# Calcul géométrique rapide pour la vision et les sciences des données

Jean Feydy
INRIA Paris

17 septembre 2021

# CV : un parcours “maths-info” dirigé vers la médecine

Formation en **mathématiques** et **sciences des données** :

**2012–2016** ENS Paris, mathématiques.

**2014–2015** M2 mathématiques, vision, apprentissage à l’ENS Cachan.

**2016–2019** Thèse en **imagerie médicale** avec Alain Trouvé de l’ENS Cachan.

**2019–2021** ***Deep learning* géométrique** avec Michael Bronstein de l’Imperial College.

**2021+** **Analyse de données médicales** dans l’équipe HeKA, INRIA Paris.

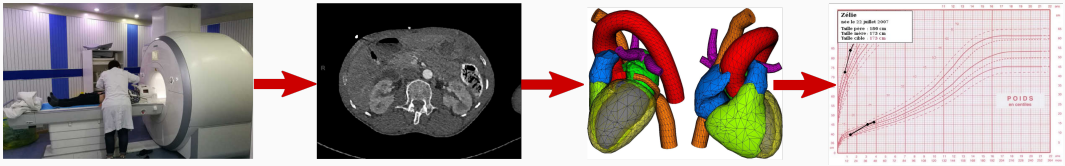
Liens étroits avec la **médecine** :

**2015** Débruitage d’images chez **Siemens Healthcare** à Princeton.

**2019+** MasterClass IA–Imagerie, pour les **internes de radio** à l’Université de Paris.

**2020+** Colloque sur l’**Imagerie médicale à l’heure de l’IA**, à l’Institut du Cerveau.

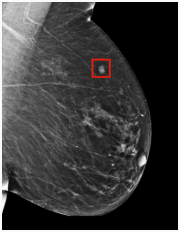
# Ma motivation : l'analyse de données médicales



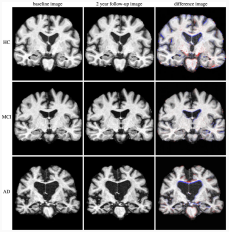

Trois **caractéristiques** :

- Données **hétérogènes** : parcours de soin, images, etc.
- Petits échantillons stratifiés : 10 - 1 000 patients par groupe.
- La gestions des ***outliers*** et de la **queue lourde** des distributions est une priorité.

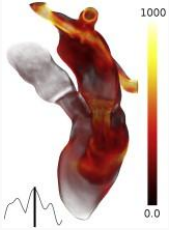
# L'anatomie computationnelle [CSG19, LSG+18, CMN14]



Détecter un objet




Analyser une variation




Recaler un modèle

Quelques particularités, dans le contexte de la vision par ordinateur :

- Prises de vues **standardisées**, sans occlusions.
- Travail de **précision** (millimétrique).
- Importance des **garanties** de robustesse et de régularité.

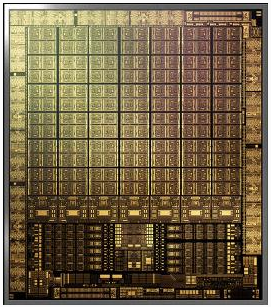
# Un domaine en mutation

**Enjeu.** Développer des modèles qui combinent **expertise** médicale et jeux de **données**.

**Défi.** Arrivée des **cartes graphiques** (GPU) :

- Excellent rapport **qualité-prix** :
  $1\,000€ \simeq 1\,000$ cœurs $\simeq 10^{12}$ opérations/s.
- **Difficulté** : contraintes sur la gestion des **registres**.

Écosystème Python “grand public” consolidé autour d’un **petit nombre d’opérations élémentaires**.



**7,000 cœurs** en batterie sur un GPU.

# Mon projet : un investissement de long terme dans les fondations du domaine

**Solution.** **Étendre la boîte à outils standard** en sciences des données pour répondre aux défis posés par le monde médical.

**Faciliter** le développement de **modèles avancés**, sans compromis sur les performances numériques.

Un travail de fond, **fondations** numériques $\longrightarrow$ applications de **haut niveau** :

1. Manipulation efficace de **matrices "symboliques"** (distances, noyaux, etc.).
2. **Transport optimal** : méthodes de tri généralisées.
3. ***Deep learning*** géométrique et applications **biomédicales**.

**Avenir** de ces outils et **perspectives cliniques**.

# 1. Matrices symboliques

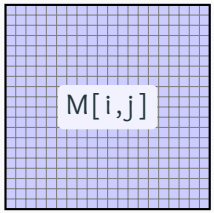
# Les outils de calcul scientifique manipulent (surtout) des matrices

**Contexte.** **Accès mémoires** contraignants sur GPU :

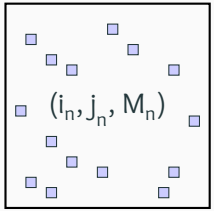
- **Temps de transfert** vers les registres pénalisent l'utilisation de matrices **denses**.
- Prime aux accès mémoires **contigus** pénalise l'utilisation de matrices **creuses**.

**Défi.** Pour atteindre des performances optimales :

- Se **restreindre** aux opérations supportées directement par le constructeur : convolutions, FFT, etc.
- Développer soi-même de nouvelles routines numériques en C++/CUDA (FAISS, KPConv…) : **plusieurs mois de travail**.




Matrice dense




Matrice creuse

# La bibliothèque KeOps : support efficace des matrices symboliques

**Solution.** KeOps – `www.kernel-operations.io` :

- Pour PyTorch, NumPy, Matlab et R sur **CPU et GPU**.
- **Différentiation automatique**.
- **Compilation** à la volée de **schémas optimisés** en C++, exécutés pour chaque **réduction** (somme, min, etc.).

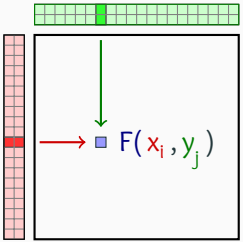
Si la formule "F" est simple ($\leqslant$ 100 opérations arithmétiques) :

calcul "100k $\times$ 100k" $\longrightarrow$ 10ms – 100ms,

calcul "1M $\times$ 1M" $\longrightarrow$ 1s – 10s.

Limite matérielle de $10^{12}$ opérations/s.

Exécution **10 à 100 fois plus rapide** / code GPU standard pour une large gamme de problèmes.




**Matrice symbolique**

Formules + données

- Distances $d(x_i, y_j)$.
- Noyaux $k(x_i, y_j)$.
- Nombreuses transformées.

# Premier exemple : la recherche de plus proches voisins

On crée de grands nuages de points avec **la syntaxe PyTorch standard** :

```
import torch
N, M, D = 10**6, 10**6, 50
x = torch.rand(N, 1, D).cuda()  # (1M,  1, 50) array
y = torch.rand(1, M, D).cuda()  # ( 1, 1M, 50) array
```

On convertit les tableaux **denses** en matrices **symboliques** :

```
from pykeops.torch import LazyTensor
x_i, y_j = LazyTensor(x), LazyTensor(y)
```

On crée une grande **matrice symbolique** des distances au carré :

```
D_ij = ((x_i - y_j) ** 2).sum(dim=2)  # (1M, 1M) symbolique
```

On utilise une **réduction** `.argmin()` pour chercher les plus proches voisins :

```
indices_i = D_ij.argmin(dim=1)  # -> tenseur torch standard
```

# La bibliothèque KeOps combine performances et flexibilité

Code précédent = méthode **compétitive** avec
les implémentations C++/CUDA de référence (**FAISS**)...
Et qui peut être utilisée avec **n'importe quelle métrique** !

