Sorting points in dimension $D > 1$

Jean Feydy

Twitter London – February 2020

Imperial College London
Collaboration with B. Charlier, J. Glaunès (KeOps library);
F.-X. Vialard, G. Peyré, T. Séjourné, A. Trouvé (OT theory).

Who am I?

Quick CV:

- Typical French curriculum (ENS Maths + MVA + Siemens)

Who am I?

Quick CV:

- Typical French curriculum (ENS Maths + MVA + Siemens)
- 2016-2019: PhD with Alain Trouvé (ENS Paris-Saclay)

Who am I?

Quick CV:

- Typical French curriculum (ENS Maths + MVA + Siemens)
- 2016-2019: PhD with Alain Trouvé (ENS Paris-Saclay)
- 2019-2022: PostDoc at Imperial with Michael

Who am I?

Quick CV:

- Typical French curriculum (ENS Maths + MVA + Siemens)
- 2016-2019: PhD with Alain Trouvé (ENS Paris-Saclay)
- 2019-2022: PostDoc at Imperial with Michael

Who am I?

Quick CV:

- Typical French curriculum (ENS Maths + MVA + Siemens)
- 2016-2019: PhD with Alain Trouvé (ENS Paris-Saclay)
- 2019-2022: PostDoc at Imperial with Michael

$\Longrightarrow$ **Geometry**, mostly for medical imaging

The medical imaging pipeline [Ptr19, EPW+11]

Sensor data

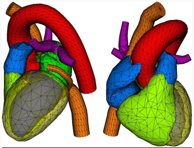
The medical imaging pipeline [Ptr19, EPW+11]

Raw image

Signal processing

Sensor data

The medical imaging pipeline [Ptr19, EPW^{+}11]



High-level description

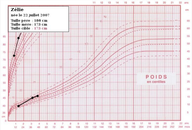
Computational anatomy

Raw image

Signal processing

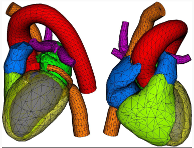
Sensor data

The medical imaging pipeline [Ptr19, EPW+11]



Valuable information

Statistics



High-level description

Computational anatomy

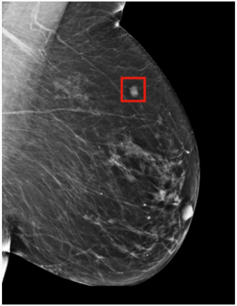
Raw image

Signal processing

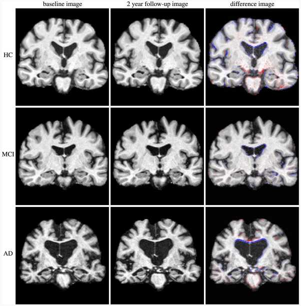
Sensor data

Computational anatomy [CSG19, LSG+18, CMN14]

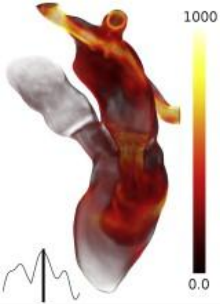
Three main problems:



Spot patterns

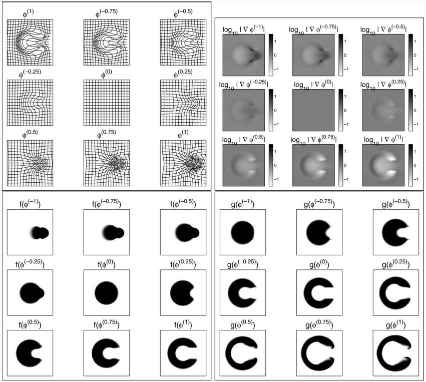


Analyze variations



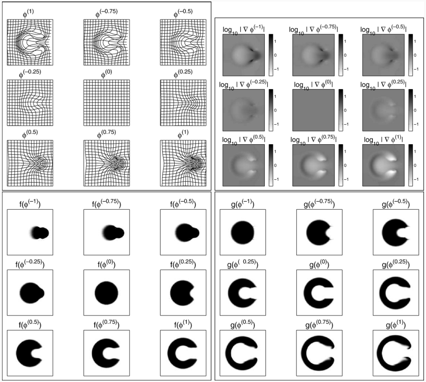
Fit models

Shape analysis [Ash07, Gla05]



Advection $\neq$ Convolution

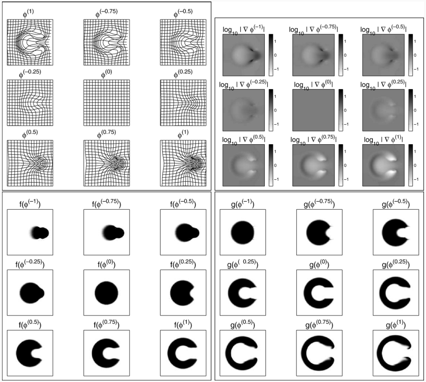
Shape analysis [Ash07, Gla05]

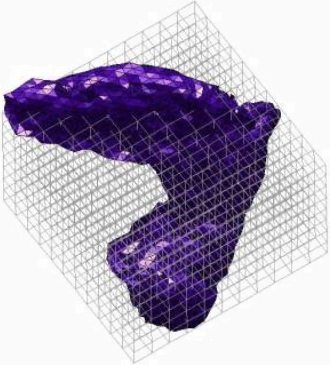


Advection $\neq$ Convolution

Mesh deformation

Shape analysis [Ash07, Gla05]

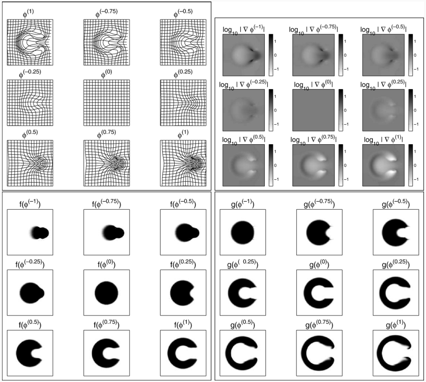




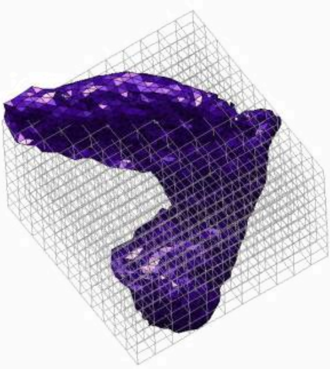
Advection $\neq$ Convolution

Mesh deformation

Shape analysis [Ash07, Gla05]

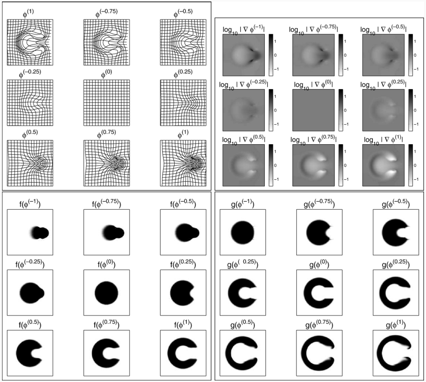


Advection $\neq$ Convolution

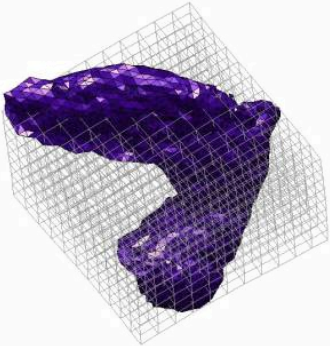


Mesh deformation

Shape analysis [Ash07, Gla05]

