Traitement et manipulation de champs de vecteurs tangents
Stage de L3

Yoann Coudert–Osmont
yoann.coudert-osmont@ens-lyon.fr

Encadré par
David Coeurjolly, LIRIS
Nicolas Bonneel, LIRIS

11 juin - 20 juillet 2018

Introduction

Les champs de vecteurs tangents sont simplement des champs de vecteurs qui sont tangents à une surface donnée. Ces derniers sont utilisés pour représenter des informations directionnelles. Cela peut permettre ensuite de placer correctement des textures ou encore d'ajouter des géométries sur une surface telles que des poils (FIGURE 1) ou des cheveux.
Durant ce stage j'ai appliqué du transport optimal sur des champs de vecteurs afin de pouvoir obtenir des interpolations entre différents champs sur une même surface ou encore obtenir des barycentres entre plusieurs champs (l'interpolation ne se faisant qu'entre deux champs). La difficulté réside dans le fait que l'on ne transporte pas des masses réelles mais des vecteurs. Il faut donc essayer de donner du sens à un tel transport tout en essayant d'obtenir des résultats jolis et naturels qui puissent être intéressant pour des artistes.

FIGURE 1 – ©Pixar/Disney

Le livre de Gabriel Peyré et Marco Cuturi explique le problème du transport optimal et donne un algorithme itératif pour résoudre ce problème dans le cas discret [4]. L'algorithme de Sinkhorn (dont une version appelée *Log-domain* permettant une meilleure stabilité dans les calculs) est donnée. C'est cet algorithme ainsi que des variantes que j'ai utilisé. Il y a par exemple une variante qui utilise des filtres de convolution [5] qui converge plus rapidement. Cette version est notamment utile pour les images. Mais la version qui nous a le plus intéressé durant ce stage est la version dite *quantum* qui permet de faire du transport optimal avec des mesures de matrices symétriques positives et qui a aussi la particularité de fonctionner entre des mesures de masses différentes [3]. Les problèmes de transport optimal où les distributions de probabilité considérées sont non normalisées sont qualifiés d'*unbalanced*.

Avant mon stage, David a mis au point un programme permettant de visualiser et d'agir assez facilement avec des champs de vecteurs tangents. Il avait notamment implémenté un algorithme permettant de générer des champs de vecteurs tangents les plus lisses possible sur des surfaces 3D [2]. Cet outil s'est avéré utile pour ce stage. De manière plus générale, les notes de cours de Fernando de Goes, Mathieu Desbrun et Yiying Tong donnent des moyens de discrétiser les opérations classiques sur les champs de vecteurs tangents [1].

Table des matières

1 Qu'est ce que le transport optimal dans des espaces discrets ?

1.1 Le problème

On se donne un espace $\mathcal{X} = \{x_1, ..., x_n\}$ (et un espace $\mathcal{Y}$) fini de cardinal n (respectivement m) et une mesure de probabilité μ (resp. ν) sur cet espace. On a :

$$\mu = \sum_{i=1}^{n} \mu_i \delta_{x_i} \qquad \sum_{i=1}^{n} \mu_i = 1 \qquad \forall i \in \{1, ..., n\}\, \mu_i \in \mathbb{R}_+$$

Où δ_{x_i} est la fonction sur $\mathcal{X}$ qui vaut 1 en x_i et 0 partout ailleurs.
On se donne ensuite une matrice de coûts $C \in R_+^{n \times m}$ où $C_{i,j}$ est le coût pour déplacer une masse de x_i à y_j (cette valeur peur être définie par une distance). On peut voir nos deux distributions μ et ν comme des répartitions de masses sur leurs espaces respectifs. Le but du transport optimal est de déplacer toute la masse suivant la distribution μ vers la distribution ν en minimisant les coûts de déplacement totaux. Le problème est alors le programme linéaire suivant :

$$W(\mu, \nu) = \inf \langle T, C \rangle = \sum_{i=1}^{n} \sum_{j=1}^{m} T_{i,j} C_{i,j}$$

$$\text{t.q.} \begin{cases} \forall i \in \{1, ..., n\} & \sum_{j=1}^{m} T_{i,j} = \mu_i \\ \forall j \in \{1, ..., m\} & \sum_{i=1}^{n} T_{i,j} = \nu_j \\ \forall i \in \{1, ..., n\}\, \forall j \in \{1, ..., m\} & T_{i,j} \geqslant 0 \end{cases} \tag{1}$$

T est la matrice qui indique comment déplacer les masses des points x_i vers les points y_j en minimisant les coûts de déplacement. La matrice T est appelée plan de transport.

W définie alors une distance entre les mesures de probabilité sur $\mathcal{X}$ et celles sur $\mathcal{Y}$ [4]. Cette distance est la distance de Wasserstein qui a pour valeur, la valeur optimal du programme linéaire. En notant T^T, la transposée de T et $\mathbb{1}_n$, le vecteur colonne $(1, ..., 1)^\mathsf{T} \in \mathbb{R}^n$, les conditions de (1) peuvent de réécrire :

$$T\mathbb{1}_m = \mu \quad T^\mathsf{T}\mathbb{1}_m = \nu \quad T \geqslant 0$$

Finalement l'intérêt de la distance de Wasserstein sera de nous permettre de définir des barycentres entre des mesures de probabilités.

1.2 Résolution

L'algorithme que j'ai utilisé durant ce stage afin de résoudre le programme linéaire précédent est l'algorithme de Sinkhorn. J'ai implémenté la version *log-domain* (Algorithm 1) qui permet une meilleur stabilité. Cet algorithme retourne un plan de transport proche de la solution optimal en utilisant une régularisation entropique. Il converge au fur et à mesure des itérations vers :

$$T_\varepsilon = \text{argmin}_T \langle C, T \rangle - \varepsilon\, H(T)$$

Où H est la fonction d'entropie suivante :

$$H(T) = -\sum_{i,j} T_{i,j}(\log T_{i,j} - 1)$$

On remarque que T_ϵ converge bien vers une solution optimale du problème initial quand ϵ tend vers 0.

Les deux autres variantes testées qui se trouvent des les papiers [5] et [3] sont des algorithmes assez similaires que je ne vais pas réécrire. Des algorithmes pour trouver des barycentres selon la distance de Wasserstein sont aussi donnés dans ces papiers. Étant données des mesures (μ_i) munies de coefficients réels (α_i), le barycentre est une mesure ν^* qui minimise la valeur suivante :

$$\nu^* = \arg\min_{\nu} \sum_i \alpha_i W(\mu_i, \nu)$$

Dans l'algorithme Sinkhorn ci-dessous, on utilise la fonction min_ϵ qui est définie par :

$$\min_{\varepsilon}(a_i) = m - \varepsilon \log\left(\sum_i \exp\left(\frac{m - a_i}{\varepsilon}\right)\right) \qquad \text{avec} \qquad m = \min_i a_i$$

.