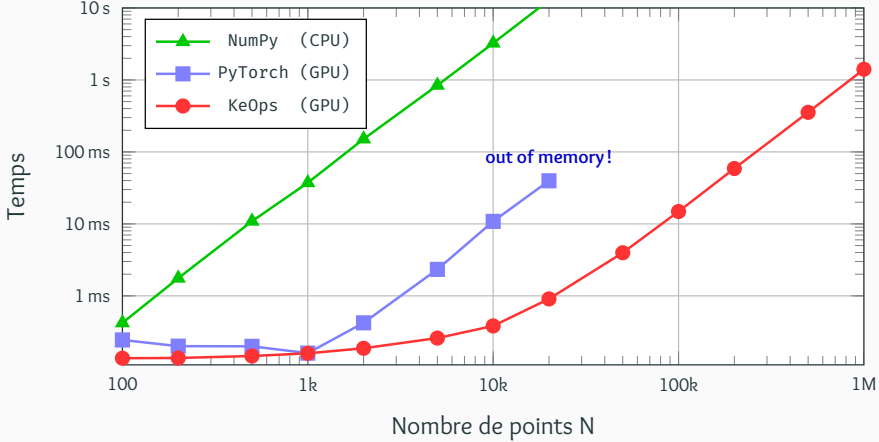
```
D_ij = ((x_i - x_j) ** 2).sum(dim=2)     # Euclidienne
M_ij =  (x_i - x_j).abs().sum(dim=2)     # Manhattan
C_ij = 1 - (x_i | x_j)                   # Cosinus
H_ij = D_ij / (x_i[...,0] * x_j[...,0])  # Hyperbolique
```

KeOps supporte des **formules** et **variables** arbitraires avec :

- **Réductions :** somme, log-sum-exp, K-min, produit matrice-vecteur, etc.
- **Opérations :** $+$, $\times$, sqrt, exp, réseaux de neurones, etc.
- **Schémas avancés :** calcul par batches, parcimonie par blocs, etc.
- **Différentiation automatique :** intégration complète avec PyTorch.

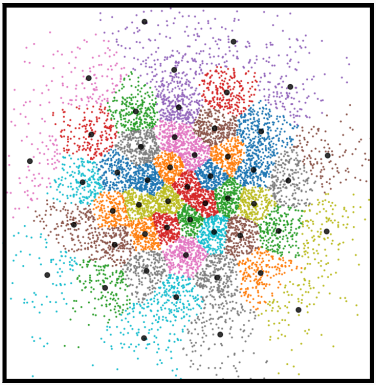
# KeOps nous permet de tirer le meilleur parti de notre matériel, simplement

Benchmark d'une **convolution** avec un noyau Gaussien,
entre **nuages de N points 3D** sur un GPU RTX 2080 Ti.



# Applications

# KeOps facilite la recherche en apprentissage machine

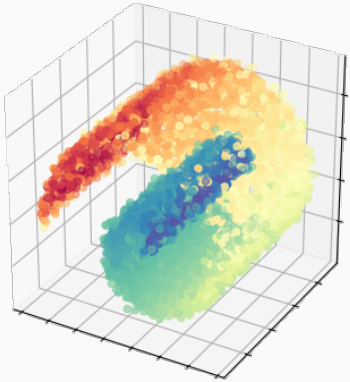


K-Means.

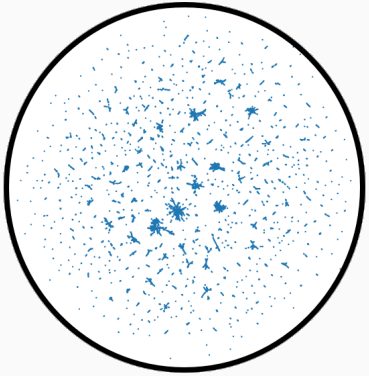
Mixtures de Gaussiennes.

KeOps est utilisable avec la formule, distance ou noyau de **votre choix** !

# KeOps facilite la recherche en apprentissage machine



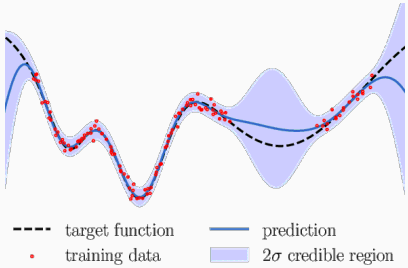
Analyse spectrale.



UMAP dans un espace hyperbolique.

KeOps est utilisable avec la formule, distance ou noyau de **votre choix** !

**Un outil standard pour la régression** [Lec18] :




Sous le capot, on résout un **système linéaire à noyaux** :

$$(\lambda\,\mathrm{Id} + K_{xx})\,a \;=\; b \qquad \text{i.e.} \qquad a \;\leftarrow\; (\lambda\,\mathrm{Id} + K_{xx})^{-1} b$$

où $\lambda \geqslant 0$ et $(K_{xx})_{i,j} = k(x_i, x_j)$ est une matrice symétrique, définie positive.

**Les tenseurs symboliques** $(K_{xx})_{i,j} = k(x_i, x_j)$ **:**

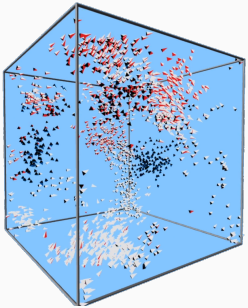
- Peuvent être interfacés avec des **solveurs standards** : SciPy, GPyTorch, etc.
- GPyTorch sur le jeu de données 3DRoad (N = 278k, D = 3) :

  **7h avec 8 GPUs** $\rightarrow$ **15mn avec 1 GPU.**

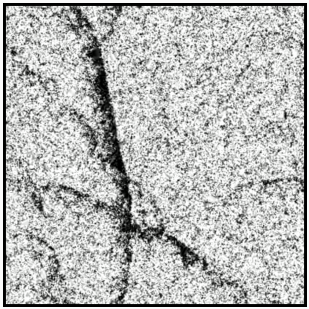
- Fournissent une **plateforme efficace et flexible** pour les projets de recherche :
  voir par exemple *Kernel methods through the roof : handling **billions of points** efficiently*,
  par G. Meanti, L. Carratino, L. Rosasco, A. Rudi (2020).

# KeOps permet aux chercheurs de se concentrer sur leurs modèles

Quelques applications aux **systèmes dynamiques** [DM08, DFMAT17]
et aux **statistiques** [CDF19] avec A. Diez, G. Clarté et P. Degond :



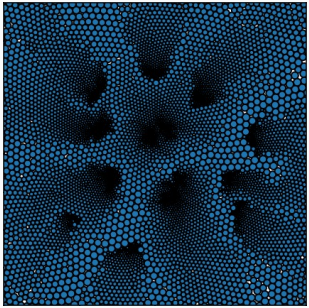
Modèle de Vicsek orienté en 3D,
démo interactive avec 2k **oiseaux**.