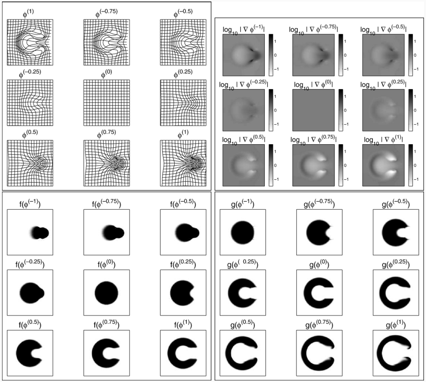


Advection $\neq$ Convolution

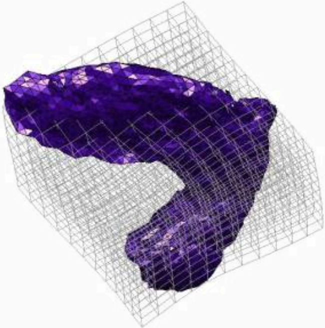


Mesh deformation

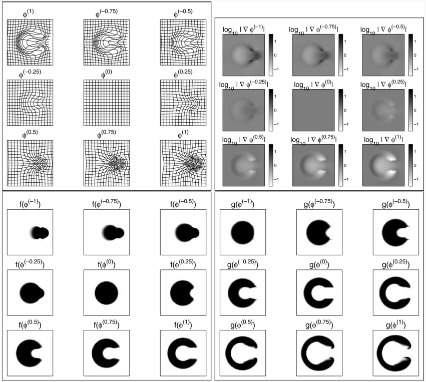
Shape analysis [Ash07, Gla05]



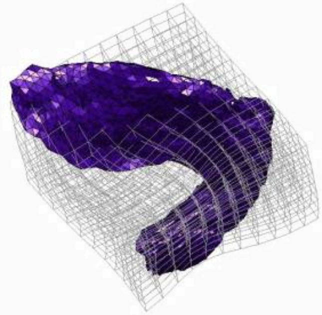
Advection $\neq$ Convolution



Mesh deformation

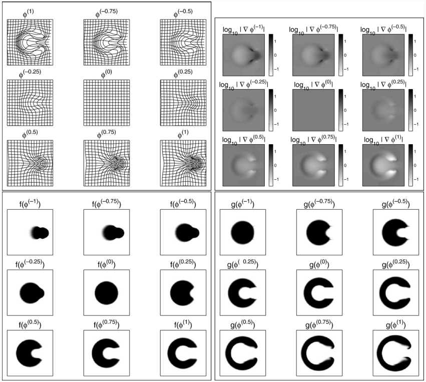


Advection $\neq$ Convolution

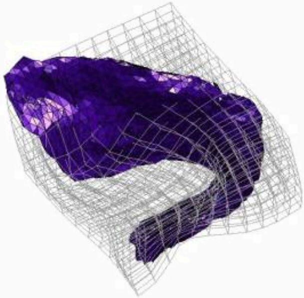


Mesh deformation

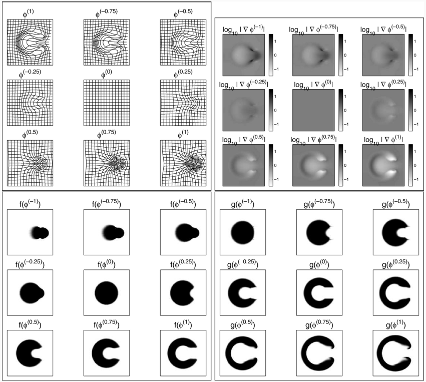
Shape analysis [Ash07, Gla05]



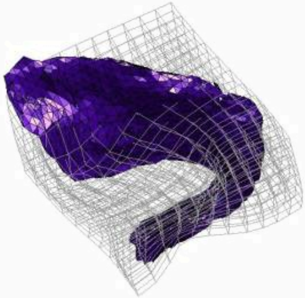
Advection ≠ Convolution



Mesh deformation

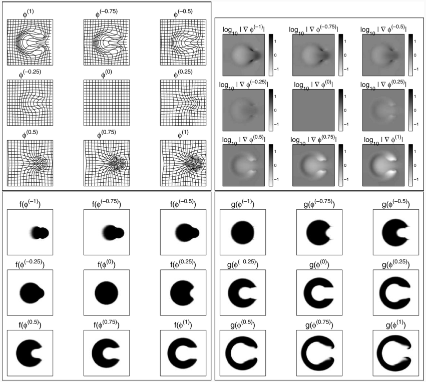


Advection $\neq$ Convolution

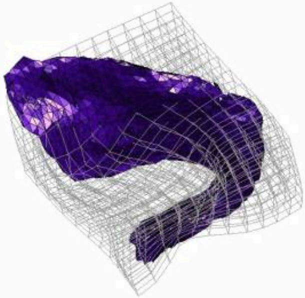


Mesh deformation

$\Longrightarrow$ We need **fast geometric primitives.**



Advection $\neq$ Convolution



Mesh deformation

$\Longrightarrow$ We need **fast geometric primitives.**

Problem: not supported well by TensorFlow and PyTorch.

My work so far:

Geometric data analysis, beyond convolutions

My work so far:

- **Efficient** GPU routines for point clouds, kernels, etc.
  $\longrightarrow$ KeOps extension for PyTorch, NumPy, Matlab, R.

My work so far:

- **Efficient** GPU routines for point clouds, kernels, etc.
  $\longrightarrow$ KeOps extension for PyTorch, NumPy, Matlab, R.

- **Robust** deformation architectures
  $\longrightarrow$ Diffeomorphisms, elastic meshes, etc.

Geometric data analysis, beyond convolutions

My work so far:

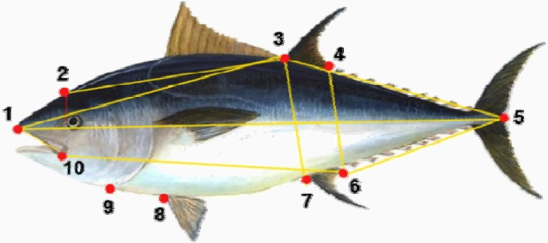
- **Efficient** GPU routines for point clouds, kernels, etc.
  $\longrightarrow$ KeOps extension for PyTorch, NumPy, Matlab, R.

- **Robust** deformation architectures
  $\longrightarrow$ Diffeomorphisms, elastic meshes, etc.

- **Geometric** loss functions
  $\longrightarrow$ Wasserstein distance = optimal transport = **sorting**.

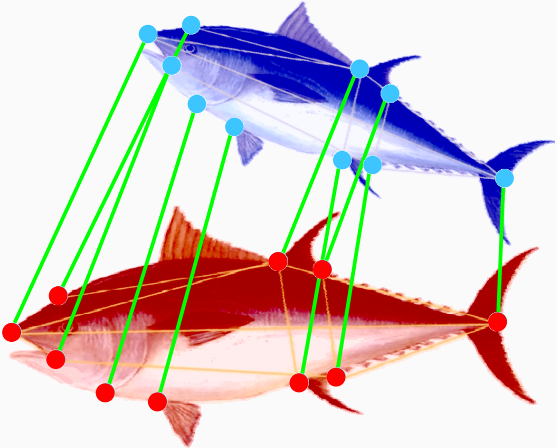
Working with unlabeled point clouds

Life is easy when you have labels



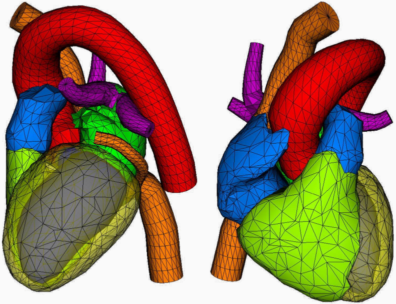
Anatomical landmarks from *A morphometric approach for the analysis of body shape in bluefin tuna*, Addis et al., 2009.