Algorithm 1 Log-domain Sinkhorn (mu, nu, C, ε, $nSteps$)

$f,\, g \leftarrow (0)_{1 \leqslant i \leqslant n},\, (1)_{1 \leqslant j \leqslant m}$
for $s = 1$ ... $nSteps$ **do**
  $\forall i,\ f_i \leftarrow f_i + \min_\varepsilon \left(C_{i,j} - f_i - g_j\right)_j + \varepsilon \log \mu_i$
  $\forall j,\ g_j \leftarrow g_j + \min_\varepsilon \left(C_{i,j} - f_i - g_j\right)_i + \varepsilon \log \nu_j$
end for
return $\left(\exp\left(\left(f_i + g_j - C_{i,j}\right) / \varepsilon\right)\right)_{i,j}$

1.3 Interpolation

On considère deux mesure μ et ν sur un même espace $\mathcal{X}$. Un interpolation entre μ et ν est une fonction f de $[0, 1]$ dans l'espace des mesures sur $\mathcal{X}$ telle que $f(0) = \mu$ et $f(1) = \nu$. Un interpolation simple est l'interpolation linéaire qui est définie par $f(t) = (1 - t)\mu + t\nu$. Cette interpolation est généralement peu naturelle et n'est pas très *jolie* à observer puisqu'elle fait disparaître la première mesure pour faire apparaître la seconde comme par magie. On aimerait bien que la masse de la première mesure se déplace vers la masse de la seconde mesure (voir FIGURE2). Notre idée est donc d'utiliser le transport optimal.

On pourrait calculer le barycentre de $(\mu, 1 - t)$ et (ν, t) au temps t selon la distance de Wasserstein mais cette méthode est très coûteuse en calculs. Le paragraphe 2.2 de [3] donne une autre manière assez intuitive d'obtenir une interpolation. En effet, lorsqu'on obtient un plan de transport T entre μ et ν, on sait exactement quel quantité de masse se déplace de où à où puisque $T_{i,j}$ est la quantité de masse qui se déplace de x_i à x_j. Si on dispose d'une distance $d_\mathcal{X}$ sur $\mathcal{X}$, alors on peut définir une interpolation entre deux points de $\mathcal{X}$ par :

$$x_{i,j}^t = \text{argmin}_x \ (1 - t)\, d_\mathcal{X}(x, x_i) + t\, d_\mathcal{X}(x, x_j) \tag{2}$$

En notant $\mu^t = f(t)$ notre interpolation au temps t, on souhaite avoir $\mu^0 = \mu$ et $\mu^1 = \nu$. Pour cela on pose :

$$T_{i,j}^t = \left((1 - t)\, \mu_i \left(\sum_k T_{i,k}\right)^{-1} + t\, \nu_j \left(\sum_k T_{k,j}\right)^{-1}\right) T_{i,j} \tag{3}$$

Notre interpolation est alors définie par l'équation suivante :

$$\mu^t = \sum_{i,j} T^t_{i,j} \delta_{x^t_{i,j}} \tag{4}$$

1.4 Exemples

Avant de m'attaquer aux champs de vecteurs tangents j'ai commencé par tester le transport optimal pour interpoler des mesures à valeurs réelles.

Dans un premier temps, j'ai essayé d'appliquer cette interpolation sur un espace discret $\mathcal{X}$ de dimension 1. Le plan de transport est calculé via Algorithm 1.

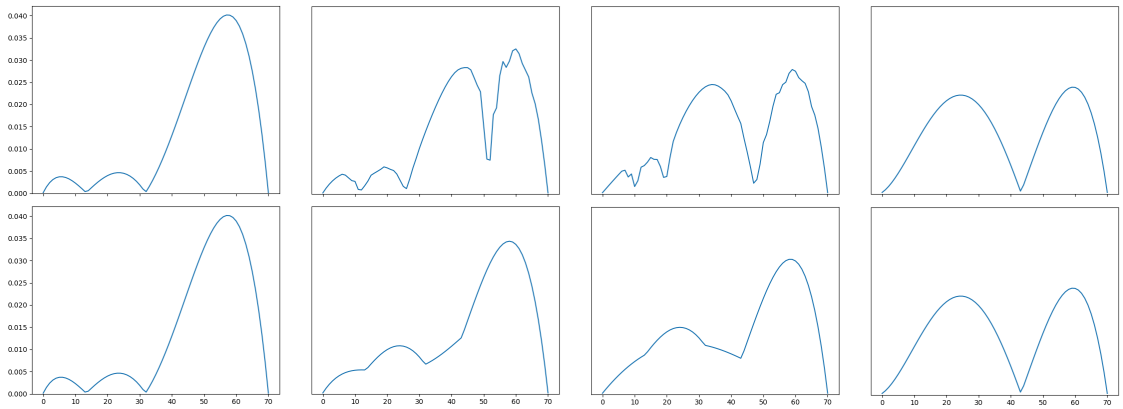

FIGURE 2 – Comparaison entre l'interpolation obtenue via le transport optimal (4) sur la première ligne et une interpolation linéaire $\mu^t_i = (1-t)\,\mu_i + t\,\nu_i$ sur la seconde ligne. Les images de gauche à droite correspondent aux temps $t = 0,\ 0.33,\ 0.66,\ 1$.

.

Sur les graphiques de la FIGURE2, la valeur à l'abscisse i est μ^t_i où t vaut 0 à la première colonne, 0.33 à la deuxième colonne, ... Ainsi la première colonne correspond à la mesure μ et la dernière à la mesure ν. On constate que le résultat obtenu via le transport optimal sur la première ligne est assez satisfaisant. On observe des masses qui se déplacent. La grosse bosse de la première mesure (μ) se scinde en deux et la partie de gauche vient se placer à l'emplacement de la première bosse de la seconde mesure (ν). Avec une interpolation linéaire (sur la seconde ligne), les changements se font tous localement et l'interpolation est alors moins naturelle.
Dans cet exemple j'ai pris $n = m = 71$, $\epsilon = 0.1$ et $nSteps = 3500$ pour un temps de calcul d'environs 8s. Le plan de transport obtenu se trouve à la FIGURE 3. La couleur du point à la position (i, j) dépend de la valeur de $T_{i,j}$ (la quantité de masse qui se déplace de la position i à la position j). Une valeur faible de $T_{i,j}$ a une couleur bleue tandis qu'une valeur élevée a une couleur rouge. On observe que ce plan (qui ressemble à une courbe un peu épaissie) semble assez précis sachant que le plan de transport optimal est bien une courbe d'expression $f(x) = N^{-1}(M(x))$ où M et N sont les primitives de μ et ν qui s'annulent en 0. En effet, $M(x)$ est la masse de μ qui se trouve entre 0 et x. Or cette masse permet de remplir ν de 0 à $N^{-1}(M(x))$. D'où le fait que l'on déplace la masse situé à la position x vers la position $f(x)$.

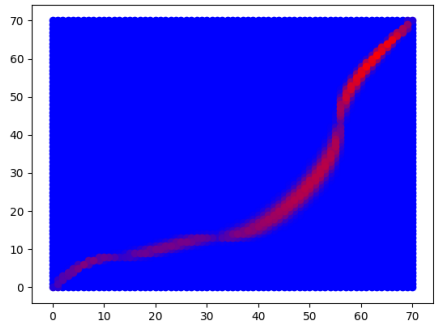

FIGURE 3 – Plan de transport associé à FIGURE 2.

.

Par la suite j'ai essayé d'interpoler des images 2D. Dans un premier temps j'ai interpolé des desseins au trait (FIGURE 4). Pour cela, μ et ν sont des mesures uniformes sur $\mathcal{X}$ et $\mathcal{Y}$ qui sont les ensembles des pixels noirs des deux images (à noté que $\mathcal{X} \neq \mathcal{Y}$). La matrice des coûts C est bien définie grâce à la distance euclidienne sur $\mathbb{R}^2$ puisque $\mathcal{X}$ et $\mathcal{Y}$ sont inclus dans $\mathbb{R}^2$. Une fois le plan de transport calculé, on calcule les $x_{i,j}^t$ dans le plan entier et non dans $\mathcal{X}$ ou $\mathcal{Y}$. On obtient l'interpolation suivante. On voit très clairement l'intérêt du transport optimal ici puisqu'une interpolation classique aurait des images intermédiaires où l'on verrait à la fois la vache disparaissant et la poule qui serait en train d'apparaître.