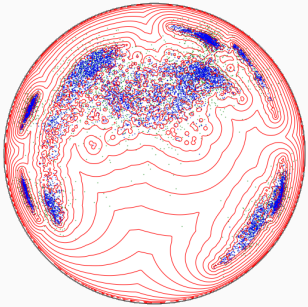
Modèle de Vicsek 2D sur le tore,
en temps réel avec 100k **nageurs**.

# KeOps permet aux chercheurs de se concentrer sur leurs modèles

$\Longrightarrow$ On simule plusieurs **millions d'agents** avec de simples scripts Python.



Problème de ***packing***
avec 10k boules répulsives.



**Échantillonnage** de Monte Carlo Collectif
dans un espace hyperbolique.

# 2. Transport optimal

# Le transport optimal (OT) généralise le tri aux espaces de dimension $D > 1$

**Contexte.** Si $A = (x_1, \dots, x_N)$ et $B = (y_1, \dots, y_N)$ sont deux nuages de N points dans $\mathbb{R}^D$, on définit :

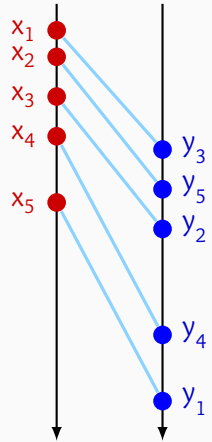
$$\mathrm{OT}(A, B) = \min_{\sigma \in \mathcal{S}_N} \frac{1}{2N} \sum_{i=1}^{N} \| x_i - y_{\sigma(i)} \|^2$$

Généralise le **tri** aux espaces métriques.

**Problème linéaire** sur la matrice de permutation $P$ :

$$\mathrm{OT}(A, B) = \min_{P \in \mathbb{R}^{N \times N}} \frac{1}{2N} \sum_{i,j=1}^{N} P_{i,j} \cdot \| x_i - y_j \|^2 ,$$

$$\text{s.t.} \quad P_{i,j} \geqslant 0 \qquad \underbrace{\textstyle\sum_j P_{i,j} = 1}_{\text{Chaque point au départ...}} \qquad \underbrace{\textstyle\sum_i P_{i,j} = 1}_{\text{est transporté jusqu'à l'arrivée.}} .$$




assignement

$\sigma : [\![1, 5]\!] \to [\![1, 5]\!]$

# Propriétés clés de cette distance “à permutation près”

La distance de Wasserstein $\sqrt{\mathrm{OT}}(A, B)$ est :

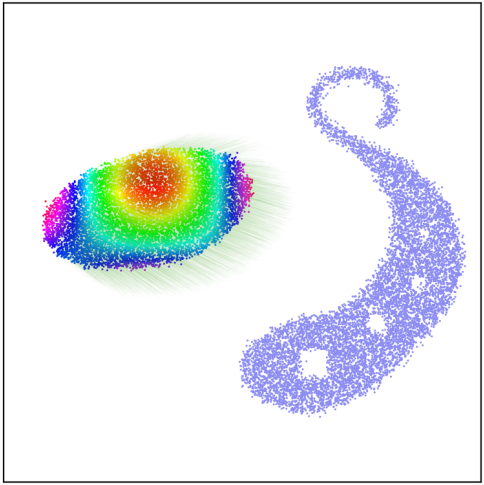
- **Symétrique :** $\mathrm{OT}(A, B) = \mathrm{OT}(B, A)$.
- **Positive :** $\mathrm{OT}(A, B) \geqslant 0$.
- **Définie :** $\mathrm{OT}(A, B) = 0 \iff A = B$.
- **Compatible avec les translations :** $\mathrm{OT}(A, \mathrm{Translation}_{\vec{v}}(A)) = \frac{1}{2}\|\vec{v}\|^2$.
- Plus généralement, OT retrouve l’unique **gradient d’une fonction convexe** $T = \nabla\phi$ qui envoie $A$ sur $B$ :

  En dimension 1, $(x_i - x_j) \cdot (y_{\sigma(i)} - y_{\sigma(j)}) \geqslant 0$

  En dimension D, $\langle x_i - x_j \ , \ T(x_i) - T(x_j) \rangle_{\mathbb{R}^D} \geqslant 0$.

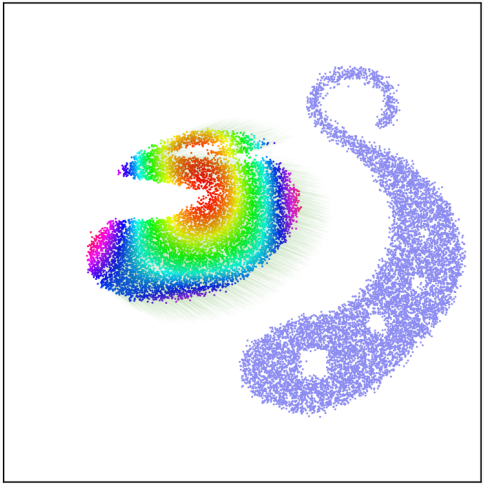
$\Longrightarrow$ Généralisation attrayante de la notion d’**application croissante**.