Life is easy when you have labels

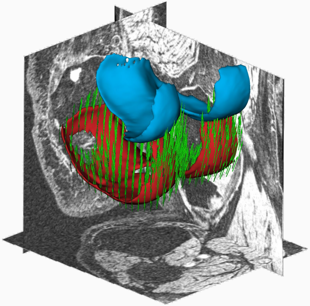


Anatomical landmarks from *A morphometric approach for the analysis of body shape in bluefin tuna*, Addis et al., 2009.

Unfortunately, medical data is often unlabeled [EPW+11]



Surface meshes



Segmentation masks

Encoding unlabeled shapes as measures

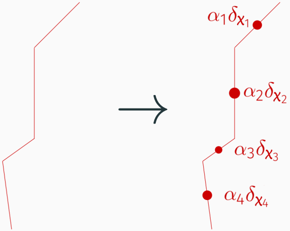
Let's enforce sampling invariance:

$$A \longrightarrow \alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i}\,, \qquad B \longrightarrow \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j}\,.$$

Encoding unlabeled shapes as measures

Let's enforce sampling invariance:

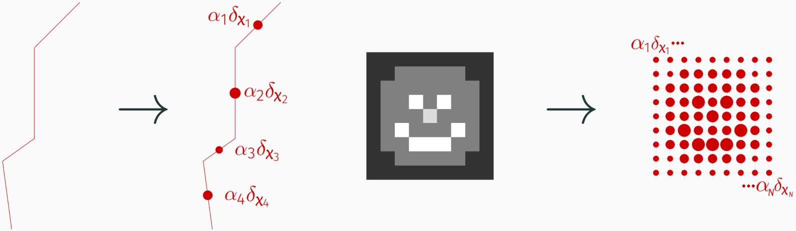
$$A \longrightarrow \alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i}, \qquad B \longrightarrow \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j}.$$



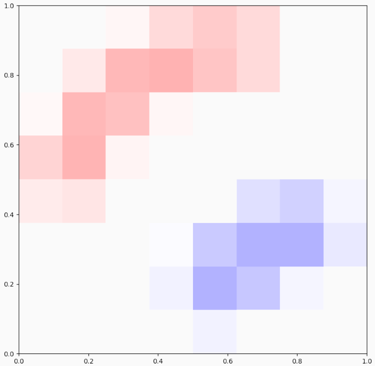
Encoding unlabeled shapes as measures

Let's enforce sampling invariance:

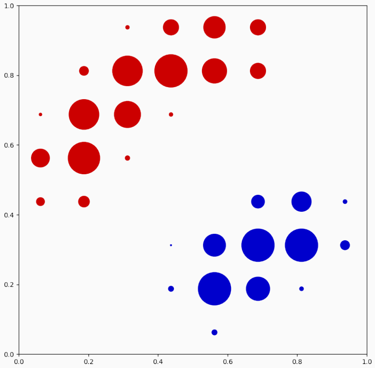
$$A \longrightarrow \alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i}\,, \qquad B \longrightarrow \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j}\,.$$



A baseline setting: density registration

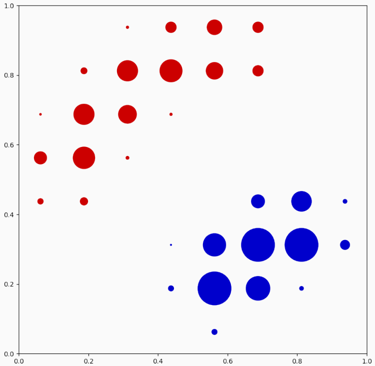


A baseline setting: density registration



$$\alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i}\,, \quad \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j}\,.$$

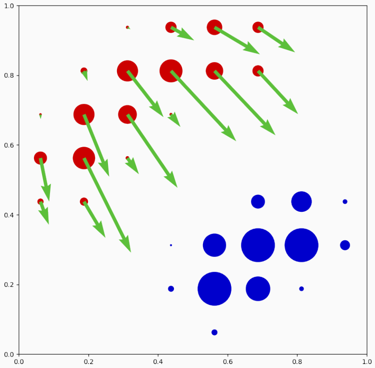
A baseline setting: density registration



$$\alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i}\,, \quad \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j}\,.$$

$$\sum_{i=1}^{N} \alpha_i = 1 = \sum_{j=1}^{M} \beta_j$$

A baseline setting: density registration

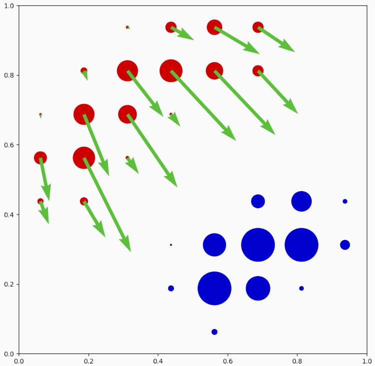


$$\alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i}\,, \quad \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j}\,.$$

$$\sum_{i=1}^{N} \alpha_i \;=\; 1 \;=\; \sum_{j=1}^{M} \beta_j$$

Display $v_i \;=\; -\frac{1}{\alpha_i} \nabla_{x_i} \text{Loss}(\alpha, \beta)$.

A baseline setting: density registration



$$\alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i}\,, \quad \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j}\,.$$

$$\sum_{i=1}^{N} \alpha_i \;=\; 1 \;=\; \sum_{j=1}^{M} \beta_j$$

Display $v_i \;=\; -\frac{1}{\alpha_i} \nabla_{x_i} \text{Loss}(\alpha, \beta)$.

Seamless extensions to:

- $\sum_i \alpha_i \neq \sum_j \beta_j$, outliers [CPSV18],
- curves and surfaces [KCC17],
- variable weights α_i.

Simple loss functions between measures

- **Chamfer** distance $\simeq$ soft-**Hausdorff**:
  Projection-based $\longrightarrow$ Degenerate gradients.

Simple loss functions between measures

- **Chamfer** distance $\simeq$ soft-**Hausdorff**:
  Projection-based $\longrightarrow$ Degenerate gradients.

- Kernel distance $\simeq$ Blurred L^2 norm, convolution-based:

  $$\text{Loss}(\alpha, \beta) \;=\; \tfrac{1}{2}\|g \star (\alpha - \beta)\|^2_{L^2(\mathbb{R}^D)} \;=\; \tfrac{1}{2}\langle\, \alpha - \beta\,,\, k \star (\alpha - \beta)\,\rangle$$

  where $k \;=\; (g \circ (x \mapsto -x)) \,\star\, g\,.$

Simple loss functions between measures

- **Chamfer** distance $\simeq$ soft-**Hausdorff**:
  Projection-based $\longrightarrow$ Degenerate gradients.