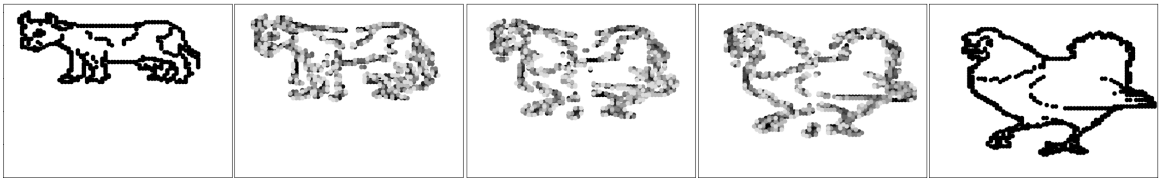

FIGURE 4 – Interpolation entre une image de vache et une autre de poule

.

Par la suite ce sont des images en couleur (RGB) de petites tailles que j'ai interpolé (FIGURE 5). L'interpolation se fait donc sur chaque couleur rouge, verte et bleu. J'utilise cette fois la version de Sinkhorn avec filtre de convolution gaussien. Comme j'utilise de petites images j'ai $\mathcal{X} = \mathcal{Y}$ qui est la grille de pixels 20x20. On a donc $n = 400$. Pour changer encore un peu la méthode, sur cette interpolation, se sont des barycentres qui sont calculés et non l'interpolation décrite via l'équation (4). Pour chaque interpolation on a $\epsilon = 0.3$ et $nSteps = 60$. La convergence est bien plus rapide lorsqu'on utilises des filtres de convolution. Le temps de calcul est alors de 4s par barycentres.

Finalement je vais finir cette sous-partie en donnant un exemple d'utilisation de l'algorithme de calcul de barycentre sur 4 petites images (FIGURE6). Après avoir dessiné une étoile $\mu^{(1)}$ (en haut à gauche), un cercle $\mu^{(2)}$ (en haut à droite), un "donuts" $\mu^{(3)}$ (en bas à gauche) et deux petits carrés non connexes $\mu^{(4)}$ (en bas à droite), j'affiche des barycentres tel que le barycentre à la position $(x, y) \in [0, 1]^2$ est le barycentre de $(\mu^{(1)}, (1-x)y)$, $(\mu^{(2)}, xy)$, $(\mu^{(3)}, (1-x)(1-y))$ et $(\mu^{(4)}, x(1-y))$. C'est encore la version de Sinkhorn avec convolution que j'utilise dans ce cas.

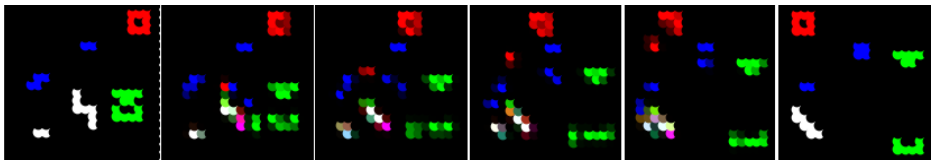

FIGURE 5 – Interpolation entre deux images RGB via filtre de convolution gaussien

.

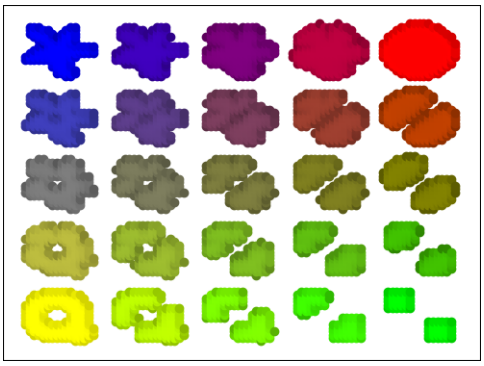

FIGURE 6 – Tableau de barycentres entre les 4 images qui sont dans les coins

.

2 Transport optimal sur des champs de vecteurs tangents

Dans les exemples précédents, ce sont des valeurs réelles positives que l'on transporte. Maintenant on aimerait bien transporter des vecteurs afin de pouvoir interpoler des champs sur des surfaces. Nous avons utilisé des surfaces à faces triangulaires. Ainsi note espace $\mathcal{X}$ est un ensemble de faces, et nous voulons interpoler des champs de vecteurs qui sont des fonctions qui à chaque face associent un vecteur tangent à la face. Pour faciliter le travail David utilise une interface codée en C++ qui s'appelle polyscope. Elle permet notamment de visualiser des champs de vecteurs tangents (FIGURE 7). Une idée naturelle est de considérer la masse comme étant la norme des vecteurs. Ensuite vient les coûts de déplacement. On peut se demander si l'on doit prendre en compte les rotations dans le coût. J'ai pu expérimenté parallèlement deux méthodes pour interpoler des champs. La première méthode m'a été proposé par David et Nicolas et la seconde est de ma propre initiative. Je vais expliquer ces méthodes par la suite. J'aurai aussi pu me lancer dans le calcul de barycentres avec plus de deux champs car j'avais déjà implémenté tous les outils nécessaires pour, mais j'ai préféré me restreindre à l'interpolation de deux champs afin obtenir des résultats satisfaisants.

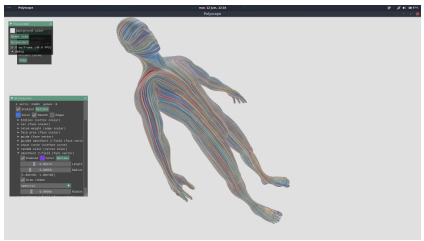

FIGURE 7 – L'interface polyscope

2.1 Interpolation classique

On peut dans un premier temps regarder ce que donne une interpolation linéaire définie par :

$$\mu^t = (1-t)\mu + t\nu$$

J'ai fait mes testes sur un cas assez complexe pour bien pouvoir comparer l'interpolation linéaire et les deux nouvelles idées que nous avons eu. Je réalise donc une interpolation entre un champ qui contient une singularité d'où partent les lignes de champ (source) et un autre champ avec une singularité où arrivent les lignes de champ (puits) ; Le tout sur une surface qui est un tore. Le résultat obtenu avec l'interpolation classique se trouve à la FIGURE8. On voit que pour $t \in]0, 1[$, la source de μ et le puits de ν sont tous les deux présents. La source disparaissant petit à petit au cours du temps et le puits s'accentuant au cours du temps. C'est un résultat convenable mais l'apparition brutale du puits au tout début de l'interpolation n'est pas très naturelle.
Ce qui pourrait être intéressant et ce que l'on aimerait réussir à obtenir serait que la source se déplace vers la position du puits tout en se transformant en puits. Cela pourrait se faire en faisant varier l'indice de la singularité de 1 vers -1. 1 étant l'indice d'une source et -1 celui d'un puits.