# Un exemple simple en 2D



t = .00

# Un exemple simple en 2D

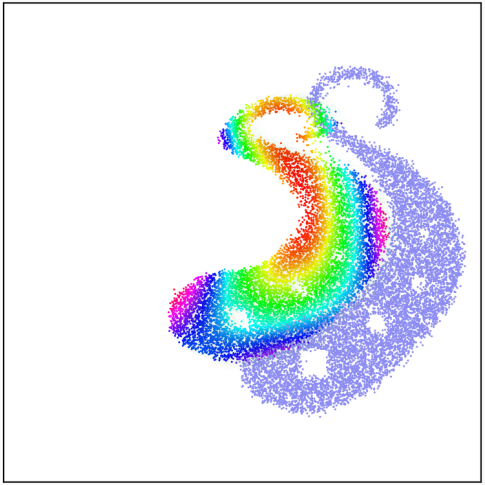


$t = .25$

# Un exemple simple en 2D

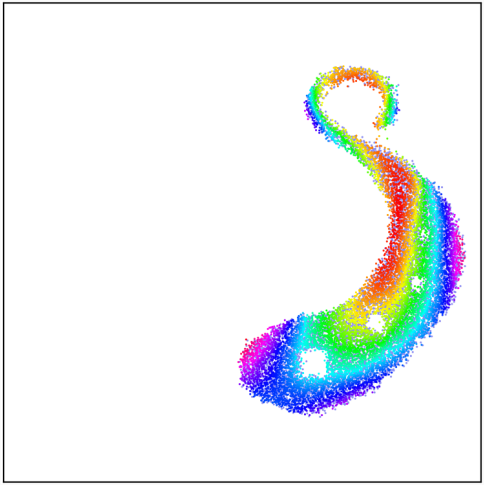
$t = .50$

# Un exemple simple en 2D



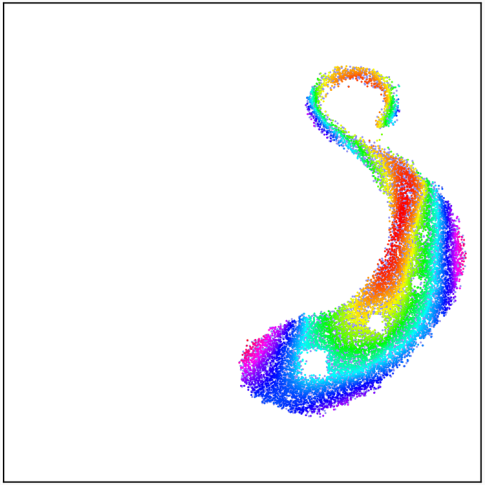
$t = 1.00$

# Un exemple simple en 2D



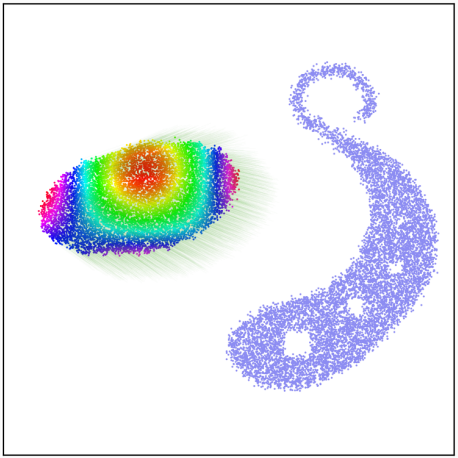
$t = 5.00$

# Un exemple simple en 2D

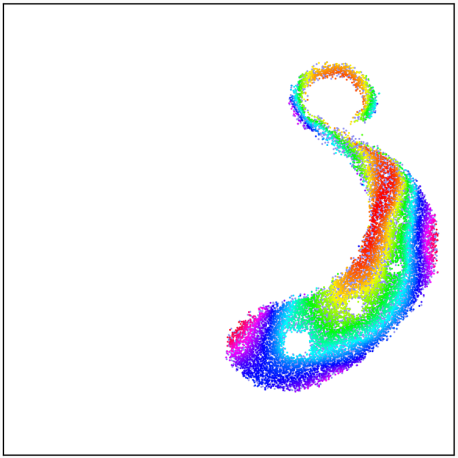


$t = 10.00$

# Un modèle efficace... Mais attention aux déchirures !

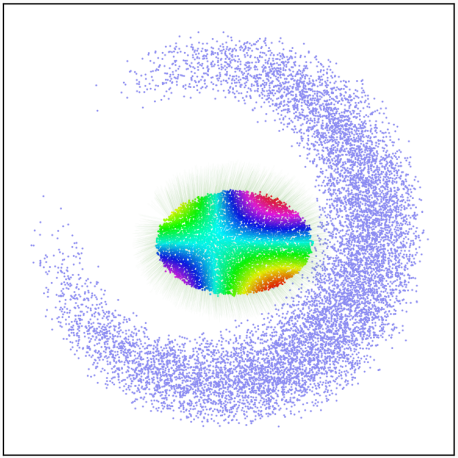


Avant

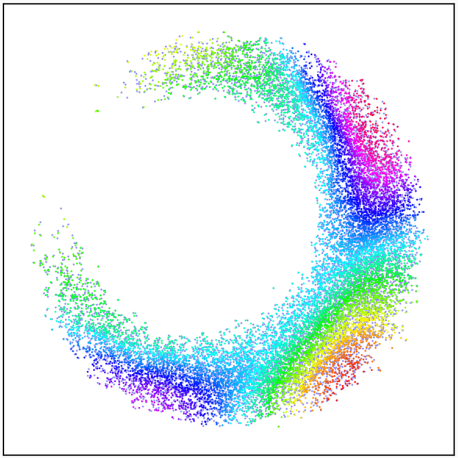


Après

# Un modèle efficace… Mais attention aux déchirures !

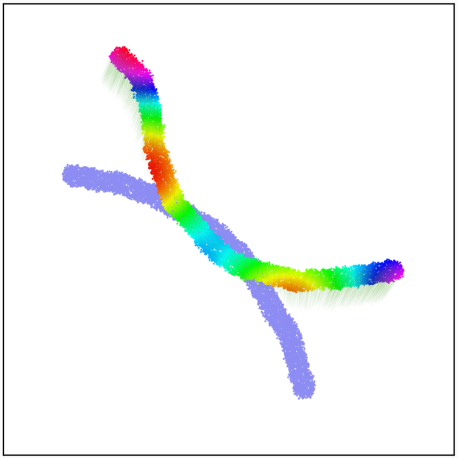


Avant

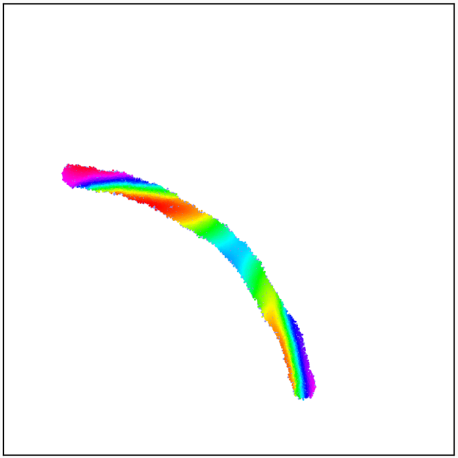


Après

# Un modèle efficace... Mais attention aux déchirures !

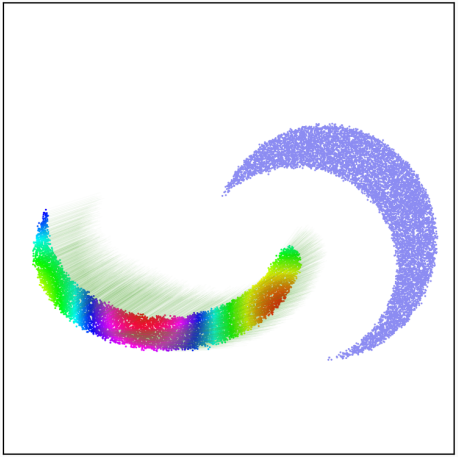


Avant

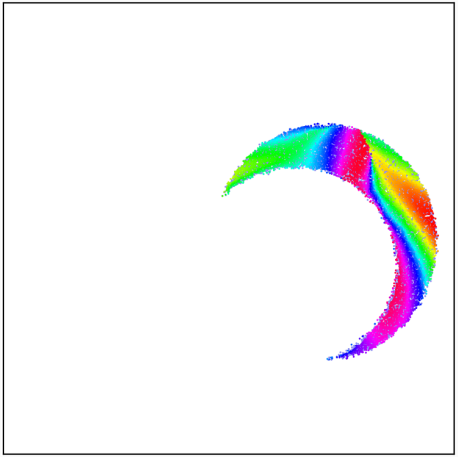


Après

# Un modèle efficace... Mais attention aux déchirures !



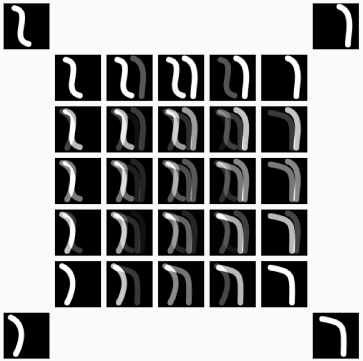
Avant



Après

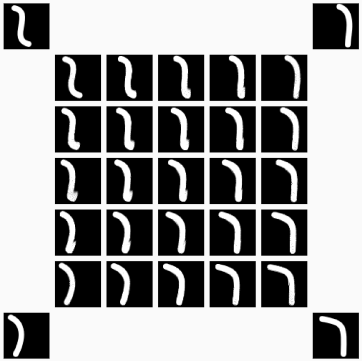
# Solutions géométriques aux problèmes des moindres carrés [AC11]

$$\text{Barycentre } \mathsf{A}^* \;=\; \arg\min_{\mathsf{A}} \sum_{i=1}^{4} \lambda_i \,\text{Loss}(\,\mathsf{A}\,,\,\mathsf{B}_i\,)\;.$$