- Kernel distance $\simeq$ Blurred L^2 norm, convolution-based:

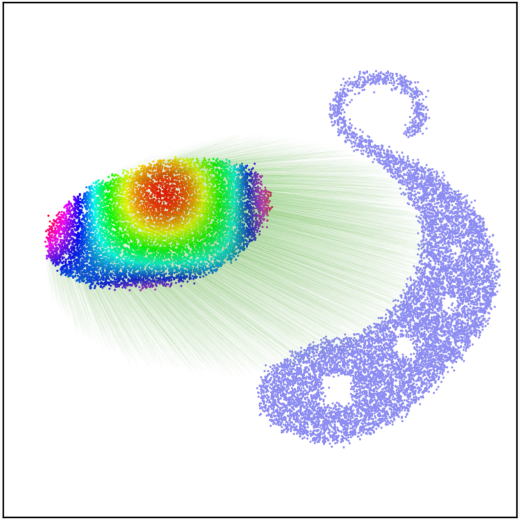
  $$\text{Loss}(\alpha, \beta) \;=\; \tfrac{1}{2}\|g \star (\alpha - \beta)\|^2_{L^2(\mathbb{R}^\mathrm{D})} \;=\; \tfrac{1}{2}\langle\, \alpha - \beta \,,\, k \star (\alpha - \beta)\,\rangle$$

  where $k \;=\; (g \circ (x \mapsto -x)) \,\star\, g\,.$

- Example: the Energy Distance, $k(x, y) = -\|x - y\|$:

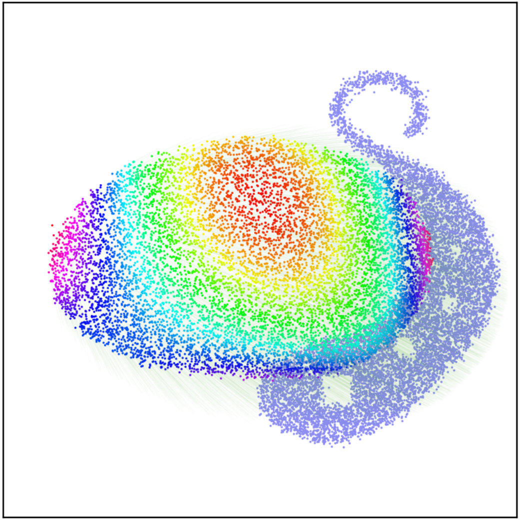
  $$\text{Loss}(\alpha, \beta) \;=\; \sum_i \sum_j \alpha_i \beta_j \,\|x_i - y_j\| \;-\; \tfrac{1}{2}\sum_i \sum_j \alpha_i \alpha_j \,\|x_i - x_j\| \;-\; \tfrac{1}{2}\sum_i \sum_j \beta_i \beta_j \,\|y_i - y_j\|\,.$$

Gradient flow as a toy registration problem



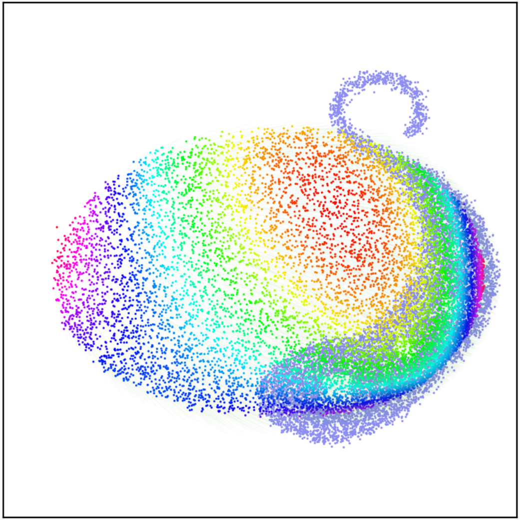
$t = .00$

Gradient flow as a toy registration problem



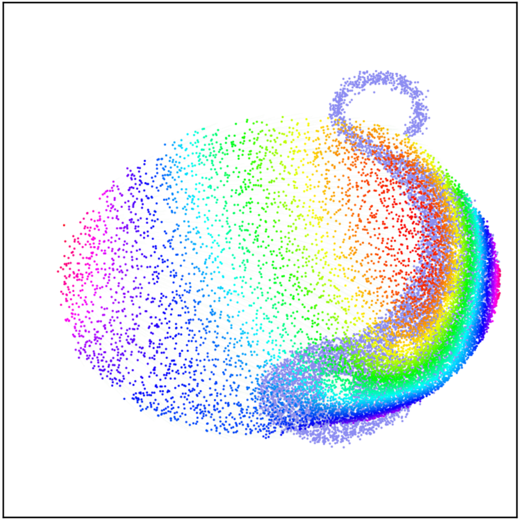
$t = .25$

Gradient flow as a toy registration problem



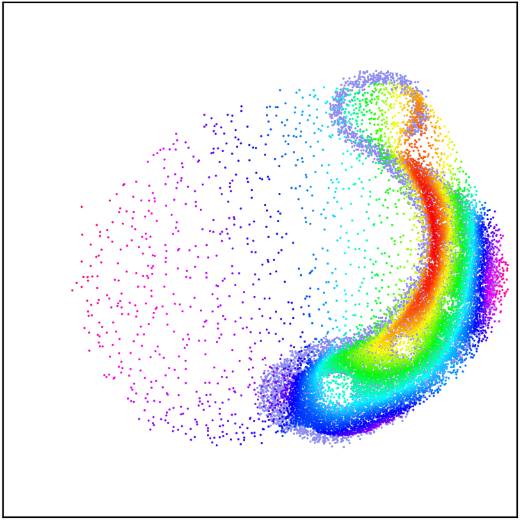
$t = .50$

Gradient flow as a toy registration problem



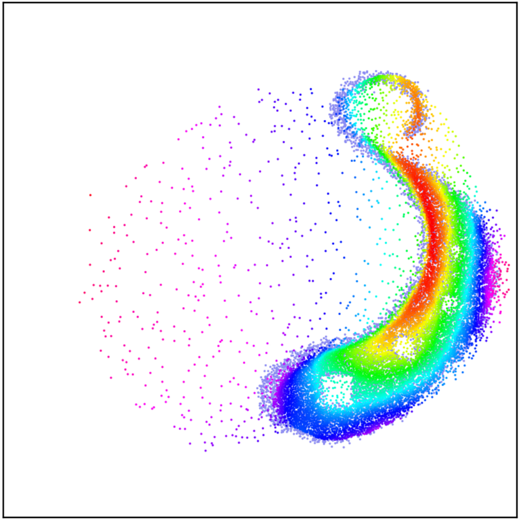
$t = 1.00$

Gradient flow as a toy registration problem



$t = 5.00$

Gradient flow as a toy registration problem



$t = 10.00$

The Wasserstein distance

We need **clean gradients**, without artifacts.

We need **clean gradients**, without artifacts. Let's **sort** our points.

The Wasserstein distance

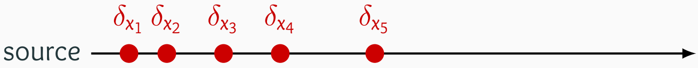
We need **clean gradients**, without artifacts. Let's **sort** our points.

Simple toy example in 1D :

The Wasserstein distance

We need **clean gradients**, without artifacts. Let's **sort** our points.