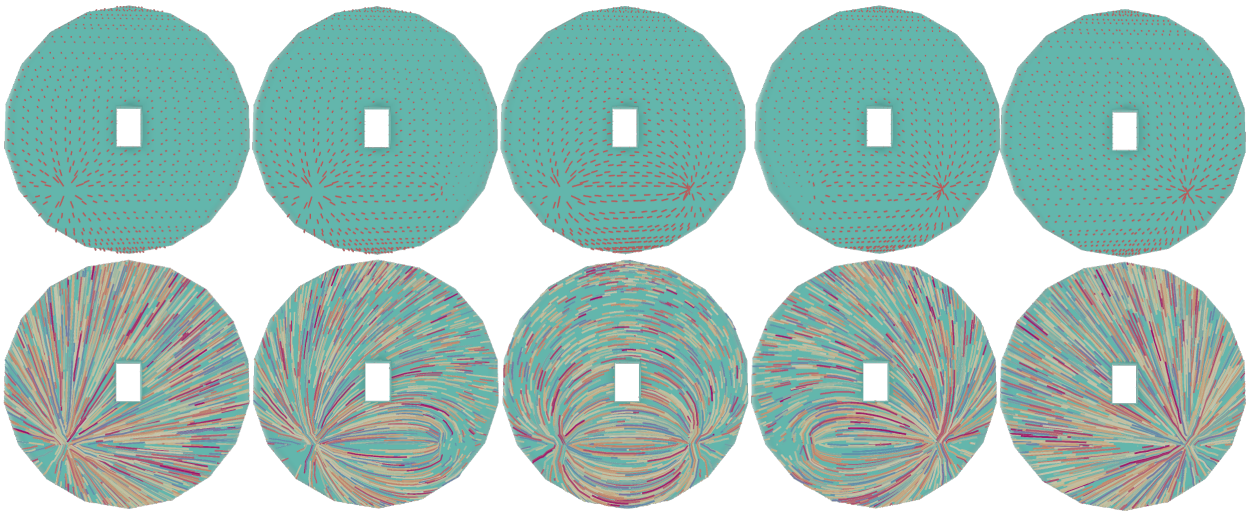

FIGURE 8 – Interpolation linéaire
Les colonnes correspondent aux temps $t = 0, 0.25, 0.5, 0.75$ et 1. Les deux lignes sont des représentations différentes des champs. La première ligne est la représentation vectorielle. On part d'un champ contenant une source en bas à gauche et fini à un champ contenant un puits en bas à droite. La deuxième ligne est la représentation par lignes de champ. Les lignes suivent la direction des vecteurs

Utiliser du transport optimal semble pouvoir satisfaire notre objectif. On s'attend au fait qu'en se déplaçant, les vecteurs emmènent en quelque sorte la singularité avec eux.

2.2 Distances Géodésiques

Bien que dans un premier temps mes tests aient été effectués avec des distances euclidiennes dans $\mathbb{R}^3$ entre les faces, j'ai dû à un moment passer à une matrice de coûts qui utilise des distances géodésiques. La distance géodésique entre deux faces est la longueur du plus court chemin entre les deux faces le long de la surface considéré. On nomme d'ailleurs lignes géodésiques les plus court chemins sur le long de la surface.
Je ne calcule pas les distances géodésiques exactes car cela pourrait demander beaucoup de calculs mais je calcule des valeurs approchées. De plus nous avons besoin de calculer des barycentres

dans l'espace discret des faces $\mathcal{X}$ et pas dans l'espace continu de la surface pour l'équation 2. Donc les lignes géodésiques que je calcule doivent passer par les centres des faces. Pour cela, je crée un graphe dont les sommets sont les centres des faces de la surface que je considère (les isobarycentre des sommets des faces). Il y a alors une arrête entre deux sommets du graphe si et seulement si les deux triangles des surfaces associées à ces deux sommets ont un sommet ou un côté en commun. Le poids associé à une arrête qui provient d'un côté en commun est la véritable distance géodésique ($\min_{I \in e} \|B - I\| + \|I - A\|$). En revanche dans le cas où l'arrête provient d'un sommet en commun J, le poids est $\|C - J\| + \|J - A\|$ (voir FIGURE 9). Les normes ici sont prises dans l'espace $\mathbb{R}^3$.
Une fois ce graphe obtenu j'applique l'algorithme de Dijkstra à partir de toutes les faces pour obtenir la matrice des distances. On note F le nombre de faces de notre surface. Comme chaque face à environs une dizaine de voisines (3 faces adjacentes par ses côtés et souvent le nombre de faces adjacentes par un des sommets est proche de 10) alors la complexité de ce calcul est en $O(F^2 \log F)$ (d'où la préférence pour Dijkstra plutôt que Floyd Warshall qui est en $O(F^3)$).

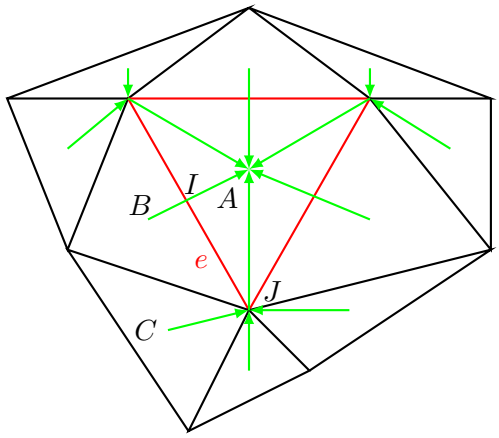

FIGURE 9 – Distances aux faces adjacentes

Résultats Sur la FIGURE10 on peut voir le résultat obtenu sur un lapin. Les bandes foncées et claires sont des lignes équidistantes par rapport à un sommet du lapin qui se trouve au niveau du dos. Même si les cercles ne sont pas parfaits les distances semblent plutôt correctes car les bandes ont toutes à peu près la même largeur.

2.3 Idée initiale : Quantum-Sinkhorn

L'idée de David et de Nicolas était d'utiliser le transport optimal avec des champs de matrices symétriques positives (que je dénoterais PSD pour *Positive SemiDefinite matrices*) pour décrire les champs de vecteurs. En effet une PSD peut être vue comme une ellipse puisque d'après le théorème spectral, les vecteurs propres des PSD forment une base orthogonale et une ellipse possède deux axes orthogonaux. On pourrait alors représenter un vecteur par une ellipse aplatie. L'algorithme quantum décrit dans l'article [3] permet de faire du transport optimal avec des champs de PSD. C'est donc l'algorithme que l'on va utiliser.