Barycentres **euclidiens.**

$\text{Loss}(\mathsf{A},\mathsf{B}) = \|\mathsf{A}-\mathsf{B}\|_{L^2}^2$



Barycentres de **Wasserstein.**

$\text{Loss}(\mathsf{A},\mathsf{B}) = \text{OT}(\mathsf{A},\mathsf{B})$

# Régularisation et biais entropique

**Défi.** Assignement linéaire : difficile à résoudre en toute généralité.

**Structure** de la matrice des distances $\|x_i - y_j\|$

$\Longrightarrow$ **Accélérer** les calculs.

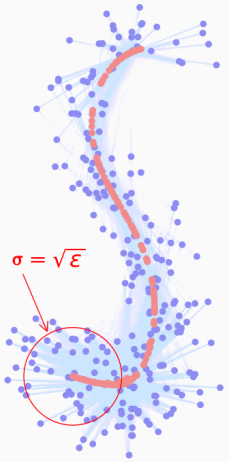
Outil fondamental : le **transport régularisé**

$\mathrm{OT}_\varepsilon(A, B) \simeq \mathrm{OT}(A, B)$ + **pénalité entropique** d'intensité $\varepsilon > 0$.

Approximation **lisse** et strictement **convexe** : plus facile à étudier, très populaire algorithme de **Sinkhorn** (ou "*SoftAssign*").

En revanche, elle ne **vérifie pas les axiomes d'une distance** :

$\mathrm{OT}_\varepsilon(B, B) > 0$.




$\arg\min_A \mathrm{OT}_\varepsilon(A, B)$.

# Solution théorique : garantie de robustesse au biais entropique

**Solution.** Les **divergences de Sinkhorn** sont définies par :

$$S_\varepsilon(A, B) = \mathrm{OT}_\varepsilon(A, B) - \tfrac{1}{2}\mathrm{OT}_\varepsilon(A, A) - \tfrac{1}{2}\mathrm{OT}_\varepsilon(B, B)$$

afin de retrouver une valeur nulle lorsque $A = B$.

**Théorème ( $S_\varepsilon$ se prête bien à l'optimisation)**

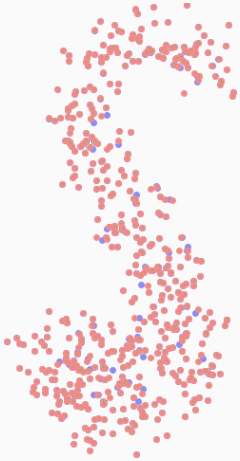
*Pour tous échantillons $A$ et $B$ :*

$$S_\varepsilon(A, B) \geqslant 0 \quad \textit{avec égalité ssi. } A = B,$$

$$A \mapsto S_\varepsilon(A, B) \quad \textit{est convexe au sens des mesures,}$$

*différentiable et métrise la convergence en loi.*

*On généralise ce résultat aux **mesures** de Radon positives, à des **métriques** $\|x_i - y_j\|$ générales et au cas "**unbalanced**".*



$\arg\min_A S_\varepsilon(A, B)$.

# En pratique, comment résoudre le problème de transport optimal ?

Dates clés pour le transport optimal avec N points :

- [Kan42] : problème **dual** de Kantorovitch.
- [Kuh55] : algorithme **hongrois** en $O(\mathrm{N}^3)$.
- [Ber79] : méthode des **enchères** en $O(\mathrm{N}^2)$.
- [KY94] : **SoftAssign** = Sinkhorn + recuit simulé, en $O(\mathrm{N}^2)$.
- [GRL$^+$98, CR00] : **Robust Point Matching** = Sinkhorn comme fonction d'erreur.
- [Cut13] : utilisation des **GPUs.**
- [Mér11, Lév15, Sch19] : solveurs **multi-échelles** en $O(\mathrm{N}\log\mathrm{N})$.

- **Solution,** aujourd'hui : **algorithme de Sinkhorn multi-échelles, sur GPU.**

$\Longrightarrow$ Algorithme de **tri rapide** généralisé.

# Solution pratique : l'algorithme de Sinkhorn multi-échelles

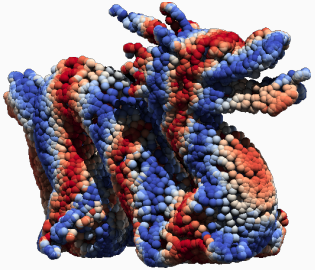
Entre 2015 et 2020, l'état de l'art pour le transport optimal discret a été amélioré par **un à trois ordres de grandeur** – selon le régime d'utilisation :

$$\text{Sinkhorn GPU} \xrightarrow{\times 10} + \text{KeOps} \xrightarrow{\times 10} + \text{Recuit simulé} \xrightarrow{\times 10} + \text{Multi-échelle}$$

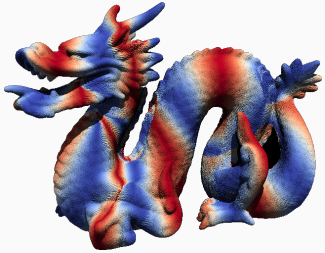
Résoudre un problème de transport avec une précision de 1% :

```
pip install
  geomloss
     +
  GPU récent
  (1 000 €)
```

$\Longrightarrow$



10k points en 30-50ms



**100k points en 100-200ms**

# 3. Deep learning géométrique

# Concevoir des modèles entraînables et adaptés aux experts

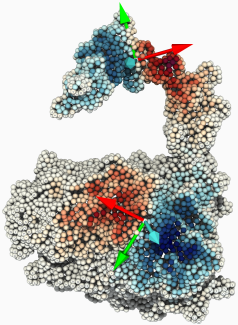
**Enjeu.** Modèles profonds sur des **domaines non-euclidiens** (nuages de points, surfaces, graphes, etc.), au-delà des images 2D/3D.

**Défi.** En dépit d'un intérêt industriel croissant, ces modèles restent **mal supportés** numériquement. Le C++/CUDA reste (souvent) incontournable.

**Solution.** En quelques lignes de Python, avec KeOps :

- Interactions **locales** : K-plus-proches voisins.
- Interactions **globales** : convolutions généralisées.