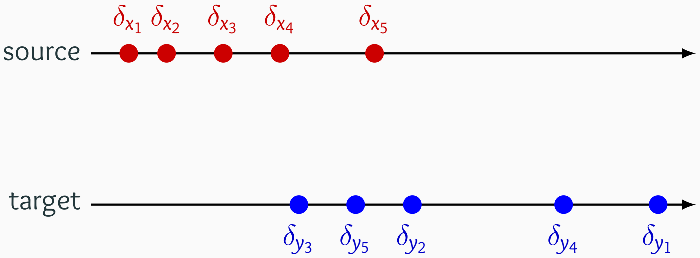
Simple toy example in 1D :



The Wasserstein distance

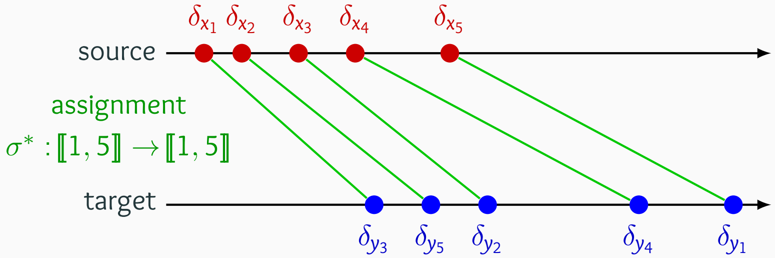
We need **clean gradients**, without artifacts. Let's **sort** our points.

Simple toy example in 1D :



We need **clean gradients**, without artifacts. Let's **sort** our points.

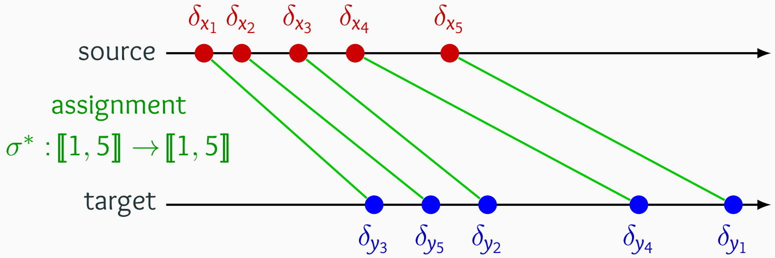
Simple toy example in 1D :



$$\text{OT}(\alpha, \beta) \;=\; \frac{1}{2N} \sum_{i=1}^{N} |x_i - y_{\sigma^*(i)}|^2$$

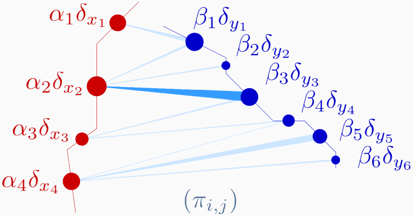
We need **clean gradients**, without artifacts. Let's **sort** our points.

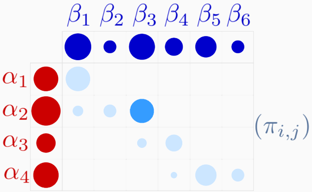
Simple toy example in 1D :



$$\mathrm{OT}(\alpha, \beta) \;=\; \frac{1}{2N} \sum_{i=1}^{N} \left| x_i - y_{\sigma^*(i)} \right|^2 \;=\; \min_{\sigma \in \mathcal{S}_N} \frac{1}{2N} \sum_{i=1}^{N} \left| x_i - y_{\sigma(i)} \right|^2$$

Optimal transport generalizes sorting to D > 1





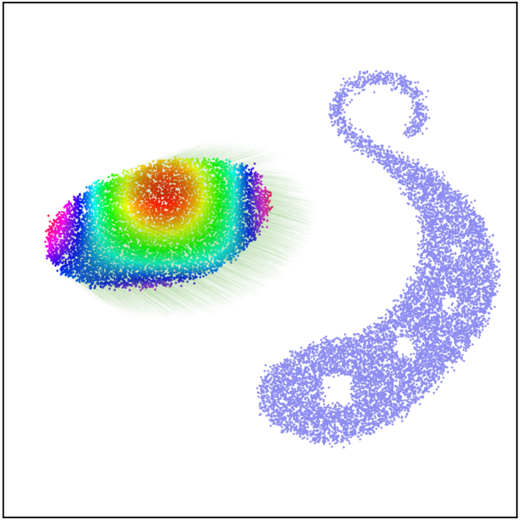
Minimize over *N*-by-*M* matrices (transport plans) π :

$$\text{OT}(\alpha, \beta) = \min_{\pi} \underbrace{\sum_{i,j} \pi_{i,j} \cdot \tfrac{1}{2}|x_i - y_j|^2}_{\text{transport cost}}$$

subject to $\quad \pi_{i,j} \geqslant 0,$

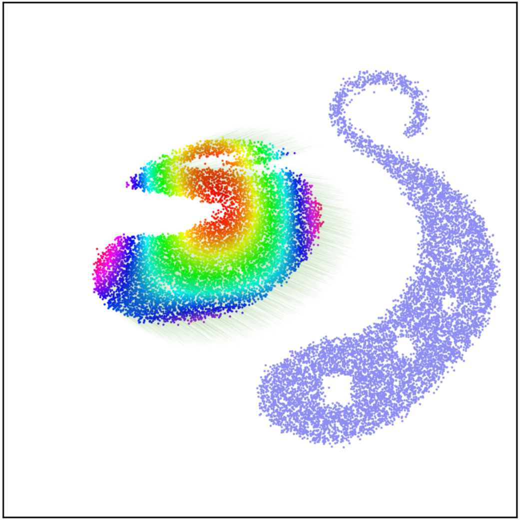
$$\sum_j \pi_{i,j} = \alpha_i, \quad \sum_i \pi_{i,j} = \beta_j.$$

Wasserstein gradients are homogeneous



$t = .00$

Wasserstein gradients are homogeneous

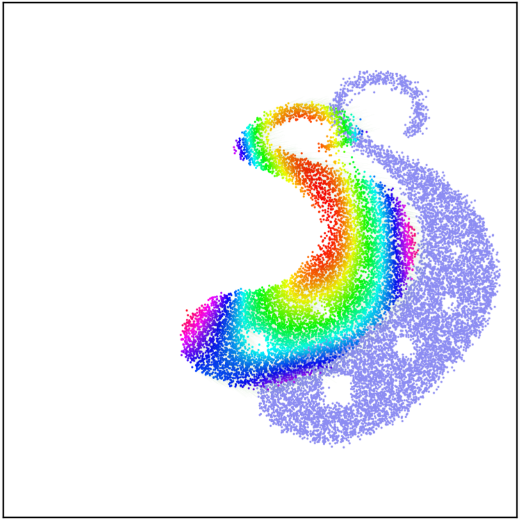


$t = .25$

Wasserstein gradients are homogeneous

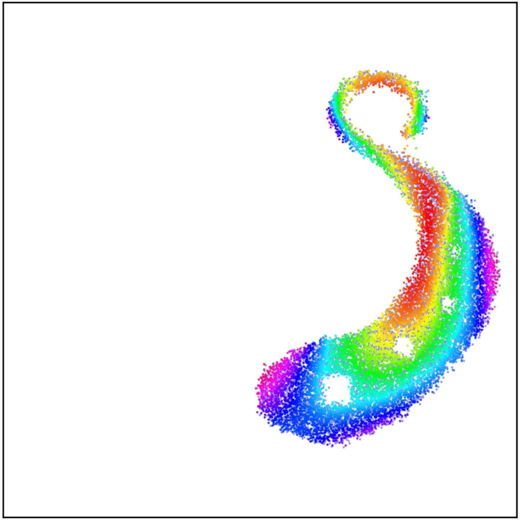
$t = .50$

Wasserstein gradients are homogeneous



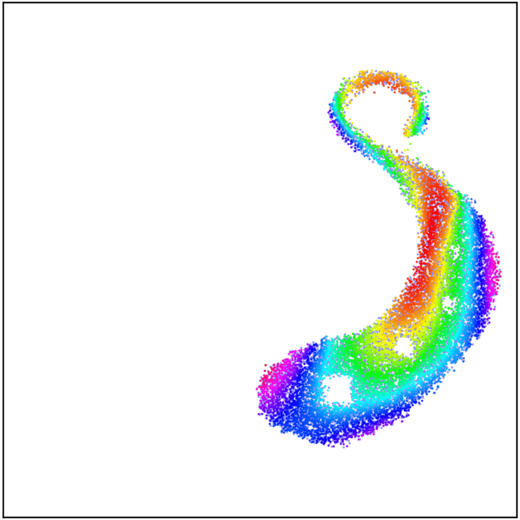
$t = 1.00$

Wasserstein gradients are homogeneous



$t = 5.00$

Wasserstein gradients are homogeneous



$t = \texttt{10.00}$

Key properties [Bre91]