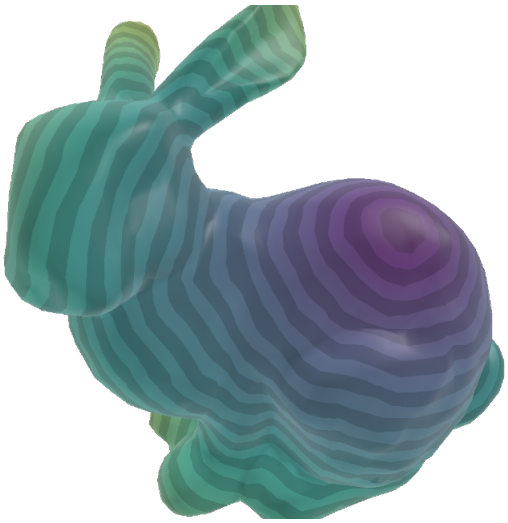

FIGURE 10 – Distances géodésique sur un lapin

Dimension des PSD Afin de limiter les temps de calcul, car l'algorithme Sinkhorn quantum sur des champs de PSD effectue beaucoup d'exponentielles et de logarithmes de matrices, nous avons décidé d'utiliser des PSD de dimension 2 (les opérations sur les matrices sont effectuées sur les valeurs propres sans changer les vecteurs propres). Je pense avoir assez optimisé le calcul des opérations sur les PSD de dimension 2 (voir la fonction `appfun` dans le code source).
Comme on se place en dimension 2, on ne va pas représenter les vecteurs dans l'espace 3D mais sur le plan tangent à la surface que l'on considère. Pour représenter un vecteur dans un plan il nous faut une base $(\vec{e_1}, \vec{e_2})$ de deux vecteurs de l'espaces de tel sorte que lorsqu'on parle du vecteur (u, v) dans ce plan on parle du vecteur $u\vec{e_1} + v\vec{e_2}$ dans l'espace. Vient alors un problème qui est de définir les bases pour représenter les vecteurs tangents sur chaque face. En effet, si on déplace un vecteur (u, v) d'une face ayant pour base $(\vec{e_1}, \vec{e_2})$ vers une face ayant pour base $(\vec{f_1}, \vec{f_2})$ on aimerait avoir les vecteurs de l'espace $u\vec{e_1}+v\vec{e_2}$ et $u\vec{f_1}+v\vec{f_2}$ qui soient aussi proches qui possible. Ainsi un vecteur qui se déplace le long d'une géodésique ne subira pas de changements brusques de directions dans l'espace. La FIGURE 11 illustre ce que serait des bases cohérentes entre deux faces voisines.

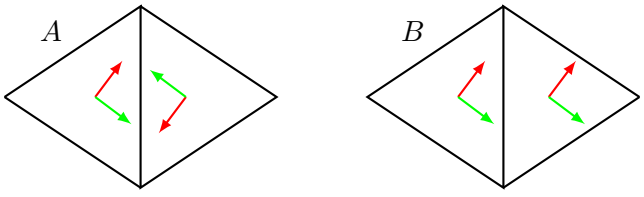

FIGURE 11 – On souhaite avoir des bases proches d'une face à une autre. On préfère le cas B au cas A.

Base des faces Pour résoudre ce problème des bases des faces, j'utilise l'algorithme qui permet d'obtenir le champ le plus lisse possible sur la surface [2] que David a implémenté. La création d'un tel champs n'étant pas le sujet de mon stage je vous invite à lire l'article [2] si vous voulez

en savoir plus sur ce qui est entendu par "champs lisse". En principe un champs lisse est un champs tel que les vecteurs qui se trouvent sur deux faces adjacentes sont proches dans l'espace.

Le champ est globalement lisse malgré quelques singularités (la FIGURE 12 en montre une sur un cube) si on ne travaille pas sur un tore (pour un tore il existe des champs qui ne contiennent pas de singularités car la caractéristique d'Euler vaut zéro). On peut donc utiliser les vecteurs de ce champ afin de définir le premier vecteur des bases de chaque face. Le second vecteur des bases s'obtient alors simplement en faisant le produit vectoriel entre le premier vecteur et la normale à la face. Une fois ces bases obtenues on utilisera des complexes pour représenter les vecteurs dans les bases des faces. Le nombre $\rho e^{i\theta}$ correspond au vecteur $(\rho\cos(\theta), \rho\sin(\theta))$ dans le plan de la face et au vecteur $\rho\cos(\theta)\vec{e_1} + \rho\sin(\theta)\vec{e_2}$ dans l'espace. Finalement la FIGURE13 montrent les base que l'on obtient sur le lapin avec cette méthode.

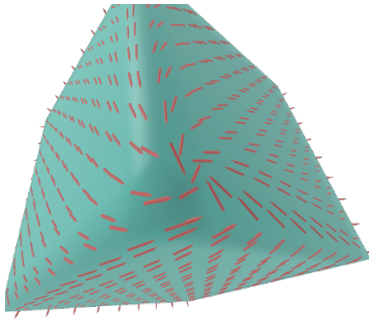

FIGURE 12 – Singularité

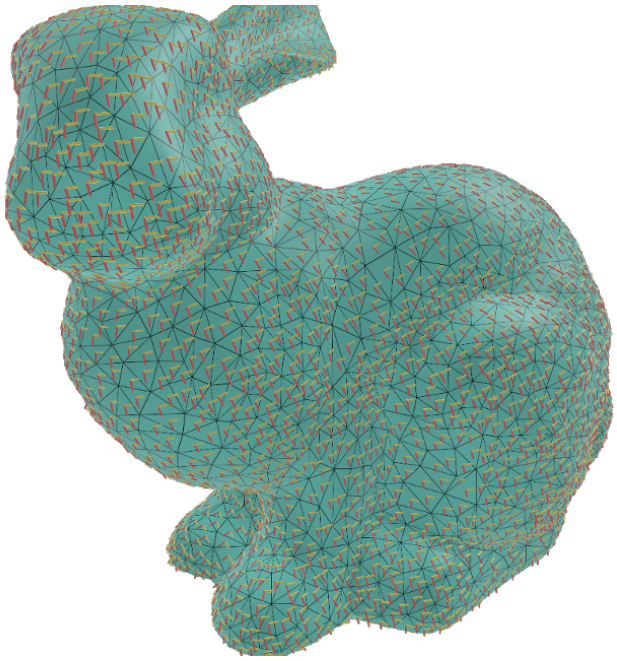

FIGURE 13 – Bases obtenues sur les faces du lapin

Conversion complexe-PSD Un autre détail à régler est le fait que les vecteurs propres des PSD sont en fait des bi-vecteurs (si u est vecteurs propres alors $-u$ l'est aussi). On ne peut donc pas faire une correspondance toute bête entre notre vecteur dans la base de la face et le premier vecteur propre de la PSD car deux vecteurs opposés correspondraient à la même PSD. Il suffit alors de diviser par deux l'argument du complexe qui représente le vecteur dans la base de la face. La valeur propre associée est la norme du vecteur. Le second vecteur propre est alors orthogonal et on lui associe la précédente valeur propre divisée par un facteur suffisamment grand pour que la PSD soit aplatie mais pas trop grand pour éviter des problèmes d'instabilité dans les calculs de l'algorithme Sinkhorn (par exemple 100 est un bon facteur) (FIGURE 14).

Pour la conversion inverse (de PSD à vecteur) il suffit de prendre le vecteur propre associé à la plus grande valeur propre et de doubler son argument. La norme sera alors la plus grande valeur propre. Comme les PSD ont tendance à devenir isotrope (les deux valeurs propres sont de plus en plus proche) durant l'interpolation, on peut ajouter un pourcentage de la seconde valeur propre à la norme afin de ne pas perdre trop de masse.
Comme dans l'équation (3) il y a des multiplications entre PSD, on peut perdre la symétrie de $T^t_{i,j}$. Mais cette perte est légère car si l'algorithme converge bien, on a : $\sum_k T_{i,k} \simeq \mu_i$ d'où le fait que $\mu_i \left(\sum_k T_{i,k}\right)^{-1}$ soit proche de l'identité et qu'ainsi $T^t_{i,j} \simeq T_{i,j}$. Il suffit alors d'appliquer la fonction $M \mapsto \frac{1}{2}(M + M^T)$ à notre matrice pour qu'elle redevienne symétrique.