**Liberté** de modélisation
$\Longrightarrow$ **Spécificités** des domaines étudiés.



Convolution quasi-géodésique sur un nuage de points orientés.

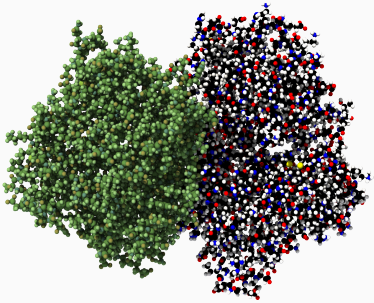
# Benchmarks de quelques “convolutions” sur ShapeNet (N = 2,048 points 3D)

$$f_i \leftarrow \sum_{x_j \in \mathrm{KNN}(x_i)} \mathrm{Conv}(x_i, x_j, f_j) \qquad \text{ou} \qquad \sum_{\text{tous les } x_j} \exp(-\|x_i - x_j\|^2 / 2\sigma^2) \cdot \mathrm{Conv}(x_i, x_j, f_j)$$

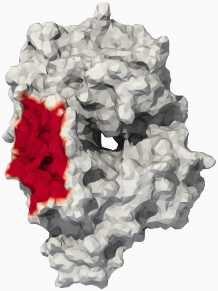
| | PyTorch | JAX | | KeOps | |
|---|---|---|---|---|---|
| Primitive $\mathrm{Conv}(x_i, x_j, f_j)$ | 40-NN | 40-NN | Gaussien | 40-NN | Gaussien |
| Moyennes locales | 686 µs | 1,052 µs | 469 µs | 121 µs | **12 µs** |
| Matrices de covariances | 700 µs | 1,093 µs | 1,259 µs | 138 µs | **23 µs** |
| MLP (H = 8) | 737 µs | 1,180 µs | 4,089 µs | 192 µs | **75 µs** |
| MLP (H = 16) | 775 µs | 1,253 µs | 7,043 µs | **240 µs** | 649 µs |

Les interactions “entre tous les points” deviennent **viables** :
nous ne sommes plus obligés de nous restreindre à des convolutions
très locales sur un graphe de K-plus-proches voisins.

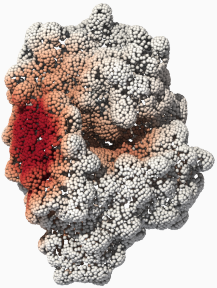
# Application au *docking* de protéines [SFCB20]



**(a)** Données brutes : atomes en 3D.

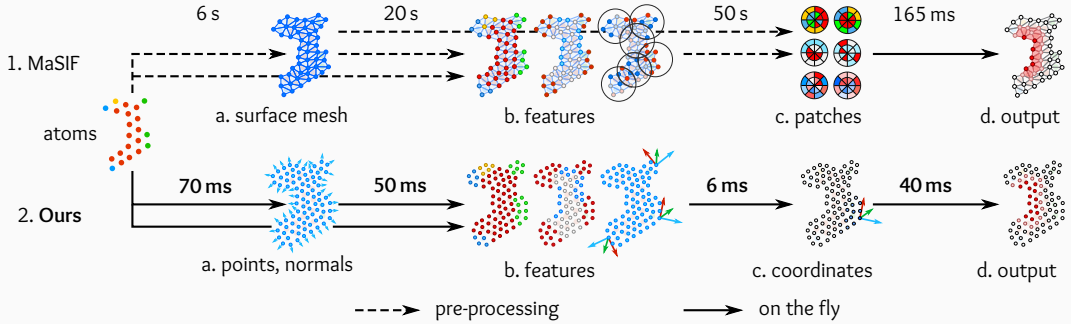


**(b)** Interface observée.



**(c)** Prédiction du modèle.

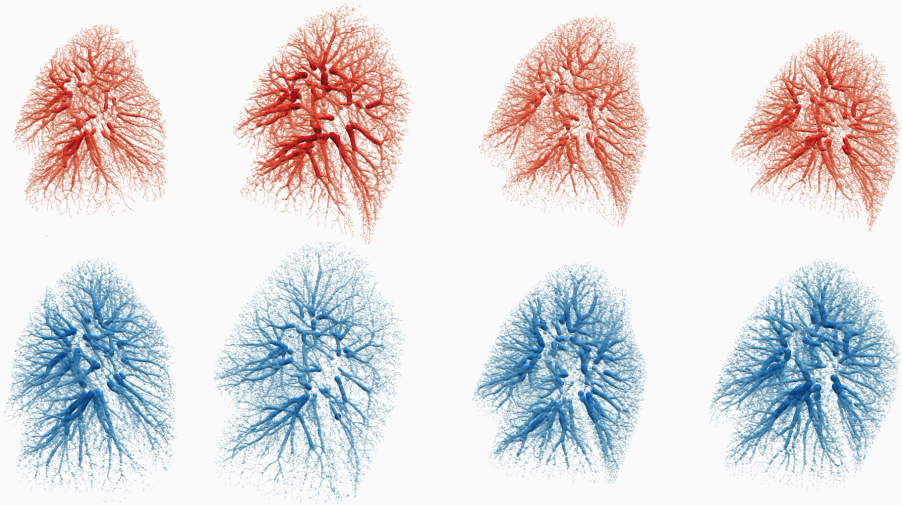
# “Fast end-to-end learning on protein surfaces” [SFCB20]





$\times 100 - \times 1{,}000$ plus rapide, léger et complètement différentiable.

# Application à l'appariement de poumons "Expiration – Inspiration"



Déformations **complexes**, haute **résolution** (50k–300k points), haute **précision** ($<$ 1mm).

# Un recalage en trois temps

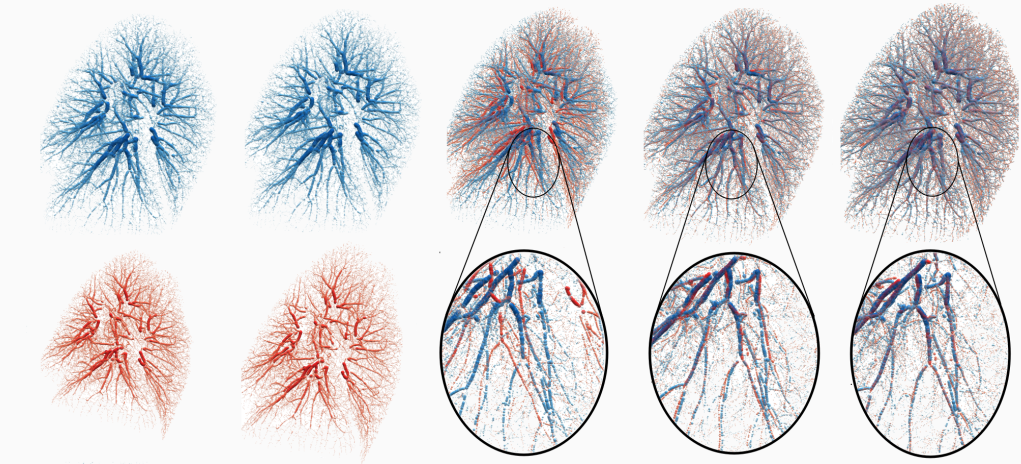
On combine :

1. Un **pré-recalage** global : OT + déformation affine.
2. Un **recalage** “au centimètre près” : réseau multi-échelles + difféomorphisme.
3. Un **affinage** “au millimètre près” : OT + régularisation spline.