The Wasserstein loss $\mathrm{OT}(\alpha, \beta)$ is:

- **Symmetric:** $\mathrm{OT}(\alpha, \beta) = \mathrm{OT}(\beta, \alpha)\,.$

Key properties [Bre91]

The Wasserstein loss $\mathrm{OT}(\alpha, \beta)$ is:

- **Symmetric:** $$\mathrm{OT}(\alpha, \beta) = \mathrm{OT}(\beta, \alpha)\,.$$
- **Positive:** $$\mathrm{OT}(\alpha, \beta) \geqslant 0\,.$$

Key properties [Bre91]

The Wasserstein loss $\text{OT}(\alpha, \beta)$ is:

- **Symmetric:** $$\text{OT}(\alpha, \beta) = \text{OT}(\beta, \alpha)\,.$$
- **Positive:** $$\text{OT}(\alpha, \beta) \geqslant 0\,.$$
- **Definite:** $$\text{OT}(\alpha, \beta) = 0 \iff \alpha = \beta\,.$$

Key properties [Bre91]

The Wasserstein loss $\text{OT}(\alpha, \beta)$ is:

- **Symmetric:** $$\text{OT}(\alpha, \beta) = \text{OT}(\beta, \alpha)\,.$$
- **Positive:** $$\text{OT}(\alpha, \beta) \geqslant 0\,.$$
- **Definite:** $$\text{OT}(\alpha, \beta) = 0 \iff \alpha = \beta\,.$$
- **Translation-aware:** $$\text{OT}(\alpha,\ \text{Translate}_{\vec{v}}(\alpha)\,) = \tfrac{1}{2}\|\,\vec{v}\,\|^2\,.$$

Key properties [Bre91]

The Wasserstein loss $\mathrm{OT}(\alpha, \beta)$ is:

- **Symmetric:** $\mathrm{OT}(\alpha, \beta) = \mathrm{OT}(\beta, \alpha)\,.$
- **Positive:** $\mathrm{OT}(\alpha, \beta) \geqslant 0\,.$
- **Definite:** $\mathrm{OT}(\alpha, \beta) = 0 \iff \alpha = \beta\,.$
- **Translation-aware:** $\mathrm{OT}(\alpha,\ \mathrm{Translate}_{\vec{v}}(\alpha)\,) = \frac{1}{2}\|\,\vec{v}\,\|^2\,.$

- More generally, OT retrieves the unique **gradient of a convex function** $T = \nabla\varphi$ that maps α onto β :

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

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

Key properties [Bre91]

The Wasserstein loss $\text{OT}(\alpha, \beta)$ is:

- **Symmetric:** $\text{OT}(\alpha, \beta) = \text{OT}(\beta, \alpha)\,.$
- **Positive:** $\text{OT}(\alpha, \beta) \geqslant 0\,.$
- **Definite:** $\text{OT}(\alpha, \beta) = 0 \iff \alpha = \beta\,.$
- **Translation-aware:** $\text{OT}(\alpha,\ \text{Translate}_{\vec{v}}(\alpha)\,) = \frac{1}{2}\|\,\vec{v}\,\|^2\,.$

- More generally, OT retrieves the unique **gradient of a convex function** $T = \nabla\varphi$ that maps α onto β :

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

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

Key properties [Bre91]

The Wasserstein loss $\text{OT}(\alpha, \beta)$ is:

- **Symmetric:** $\text{OT}(\alpha, \beta) = \text{OT}(\beta, \alpha)\,.$
- **Positive:** $\text{OT}(\alpha, \beta) \geqslant 0\,.$
- **Definite:** $\text{OT}(\alpha, \beta) = 0 \iff \alpha = \beta\,.$
- **Translation-aware:** $\text{OT}(\alpha,\ \text{Translate}_{\vec{v}}(\alpha)\,) = \frac{1}{2}\|\,\vec{v}\,\|^2\,.$

- More generally, OT retrieves the unique **gradient of a convex function** $T = \nabla\varphi$ that maps α onto β :

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

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

$\Longrightarrow$ Appealing generalization of an **increasing mapping.**

Can we scale all of this?

Kantorovitch's dual formulation

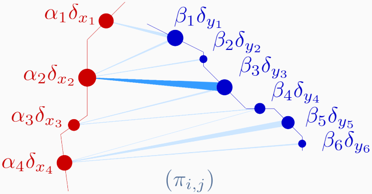
$$\mathrm{OT}(\alpha, \beta) = \min_{\pi} \langle \pi , \mathrm{C} \rangle, \text{ with } \mathrm{C}(x_i, y_j) = \frac{1}{p}\|x_i - y_j\|^p \quad \longrightarrow \text{Assignment}$$

$$\text{s.t. } \pi \geqslant 0, \quad \pi \mathbf{1} = \alpha, \quad \pi^{\intercal}\mathbf{1} = \beta$$

$$\mathrm{OT}(\alpha,\beta) = \min_{\pi} \langle \pi, C \rangle, \text{ with } C(x_i,y_j) = \frac{1}{p}\|x_i - y_j\|^p \longrightarrow \text{Assignment}$$

$$\text{s.t. } \pi \geqslant 0, \quad \pi \mathbf{1} = \alpha, \quad \pi^\top \mathbf{1} = \beta$$

$\sum_{i,j} \pi_{i,j} C(x_i, y_j)$

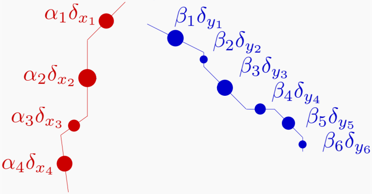


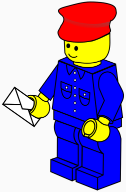
Kantorovitch's dual formulation

$$\mathrm{OT}(\alpha,\beta) = \min_{\pi} \langle \pi , \mathrm{C} \rangle, \quad \text{with } \mathrm{C}(x_i, y_j) = \frac{1}{p}\|x_i - y_j\|^p \quad \longrightarrow \text{Assignment}$$

$$\text{s.t. } \pi \geqslant 0, \quad \pi \mathbf{1} = \alpha, \quad \pi^\top \mathbf{1} = \beta$$