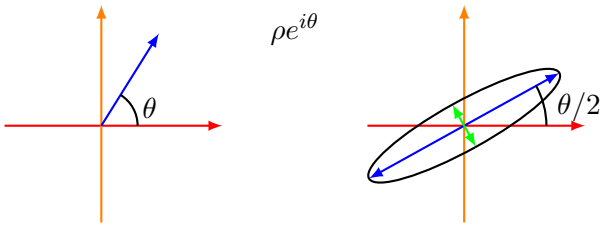

FIGURE 14 – Conversion de vecteur/complexe à PSD

Reformulation du transport optimal avec des PSD On a désormais $\mu_i \in S_+$. Où S_+ est l'ensemble des PSD. Le plan de transport T a alors des coefficients dans S_+. Les relations (1) ne sont plus vérifiées mais on introduit de nouveaux termes dans la valeur à minimiser afin de se rapprocher le plus possible de ces relations. Le plan de transport doit alors minimiser (cf. [3]) :

$$\langle T, C\rangle + \rho_1 \,\mathrm{KL}(T\mathbb{1}_m|\mu) + \rho_2 \,\mathrm{KL}(T^\mathsf{T}\mathbb{1}_n|\nu)$$

Avec :

$$\langle T, C\rangle = \sum_{i,j} \mathrm{tr}(T_{i,j}C^\mathsf{T}_{i,j}) \quad \text{et} \quad \mathrm{KL}(\alpha|\beta) = \mathrm{tr}(\alpha\log(\alpha) - \alpha\log(\beta) - \alpha + \beta)$$

La divergence de Kullback-Leibler KL permet de mesurer une distance entre deux distributions. La matrice de coût C est définie par $C_{i,j} = {d_\mathcal{X}}^\alpha(x_i, y_j)\,\mathrm{Id}_{2\times 2}$ où α est une constante strictement plus grande que 1 (j'ai choisi sa valeur à 2). Ainsi $\langle T, C\rangle = \sum_{i,j} \mathrm{tr}(T_{i,j}){d_\mathcal{X}}^\alpha(x_i, y_j)$. Comme la trace d'une PSD est la somme de ses valeurs propres, la distance de Wassertein induite par le premier terme ne prend en compte que les déplacements des normes des vecteurs. Dans notre cas $d_\mathcal{X}$ sera la distance géodésique sur la surface.
Les paramètres ρ_1 et ρ_2 permettent de gérer l'importance d'avoir $T\mathbb{1}_m$ proche de μ et $T^\mathsf{T}\mathbb{1}_n$ proche de ν. Plus leurs valeurs sont élevées, plus on se rapproche des conditions (1) mais plus le nombre d'itérations nécessaires pour que Sinkhorn converge est élevé.
Pour plus de détails je vous invite à aller voir l'article *Quantum Optimal Transport for Tensor Field Processing* [3]. La version modifiée de Sinkhorn y est donnée. Le principe est le même, on rajoute un terme d'entropie qui rend le problème strictement convexe.

Détail sur le calcul de μ^t On conserve l'équation (3), en revanche on ajoute pas $T^t_{i,j}$ à une seule face comme suggéré par (4). L'idée est alors de calculer $x^t_{i,j}$ sur l'espace continue de la géodésique. Mais en faisant cela, $x^t_{i,j}$ ne correspond plus nécessairement au barycentre d'une face.

On ajoute donc $T^t_{i,j}$ aux deux faces qui entourent ce point sur la géodésique tout en pondérant par la distance à $x^t_{i,j}$.

Ainsi en prenant l'exemple de la FIGURE 15, on ajoute $\frac{d_{\mathcal{X}}(A,x^t_{A,C})}{d_{\mathcal{X}}(A,B)} T^t_{A,C}$ à μ^t_B et $\frac{d_{\mathcal{X}}(x^t_{A,C},B)}{d_{\mathcal{X}}(A,B)} T^t_{A,C}$ à μ^t_A car A et B sont les deux faces adjacentes à $x^t_{A,C}$.

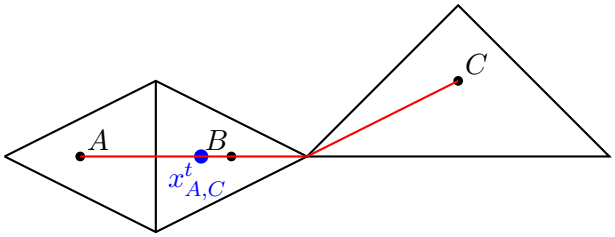

FIGURE 15 – Ajout aux faces de $T^t_{i,j}$

Résultats Le tore utilisé comme surface a un diamètre d'unité 1. Le paramètre ε est choisi en fonction de ce diamètre car sa valeur peut être vue comme une distance limite que vont parcourir la plupart des masses lors de l'interpolation. Les paramètres choisis sont $\epsilon = 0.1$, $\rho_1 = \rho_2 = 2$ et $nSteps = 80$. J'ai pu ajusté $nSteps$ de sorte qu'à la dernière itération de l'Algorithme 1, les coefficients de $T\mathbb{1}_m$ et $T^T\mathbb{1}_n$ aient des variations de l'ordre du millième de la valeur des coefficients de μ et ν. Pour que l'algorithme puisse converger en temps raisonnable ρ_1 et ρ_2 ne devaient pas dépasser des valeurs de l'ordre de 5. Finalement 2 m'a paru un bon compromis entre temps et bonne approximation de l'équation (1). De plus prendre des valeurs faibles semble réduire le soucis des vecteurs à normes très faibles que l'on peut observer sur les bords du tore quand t tend vers $\frac{1}{2}$. Je n'ai pas réussi à trouver d'explications à ce phénomène. Cela pourrait venir de problèmes d'instabilité dans les calculs mais rien n'est sûr.
Sur ma machine, le temps de calcul est de 2 minutes et 15 secondes et la surface contient 1106 faces. C'est le calcul du plan de transport qui prend beaucoup de temps. Calculer le champ interpolé à différents temps t est assez rapide.
Si vous visualisez ce rapport sur ordinateur, n'hésitez pas à zoomer pour mieux voir. On peut voir sur la FIGURE 16 que la singularité se déplace. C'est ce à quoi on aimait s'attendre. Donc le résultat est satisfaisant. Par contre le temps de calcul et le problème des vecteurs qui deviennent presque nuls sur les bords du tore laissent à désirer. Mais je rappelle que ce test est difficile car on interpole une source vers un puits. Sur les cas plus simples que j'ai pu essayé, les problèmes que l'on a ici sur les bords du tore n'interviennent pas. De plus malgré les fortes rotations que subissent les vecteurs, les PSD ne deviennent pas tout à fait isotropes (l'isotopie étant le fait que les deux valeurs propres soient proches). Ceci est un bon point car si jamais la plus petite valeur propre devenait la plus grande on aurait une rotation brutale du vecteur associé à la PSD. Pour rappelle la seconde valeur propre est la première valeur propre divisé par 100 pour le champ de départ et le champ d'arrivée. C'est pour cela que les ellipses sont tant aplaties sur la première et la dernière image de la première ligne.