Cette méthode **pragmatique** :

- Est **facile à entraîner** sur des données synthétiques.
- Passe à l’échelle : 100k points en $<$ 1s.
- Donne d’excellents résultats : **KITTI** (*scene flow*) et **DirLab** (poumons).

# Un recalage en trois temps



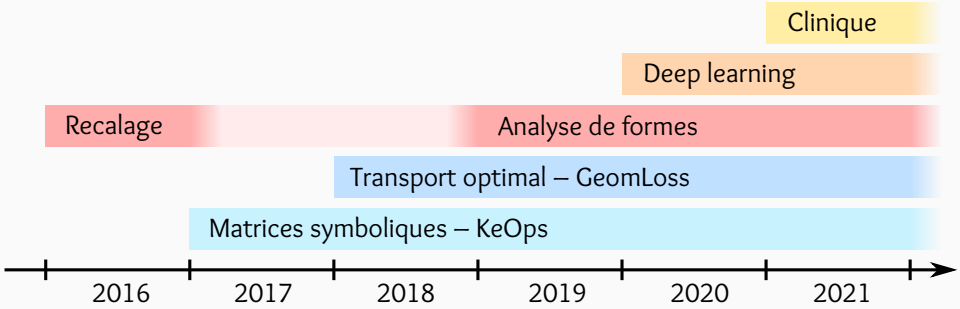
0. Données. 1. Pré-recalage. Zoom ! 2. Au cm près. 3. Au mm près.

*Accurate point cloud registration with **robust** optimal transport* : à paraître à la fin du mois.

# Bilan : un investissement de long terme qui porte ses fruits

Deux évolutions majeures :

- “Gros” problème géométrique : $N > \mathbf{10k} \longrightarrow N > \mathbf{1M}$.
- Transport optimal : **problème** linéaire $+$ **tri rapide** généralisé.



# Avenir de ces outils, contexte scientifique

# Un réel travail d'équipe

Alain Trouvé

Thibault Séjourné

F.-X. Vialard

Gabriel Peyré

Benjamin Charlier

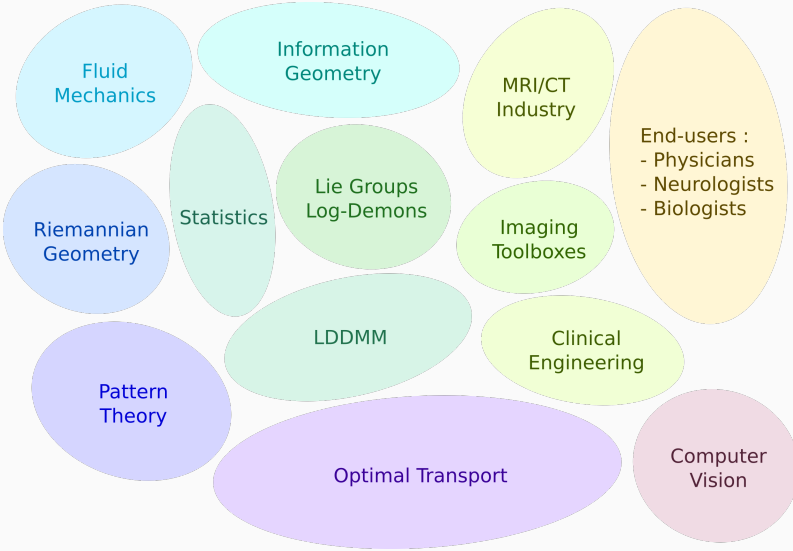
Joan Glaunès

Freyr Sverrisson

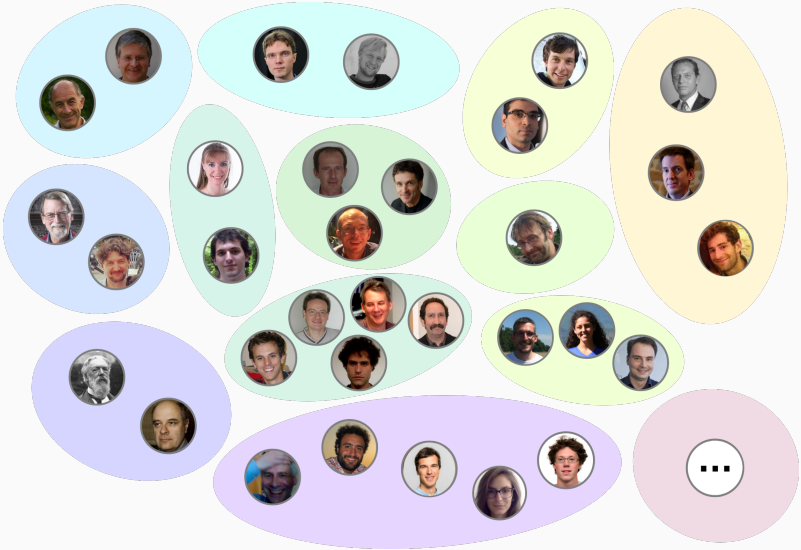
Shen Zhengyang

+ Marc Niethammer, Bruno Correia, Michael Bronstein...

# Promouvoir des interactions entre domaines



# Promouvoir des interactions entre domaines



# La révolution Python

L'émergence d'une boîte à outils ouverte et **modulaire**
a été une **bénédiction** pour la communauté.

Les bibliothèques de *deep learning* ont mis le **calcul sur GPU**
et la **différentiation automatique** à la portée de tous.
(Incroyable !)

Ces logiciels sont soutenus par des acteurs **industriels majeurs** (Google, Facebook…)
et ont permis à nos domaines **d'exploser** au cours de la dernière décennie.

**Interagir** avec d'autres chercheurs, des médecins
et des ingénieurs n'a jamais été aussi **facile**.

# La révolution Python

Mais d'un autre côté, PyTorch et TensorFlow ont aussi **biaisé** la recherche vers un **petit nombre** d'opérations **bien supportées** : convolutions et produits matrice-matrice.

Ce focus n'est **pas** la conséquence d'une limitation intrinsèque des GPUs : notre matériel est parfaitement capable de **simuler** de vastes **environnements 3D** en temps réel !

En tant que chercheurs, nous devons travailler à **ouvrir d'autres chemins.** Permettre le développement d'une gamme d'outils **complète**, des méthodes convexes et **robustes** aux modèles de *deep learning* **expressifs**.

# Notre contribution à la communauté

KeOps et GeomLoss sont :

- **Rapides** : 10 à 1,000 fois plus efficaces que des codes GPU standards.
- **Économes en mémoire** : $O(\mathrm{N})$ et pas $O(\mathrm{N}^2)$.
- **Versatiles**, avec une interface **transparente** : liberté !
- **Puissantes et bien documentées** : pensées pour la recherche.

- **Lentes** avec de **très grands vecteurs de *features*** en dimension $\mathrm{D} > 100$.