$\sum_{i,j} \pi_{i,j} \, \mathrm{C}(x_i, y_j)$



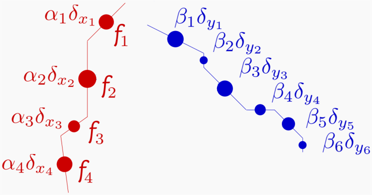


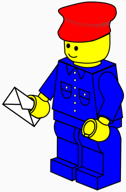
Kantorovitch's dual formulation

$$\mathrm{OT}(\alpha,\beta) = \min_{\pi} \langle \pi , \mathrm{C} \rangle, \text{ with } \mathrm{C}(x_i,y_j) = \frac{1}{p}\|x_i - y_j\|^p \quad \longrightarrow \text{ Assignment}$$

$$\text{s.t. } \pi \geqslant 0, \quad \pi\mathbf{1} = \alpha, \quad \pi^\top \mathbf{1} = \beta$$

$\sum_{i,j} \pi_{i,j}\, \mathrm{C}(x_i,y_j)$



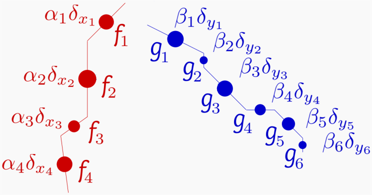


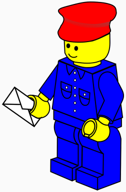
Kantorovitch's dual formulation

$$\mathrm{OT}(\alpha,\beta) = \min_{\pi} \langle \pi \,,\, \mathrm{C} \rangle, \ \text{ with } \mathrm{C}(x_i,y_j) = \frac{1}{p}\|x_i - y_j\|^p \quad \longrightarrow \text{Assignment}$$

$$\text{s.t. } \pi \geqslant 0, \quad \pi\,\mathbf{1} = \alpha, \quad \pi^{\mathsf{T}}\mathbf{1} = \beta$$

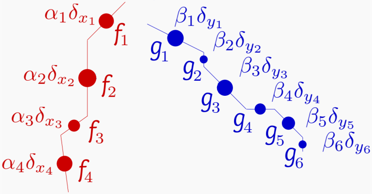
$\sum_{i,j} \pi_{i,j}\, \mathrm{C}(x_i,y_j)$

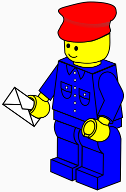




$$\mathrm{OT}(\alpha,\beta) = \min_{\pi} \langle \pi , \mathrm{C} \rangle, \ \text{with } \mathrm{C}(x_i,y_j) = \tfrac{1}{p}\|x_i - y_j\|^p \longrightarrow \text{Assignment}$$

$$\text{s.t. } \pi \geqslant 0, \quad \pi \mathbf{1} = \alpha, \quad \pi^\intercal \mathbf{1} = \beta$$





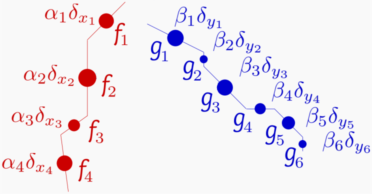
$\sum_{i,j} \pi_{i,j} \, \mathrm{C}(x_i,y_j)$

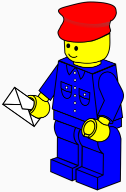
$\sum_i \alpha_i f_i + \sum_j \beta_j g_j$

Kantorovitch's dual formulation

$$\mathrm{OT}(\alpha,\beta) = \min_{\pi} \langle \pi , \mathrm{C} \rangle, \text{ with } \mathrm{C}(x_i, y_j) = \frac{1}{p}\|x_i - y_j\|^p \longrightarrow \text{Assignment}$$

$$\text{s.t. } \pi \geqslant 0, \quad \pi \mathbf{1} = \alpha, \quad \pi^\intercal \mathbf{1} = \beta$$





$\sum_{i,j} \pi_{i,j} \mathrm{C}(x_i, y_j)$

$\sum_i \alpha_i f_i + \sum_j \beta_j g_j$

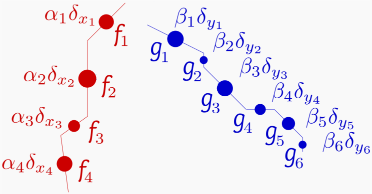
$$\max_{f,g} \quad \langle \alpha , f \rangle + \langle \beta , g \rangle \qquad \longrightarrow \text{FedEx}$$

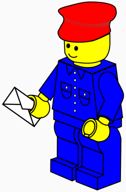
$$\text{s.t.} \quad f(x_i) + g(y_j) \leqslant \mathrm{C}(x_i, y_j),$$

Kantorovitch's dual formulation

$$\mathrm{OT}(\alpha,\beta) = \min_{\pi} \langle \pi , \mathrm{C} \rangle, \quad \text{with } \mathrm{C}(x_i,y_j) = \tfrac{1}{p}\|x_i - y_j\|^p \quad \longrightarrow \text{Assignment}$$

$$\text{s.t. } \pi \geqslant 0, \quad \pi\mathbf{1} = \alpha, \quad \pi^\top\mathbf{1} = \beta$$





$\sum_{i,j} \pi_{i,j}\, \mathrm{C}(x_i,y_j)$ $\qquad$ $\sum_i \alpha_i f_i + \sum_j \beta_j g_j$

$$= \max_{f,g} \quad \langle \alpha , f \rangle + \langle \beta , g \rangle \quad \longrightarrow \text{FedEx}$$

$$\text{s.t.} \quad f(x_i) + g(y_j) \leqslant \mathrm{C}(x_i,y_j),$$

How should we solve the OT problem?

Key dates:

- [Kan42]: **Dual** problem, $O(N^2) \to O(N)$ memory footprint.

How should we solve the OT problem?

Key dates:

- [Kan42]: **Dual** problem, $O(N^2) \rightarrow O(N)$ memory footprint.
- [Kuh55]: **Hungarian** method in $O(N^3)$.

How should we solve the OT problem?

Key dates:

- [Kan42]: **Dual** problem, $O(N^2) \rightarrow O(N)$ memory footprint.
- [Kuh55]: **Hungarian** method in $O(N^3)$.
- [Ber79]: **Auction** algorithm in $O(N^2)$.

How should we solve the OT problem?

Key dates:

- [Kan42]: **Dual** problem, $O(N^2) \rightarrow O(N)$ memory footprint.
- [Kuh55]: **Hungarian** method in $O(N^3)$.
- [Ber79]: **Auction** algorithm in $O(N^2)$.
- [KY94]: **SoftAssign** = Sinkhorn + annealing, in $O(N^2)$.

How should we solve the OT problem?

Key dates:

- [Kan42]: **Dual** problem, $O(N^2) \rightarrow O(N)$ memory footprint.
- [Kuh55]: **Hungarian** method in $O(N^3)$.
- [Ber79]: **Auction** algorithm in $O(N^2)$.
- [KY94]: **SoftAssign** = Sinkhorn + annealing, in $O(N^2)$.
- [GRL$^+$98, CR00]: **Robust Point Matching** = Sinkhorn as a loss.

How should we solve the OT problem?

Key dates:

- [Kan42]: **Dual** problem, $O(N^2) \rightarrow O(N)$ memory footprint.
- [Kuh55]: **Hungarian** method in $O(N^3)$.
- [Ber79]: **Auction** algorithm in $O(N^2)$.
- [KY94]: **SoftAssign** = Sinkhorn + annealing, in $O(N^2)$.
- [GRL+98, CR00]: **Robust Point Matching** = Sinkhorn as a loss.
- [Cut13]: Start of the **GPU era**.

How should we solve the OT problem?