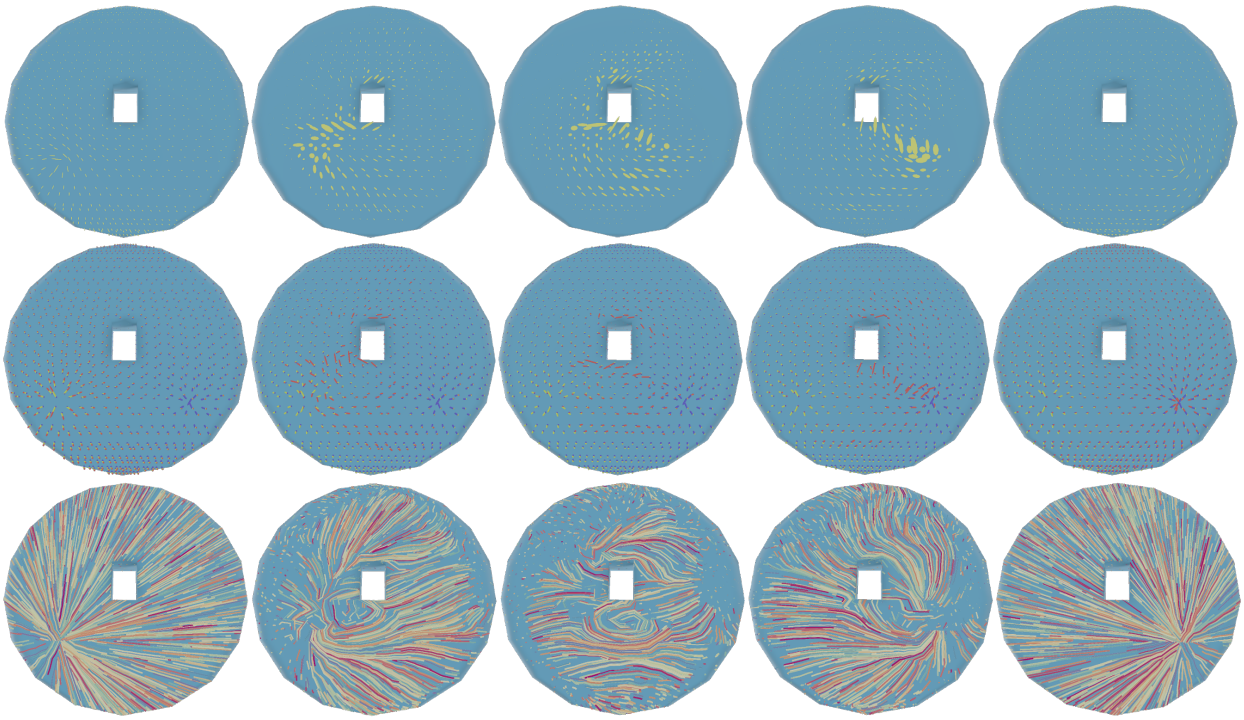

FIGURE 16 – Interpolation utilisant la version quantum de Sinkhorn
La première ligne est la représentation tensorielle, c'est à dire que pour chaque face on affiche l'ellipse associé à la PSD de la face. La seconde ligne est la représentation vectorielle. Le champs vert est le champ de départ contenant une source, le champ violet est le champ d'arrivée contenant un puits et le champs rouge est l'interpolation au temps t. Enfin la troisième ligne est la représentation par lignes de champ

2.4 Une autre idée

Les temps de calcul élevés m'ont donné envie d'essayer autre chose que du transport optimal avec des PSD. Par la suite on ne considérera plus des champs à valeur dans l'ensemble des PSD mais des champs à valeurs complexes pour représenter les champs de vecteurs. J'ai pensé à utiliser Sinkhorn coordonnées par coordonnées avec quelques subtilités comme le fait d'avoir 4 coordonnées au lieu de 2 (une coordonnée pour le sens positif et le sens négatif de chaque axes) afin de manipuler uniquement des valeurs positives. Il y a aussi l'idée de transporter le carré des coordonnées afin d'éviter les changements de normes. En effet si on ne transporte pas les carrés, les vecteurs $(\frac{1}{2}, \frac{1}{2})$ et $(1, 0)$ ont la même masse qui vaut 1 mais ont des normes différentes $\frac{1}{\sqrt{2}}$ et 1. Hélas cette idée donne des rotations à vitesses non uniformes.
Finalement un autre moyen encore plus simple était envisageable : ne transporter que les normes des vecteurs mais ajuster la matrice des coûts de déplacement en fonction des rotations à effectuer. C'est ce que j'ai essayé.

La nouvelle matrice de coûts Désormais la matrice C dépend de μ et ν. Dans cette version μ et ν sont à valeurs complexes. On pose alors $C_{i,j} = d_{\mathcal{X}}(x_i, y_j) + \gamma|\arg(\mu_i) - \arg(\nu_j)|$ où la différence d'arguments est prise dans $]-\pi, \pi]$. γ est un nouveau paramètre qui permet de choisir

l'importance de la rotation dans le coût de transport. J'ai personnellement choisi sa valeur de sorte à avoir $\gamma\pi$ qui vaut la moitié du diamètre de la surface.
Une fois cette matrice de coûts obtenue, on utilise l'algorithme de Sinkhorn classique entre les distributions à valeurs réelles $\left(\frac{|\mu_i|}{\sum_k |\mu_k|}\right)_i$ et $\left(\frac{|\nu_j|}{\sum_k |\nu_k|}\right)_j$.

Interpolation Pour l'interpolation il y a plusieurs subtilités. La première est sur le fait que notre plan de transport T n'est utile que pour les normes des vecteurs à déplacer. Ainsi $T^t_{i,j}$ donnée par la formule (3) est la norme à l'instant t du vecteur qui se déplace de x_i à y_j. Pour l'argument de ce vecteur, il suffit de prendre le barycentre des arguments de μ_i et ν_j pondéré par $(1-t)$ et t. Attention il faut au préalable ajoute $\pm 2\pi$ à $\arg(\mu_i)$ afin d'avoir $|\arg(\mu_i)-\arg(\nu_j)| \leqslant \pi$ si ce n'est pas déjà le cas. Sinon la rotation faite par le vecteur qui va de la face i à j ne sera pas forcément la rotation la plus courte. La seconde décision à prendre est sur la nouvelle forme de l'équation (4). En effet, on peut sommer des complexes avec l'addition classique sur les complexes ou bien essayer de définir une nouvelle addition qui aurait des propriétés intéressantes comme $|a \oplus b| = |a| + |b|$ afin de conserver la masse totale du champ.
Voici alors les différentes méthodes que j'ai testé :

- la somme classique entre des complexes.
- Utiliser l'opération $a \oplus b = (a+b)\dfrac{|a|+|b|}{|a+b|}$.
- Si on effectue des interpolations à des temps t_k croissants. Alors sur la face x_i au temps t_{k+1}, on peut utiliser l'opération

  $$a \oplus_{i,k} b = (|a|+|b|)\exp\left(i\frac{|a|(\arg(a)-\arg(\mu_i^{t_k}))+|b|(\arg(b)-\arg(\mu_i^{t_k}))}{|a|+|b|}\right)\exp\left(i\arg(\mu_i^{t_k})\right)$$

  où les deux différences d'arguments sont prises dans $]-\pi,\pi]$. Avec cette méthode on part du principe que $\mu_i^{t_{k+1}}$ et $\mu_i^{t_k}$ sont proches ce qui permettrait alors d'empêcher les retournements de vecteurs.
- Convertir les complexes en PSD comme pour la version quantum (à la différence qu'on laisse la seconde valeur propre nulle) puis sommer les PSD et reconvertir en complexe. En écrivant M_a la PSD de a, on a $|a| = \mathrm{tr}(M_a)$ car on prend une seconde valeur propre nulle. Si on reconverti en complexe en prenant pour module du complexe la plus grande valeur propre de la PSD on a pas $|a \oplus b| = |a| + |b|$. Par contre si on prend pour module la trace on a bien l'égalité. J'ai finalement opté pour prendre $v_1 + \alpha v_2$ comme module où v_1 est la plus grande valeur propre, v_2 la plus petite et α un coefficient dans $[0,1]$.

