L4	8.239e-03	1.513e-01	1.424e-01	4.579e-03
L5	1.233e-02	1.261e-01	1.248e-01	7.380e-03
L6	1.401e-02	6.255e-02	4.901e-02	7.007e-03
L7	1.411e-02	5.219e-02	4.352e-02	7.261e-03
L8	1.089e-02	8.937e-02	1.745e-02	6.743e-03
L9	1.520e-02	6.471e-02	2.433e-02	8.594e-03
L10	1.441e-02	3.585e-02	3.077e-02	8.440e-03

Table D.2 – $H_{2,r}(L_3, L_j)_{j\neq 3}$ for the different methods of aligning.

Sample	Procrustes	Flip	Fold	Procrustes Fold
L1	1.102e-02	9.280e-02	7.840e-02	6.226e-03
L2	1.333e-02	1.818e-01	1.697e-01	5.547e-03
L3	8.239e-03	1.513e-01	1.424e-01	4.579e-03
L5	6.580e-03	1.509e-02	7.278e-03	4.900e-03
L6	8.475e-03	1.292e-01	6.501e-02	4.345e-03
L7	9.513e-03	1.140e-01	1.044e-01	4.643e-03
L8	1.021e-02	1.509e-01	1.382e-01	5.279e-03
L9	1.737e-02	2.074e-01	1.274e-01	7.323e-03
L10	1.443e-02	4.898e-02	2.475e-02	6.057e-03

Table D.3 – $H_{2,r}(L_4, L_j)_{j\neq 4}$ for the different methods of aligning.

Sample	Procrustes	Flip	Fold	Procrustes Fold
L1	1.453e-02	5.788e-02	5.333e-02	8.677e-03
L2	1.814e-02	2.076e-01	1.476e-01	8.404e-03
L3	1.233e-02	1.261e-01	1.248e-01	7.380e-03
L4	6.580e-03	1.509e-02	7.278e-03	4.900e-03
L6	8.782e-03	1.157e-01	7.687e-02	5.271e-03
L7	1.049e-02	8.895e-02	8.754e-02	5.395e-03
L8	1.043e-02	1.945e-01	1.236e-01	5.950e-03
L9	1.725e-02	1.920e-01	1.232e-01	8.764e-03
L10	1.441e-02	3.539e-02	2.451e-02	7.559e-03

Table D.4 – $H_{2,r}(L_5, L_j)_{j\neq 5}$ for the different methods of aligning.

Sample	Procrustes	Flip	Fold	Procrustes Fold
L1	1.614e-02	6.265e-02	5.210e-02	9.408e-03
L2	2.722e-02	8.492e-02	5.508e-02	9.207e-03
L3	1.401e-02	6.255e-02	4.901e-02	7.007e-03
L4	8.475e-03	1.292e-01	6.501e-02	4.345e-03
L5	8.782e-03	1.157e-01	7.687e-02	5.271e-03
L7	5.210e-03	1.076e-02	8.294e-03	3.465e-03
L8	5.817e-03	7.354e-02	1.629e-02	3.252e-03
L9	8.996e-03	5.270e-02	3.293e-02	4.787e-03
L10	7.244e-03	2.962e-02	1.821e-02	4.333e-03

Table D.5 – $H_{2,r}(L_6, L_j)_{j\neq 6}$ for the different methods of aligning.

Sample	Procrustes	Flip	Fold	Procrustes Fold
L1	1.720e-02	5.556e-02	4.610e-02	1.007e-02
L2	2.951e-02	8.136e-02	4.841e-02	9.796e-03
L3	1.411e-02	5.219e-02	4.352e-02	7.261e-03
L4	9.513e-03	1.140e-01	1.044e-01	4.643e-03
L5	1.049e-02	8.895e-02	8.754e-02	5.395e-03
L6	5.210e-03	1.076e-02	8.294e-03	3.465e-03
L8	5.516e-03	1.250e-01	2.262e-02	2.935e-03
L9	8.548e-03	6.196e-02	1.869e-02	4.838e-03
L10	5.925e-03	2.970e-02	1.783e-02	3.634e-03

Table D.6 – $H_{2,r}(L_7, L_j)_{j\neq 7}$ for the different methods of aligning.

Sample	Procrustes	Flip	Fold	Procrustes Fold
L1	1.467e-02	1.091e-01	2.964e-02	1.026e-02
L2	2.016e-02	4.766e-02	1.512e-02	9.483e-03
L3	1.089e-02	8.937e-02	1.745e-02	6.743e-03
L4	1.021e-02	1.509e-01	1.382e-01	5.279e-03
L5	1.043e-02	1.945e-01	1.236e-01	5.950e-03
L6	5.817e-03	7.354e-02	1.629e-02	3.252e-03
L7	5.516e-03	1.250e-01	2.262e-02	2.935e-03
L9	5.096e-03	9.636e-02	2.253e-02	3.395e-03
L10	5.115e-03	1.016e-01	2.960e-02	3.181e-03

Table D.7 – $H_{2,r}(L_8, L_j)_{j\neq 8}$ for the different methods of aligning.

Sample	Procrustes	Flip	Fold	Procrustes Fold
L1	1.918e-02	6.930e-02	5.454e-02	1.240e-02
L2	2.929e-02	1.158e-01	3.129e-02	1.346e-02
L3	1.520e-02	6.471e-02	2.433e-02	8.594e-03
L4	1.737e-02	2.074e-01	1.274e-01	7.323e-03
L5	1.725e-02	1.920e-01	1.232e-01	8.764e-03
L6	8.996e-03	5.270e-02	3.293e-02	4.787e-03
L7	8.548e-03	6.196e-02	1.869e-02	4.838e-03
L8	5.096e-03	9.636e-02	2.253e-02	3.395e-03
L10	6.841e-03	6.175e-02	2.465e-02	4.300e-03

Table D.8 – $H_{2,r}(L_9, L_j)_{j\neq 9}$ for the different methods of aligning.

Sample	Procrustes	Flip	Fold	Procrustes Fold
L1	1.957e-02	3.412e-02	2.435e-02	1.248e-02
L2	2.662e-02	6.563e-02	3.023e-02	1.216e-02
L3	1.441e-02	3.585e-02	3.077e-02	8.440e-03
L4	1.443e-02	4.898e-02	2.475e-02	6.057e-03
L5	1.441e-02	3.539e-02	2.451e-02	7.559e-03
L6	7.244e-03	2.962e-02	1.821e-02	4.333e-03
L7	5.925e-03	2.970e-02	1.783e-02	3.634e-03
L8	5.115e-03	1.016e-01	2.960e-02	3.181e-03
L9	6.841e-03	6.175e-02	2.465e-02	4.300e-03

Table D.9 – $H_{2,r}(L_{10}, L_j)_{j\neq 10}$ for the different methods of aligning.