À venir cette année :

$\rightarrow$ **Méthodes d'approximation** (Nyström, etc.) dans KeOps.

$\rightarrow$ Support optimal des **barycentres** et des **images** dans GeomLoss.

# Un projet en pleine évolution

2017–18 **Preuve de concept** avec articles de conférences et codes en ligne.
Premiers retours de la communauté.

2019–21 **Bibliothèque stable** avec des théorèmes solides et une API bien documentée.
Backend KeOps pour des paquets de haut niveau.
**100k+ téléchargements** sur pip, etc.

2021–22 **Bibliothèque mûre** avec des articles d'application concrets.
Fonctionne *out-of-the-box* pour les étudiants et les ingénieurs.
$\Longrightarrow$ Refonte du moteur de compilation de KeOps pour la v1.6.
$\Longrightarrow$ GeomLoss comme backend pour POT v1.0.

2022+ **Une bibliothèque standard** avec de réelles applications cliniques ? C'est l'objectif !

# Conclusion

# Points clés

- Les matrices **symboliques** sont au ML **géométrique** ce que les matrices **creuses** sont au traitement de **graphes** :
  - ⟶ KeOps : **accélération x30** vs. PyTorch, TF et JAX.
  - ⟶ Utiles pour de nombreux problèmes.

- Transport optimal = **tri généralisé** :
  - ⟶ Appariement géométrique.
  - ⟶ Solveurs très rapides en $O(\mathrm{N} \log \mathrm{N})$.

- Ces outils ouvrent de **nouveaux chemins** pour les géomètres et les statisticiens :
  - ⟶ Les GPUs sont plus **versatiles** qu'on ne le croit.
  - ⟶ Travail constant sur les **fondations numériques** de la recherche, au-delà de ce pourquoi Google et Facebook sont prêts à payer.

# Conclusion

**KeOps** et **GeomLoss** stimulent la recherche sur :

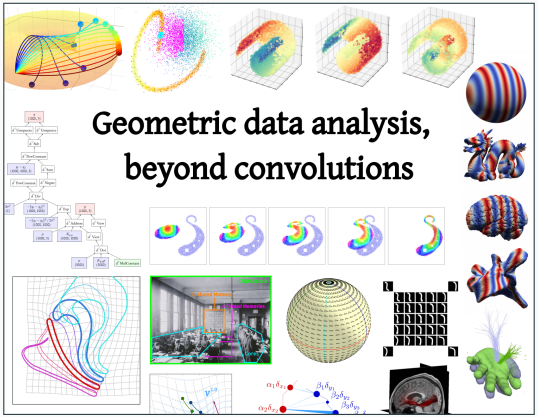
- Les méthodes de **clustering** : K-Means et EM rapides.
- La **représentation** des données : UMAP, KNN rapides avec n'importe quelle métrique.
- Les méthodes à **noyaux** : matrices de noyaux.
- Les **processus Gaussiens** : matrices de covariances.

- Le ***deep learning* géométrique** : convolutions sur les points.
- L'imagerie **médicale** : anatomie computationnelle.
- Les **statistiques géométriques** : au-delà des modèles Euclidiens.
- Le traitement du **langage** : *transformers* efficaces ?

Qu'en pensez-vous ?

# Notre documentation et de nombreux tutoriaux sont disponibles en ligne

$\Longrightarrow$ www.kernel-operations.io $\Longleftarrow$




www.jeanfeydy.com/geometric_data_analysis.pdf

# Annexe

# References i


M. Agueh and G. Carlier.

**Barycenters in the Wasserstein space.**

*SIAM Journal on Mathematical Analysis*, 43(2):904–924, 2011.

Dimitri P Bertsekas.

**A distributed algorithm for the assignment problem.**

*Lab. for Information and Decision Systems Working Paper, M.I.T., Cambridge, MA*, 1979.

# References ii


Grégoire Clarté, Antoine Diez, and Jean Feydy.

**Collective proposal distributions for nonlinear MCMC samplers : Mean-field theory and fast implementation.**

*arXiv preprint arXiv:1909.08988*, 2019.

Christophe Chnafa, Simon Mendez, and Franck Nicoud.

**Image-based large-eddy simulation in a realistic left heart.**

*Computers & Fluids*, 94:173–187, 2014.

# References iii


Haili Chui and Anand Rangarajan.

**A new algorithm for non-rigid point matching.**

In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 44–51. IEEE, 2000.

Adam Conner-Simons and Rachel Gordon.

**Using ai to predict breast cancer and personalize care.**

http://news.mit.edu/2019/using-ai-predict-breast-cancer-and-personalize-care-0507, 2019.

MIT CSAIL.

Marco Cuturi.

**Sinkhorn distances : Lightspeed computation of optimal transport.**

In *Advances in Neural Information Processing Systems*, pages 2292–2300, 2013.

Pierre Degond, Amic Frouvelle, Sara Merino-Aceituno, and Ariane Trescases.

**Alignment of self-propelled rigid bodies : from particle systems to macroscopic equations.**

In *International workshop on Stochastic Dynamics out of Equilibrium*, pages 28–66. Springer, 2017.

Pierre Degond and Sébastien Motsch.

**Continuum limit of self-driven particles with orientation interaction.**

*Mathematical Models and Methods in Applied Sciences*, 18(supp01):1193–1215, 2008.

Steven Gold, Anand Rangarajan, Chien-Ping Lu, Suguna Pappu, and Eric Mjolsness.

**New algorithms for 2d and 3d point matching : Pose estimation and correspondence.**

*Pattern recognition*, 31(8):1019–1031, 1998.

Leonid V Kantorovich.

**On the translocation of masses.**

In *Dokl. Akad. Nauk. USSR (NS)*, volume 37, pages 199–201, 1942.

Harold W Kuhn.

**The Hungarian method for the assignment problem.**

*Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Jeffrey J Kosowsky and Alan L Yuille.

**The invisible hand algorithm : Solving the assignment problem with statistical physics.**

*Neural networks*, 7(3):477–490, 1994.

Florent Leclercq.

**Bayesian optimization for likelihood-free cosmological inference.**

*Physical Review D*, 98(6):063511, 2018.

Bruno Lévy.

**A numerical algorithm for l2 semi-discrete optimal transport in 3d.**

*ESAIM : Mathematical Modelling and Numerical Analysis*, 49(6):1693–1715, 2015.

Christian Ledig, Andreas Schuh, Ricardo Guerrero, Rolf A Heckemann, and Daniel Rueckert.

**Structural brain imaging in Alzheimer's disease and mild cognitive impairment : biomarker analysis and shared morphometry database.**

*Scientific reports*, 8(1):11258, 2018.

Quentin Mérigot.

**A multiscale approach to optimal transport.**

In *Computer Graphics Forum*, volume 30, pages 1583–1592. Wiley Online Library, 2011.

Bernhard Schmitzer.

**Stabilized sparse scaling algorithms for entropy regularized transport problems.**

*SIAM Journal on Scientific Computing*, 41(3):A1443–A1481, 2019.

Freyr Sverrisson, Jean Feydy, Bruno E. Correia, and Michael M. Bronstein.

**Fast end-to-end learning on protein surfaces.**

*bioRxiv*, 2020.