Ma conclusion de ces différents essais est qu'avec la première méthode il y a trop de soucis de pertes de masses. Avec les deux suivantes les problèmes que l'on obtient sont des changements brutaux de directions. La dernière méthode entraîne également des pertes de masse si α est trop petit et aussi des changements brutaux de directions. Augmenter α permet alors de réduire les problèmes de masse mais rend plus visible les soucis de changements de directions bien qu'il n'y en ai finalement peu. C'est finalement pour cette dernière méthode que j'ai opté.

Résultats Les résultats obtenus sont très proches de la méthode quantum, mais le gros avantage est la rapidité des calculs. En effet pour $\epsilon = 0.23$, le coefficient de la rotation dans le coût de

transport qui vaut 0.36 et $nSteps = 45$, on obtient encore des variations de l'ordre d'un millième de $T\mathbb{1}_m$ et $T^T\mathbb{1}_n$ sur la dernière itération et le temps de calcul est de 10s. Le temps de calcul est donc 13 fois plus court. De plus on a plus le soucis des vecteurs à très faibles normes sur les bords du tore.
Vous remarquerez qu'on a pris ε plus grand qu'avec la méthode quantum. Cela est dû au fait que les distances entre les faces sont plus grandes puisqu'on rajoute le coût de rotation.

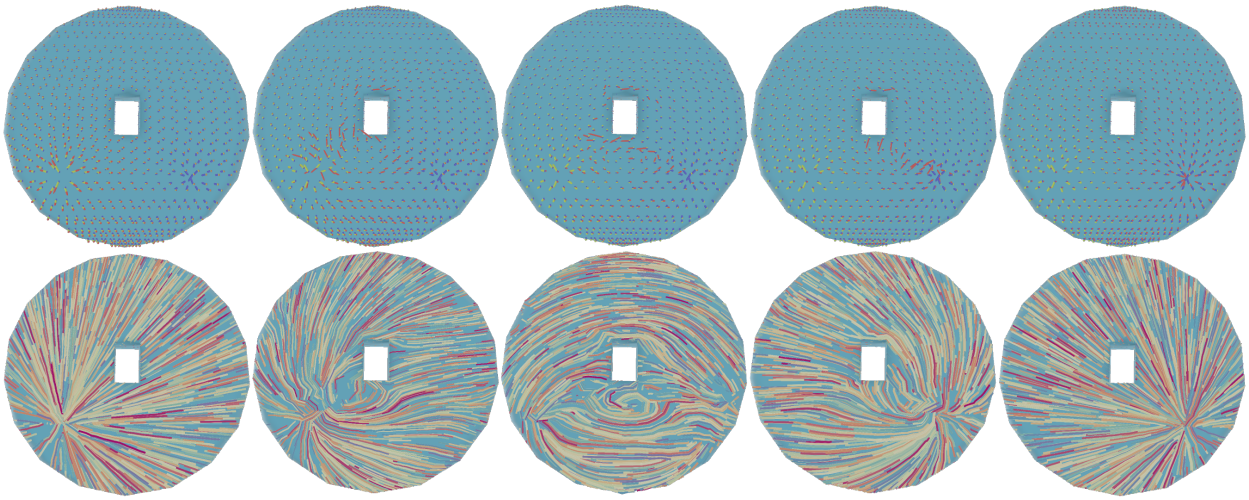

FIGURE 17 – Interpolation utilisant la nouvelle idée.
La première ligne est la représentation vectorielle et la deuxième la représentation par lignes de champ.

2.5 Optimisations

On note toujours F le nombre de faces de notre surface. En supposant qu'une géodésique entre deux faces traverse en moyenne un nombre de faces de l'ordre de $\sqrt{F}$, la complexité du calcul d'une interpolation en connaissant le plan de transport est alors en $O(\sqrt{F}F^2)$. Ce calcul peut alors prendre du temps. J'ai donc implémenté une fonction qui ne conserve que les coefficients du plan de transport les plus élevés tout en conservant au moins un certain pourcentage de la masse de chaque colonne et chaque ligne de la matrice.
Plus le coefficient ϵ est faible plus cette technique est efficace. Ainsi en conservant 92% de la masse du plan de transport j'ai pu diviser le temps de calcul par un facteur pouvant aller de 10 à 30 et les différences dans l'interpolation obtenue sont presque imperceptibles.
Toujours pour une plus grande rapidité de calculs, les codes ont aussi été parallélisés avec open mp.

3 Conclusion

J'ai réussi à obtenir une interpolation entre des champs de vecteurs tangents plutôt convenable par rapport à une interpolation linéaire définie par $\mu_i^t = (1-t)\mu_i + t\nu_i$. En effet avec une telle interpolation, la singularité ne se déplace pas et pour $0 < t < 1$, les singularités des champs μ et ν sont présentes (FIGURE 8). En revanche la complexité du calcul du plan de transport est en

$O(F^2 nSteps)$ ce qui devient problématique pour des surfaces qui ont un nombre de faces d'un ordre supérieur à 10^4. L'optimisation du paragraphe précédent permet de réduire les temps de calculs qui viennent après le calcul du plan de transport car on supprime une bonne partie des coefficients mais il serait bien de ne pas calculer le plan de transport en entier et de supprimer certains coefficients du plan de transport durant les itérations de Sinkhorn. Cela pourrait être une nouvelle piste a exploré. La deuxième solution proposé n'utilisant pas de PSD semblent être la plus intéressante. Elle est plus rapide à calculer et donne un résultat tout aussi correct si ce n'est plus puisque cette solution n'engendre pas de problèmes de vecteurs à très faibles normes sur les bords du tore.

A Contexte institutionnel et social

J'ai effectué ce stage au LIRIS (Laboratoire d'InfoRmatique en Image et Systèmes d'information) à l'université Claude Bernard Lyon 1 avec David Coeurjolly et Nicolas Bonneel deux chercheurs du CNRS. Le LIRIS a aussi des laboratoires à l'INSA Lyon, L'université Lyon 2 et l'école Centrale Lyon. Le LIRIS compte 6 pôles de compétences et 14 équipes. Mon stage se trouvait dans le cadre du pôle Géométrie et Modélisation. Nicolas et David sont tous les deux dans l'équipe M2DisCo (Modèles Multirésolution, Discrets et Combinatoires) qui s'intéresse à l'analyse géométrique et topologique d'objets modélisés par des structures discrètes et combinatoires. Nicolas fait aussi parti de l'équipe GeoMod (Modélisation Géométrique). Quant à David il est aussi directeur de la Fédération Informatique de Lyon.
Durant le stage j'ai pu assisté à la conférence de géométrie JIG 2018 ainsi qu'à une réunion du projet ANR CoMeDiC.

Références

[1] Fernando de Goes, Mathieu Desbrun, and Yiying Tong. Vector field processing on triangle meshes. In *SIGGRAPH Courses*, pages 27 :1–27 :49. ACM, 2016.

[2] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Globally optimal direction fields. *ACM Trans. Graph.*, 32(4) :59 :1–59 :10, 2013.

[3] Gabriel Peyré, Lenaïc Chizat, François-Xavier Vialard, and Justin Solomon. Quantum optimal transport for tensor field processing. *CoRR*, abs/1612.08731, 2016.

[4] Gabriel Peyré and Marco Cuturi. *Computational Optimal Transport*. arxiv.org/abs/1803.00567, 2018.

[5] Justin Solomon, Fernando de Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas J. Guibas. Convolutional wasserstein distances : efficient optimal transportation on geometric domains. *ACM Trans. Graph.*, 34(4) :66 :1–66 :11, 2015.