Key dates:

- [Kan42]: **Dual** problem, $O(N^2) \to O(N)$ memory footprint.
- [Kuh55]: **Hungarian** method in $O(N^3)$.
- [Ber79]: **Auction** algorithm in $O(N^2)$.
- [KY94]: **SoftAssign** = Sinkhorn + annealing, in $O(N^2)$.
- [GRL$^+$98, CR00]: **Robust Point Matching** = Sinkhorn as a loss.
- [Cut13]: Start of the **GPU era**.
- [Mér11, Lév15, Sch19]: **Multiscale** CPU solvers in $O(N \log N)$.

How should we solve the OT problem?

Key dates:

- [Kan42]: **Dual** problem, $O(N^2) \rightarrow O(N)$ memory footprint.
- [Kuh55]: **Hungarian** method in $O(N^3)$.
- [Ber79]: **Auction** algorithm in $O(N^2)$.
- [KY94]: **SoftAssign** = Sinkhorn + annealing, in $O(N^2)$.
- [GRL$^+$98, CR00]: **Robust Point Matching** = Sinkhorn as a loss.
- [Cut13]: Start of the **GPU era**.
- [Mér11, Lév15, Sch19]: **Multiscale** CPU solvers in $O(N \log N)$.
- Today: **Multiscale Sinkhorn algorithm, on the GPU.**

How should we solve the OT problem?

Key dates:

- [Kan42]: **Dual** problem, $O(N^2) \rightarrow O(N)$ memory footprint.
- [Kuh55]: **Hungarian** method in $O(N^3)$.
- [Ber79]: **Auction** algorithm in $O(N^2)$.
- [KY94]: **SoftAssign** = Sinkhorn + annealing, in $O(N^2)$.
- [GRL$^+$98, CR00]: **Robust Point Matching** = Sinkhorn as a loss.
- [Cut13]: Start of the **GPU era**.
- [Mér11, Lév15, Sch19]: **Multiscale** CPU solvers in $O(N \log N)$.
- Today: **Multiscale Sinkhorn algorithm, on the GPU**.

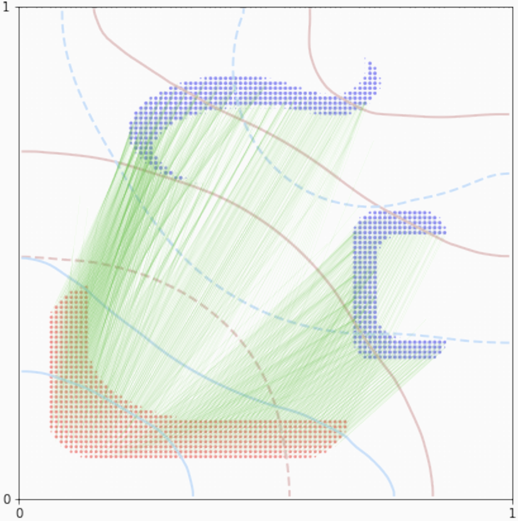
How should we solve the OT problem?

Key dates:

- [Kan42]: **Dual** problem, $O(N^2) \rightarrow O(N)$ memory footprint.
- [Kuh55]: **Hungarian** method in $O(N^3)$.
- [Ber79]: **Auction** algorithm in $O(N^2)$.
- [KY94]: **SoftAssign** = Sinkhorn + annealing, in $O(N^2)$.
- [GRL$^+$98, CR00]: **Robust Point Matching** = Sinkhorn as a loss.
- [Cut13]: Start of the **GPU era**.
- [Mér11, Lév15, Sch19]: **Multiscale** CPU solvers in $O(N \log N)$.
- Today: **Multiscale Sinkhorn algorithm, on the GPU.**

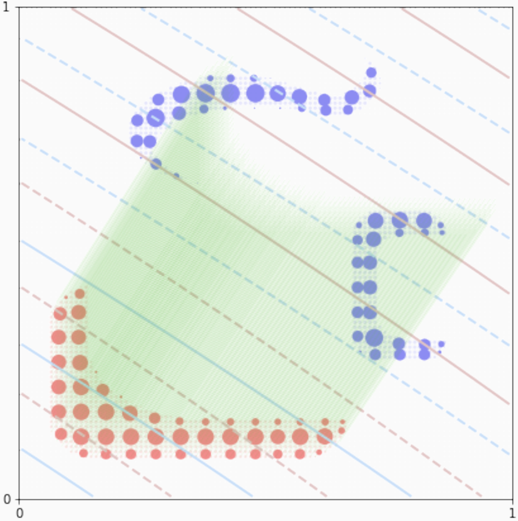
$\Longrightarrow$ Generalized **QuickSort** algorithm.

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha, \beta)$



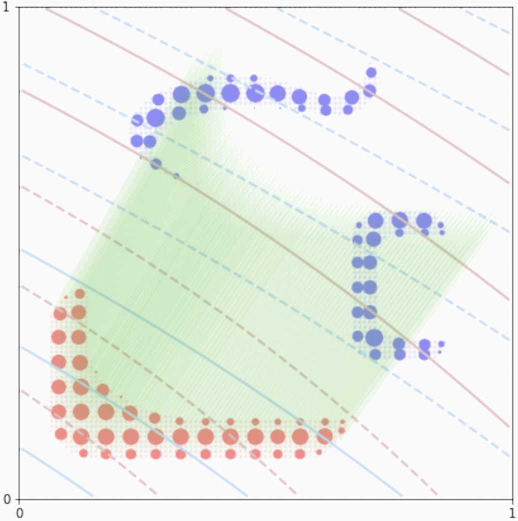
OT plan in 2D.

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha, \beta)$



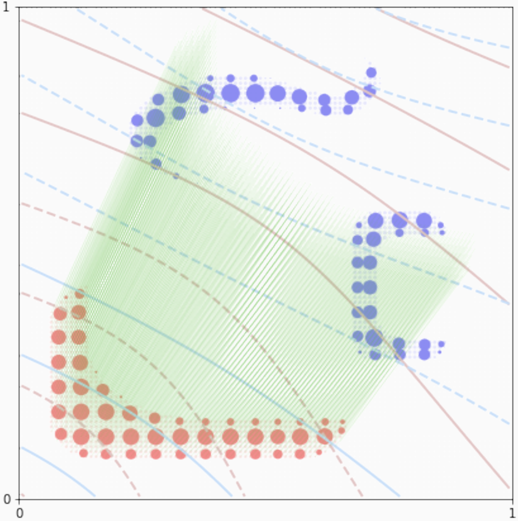
Iteration 0, blur = 2^0

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha, \beta)$



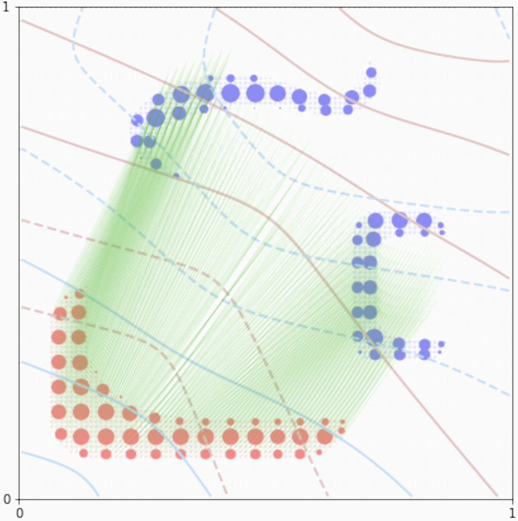
Iteration 1, blur = 2^{-1}

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha, \beta)$



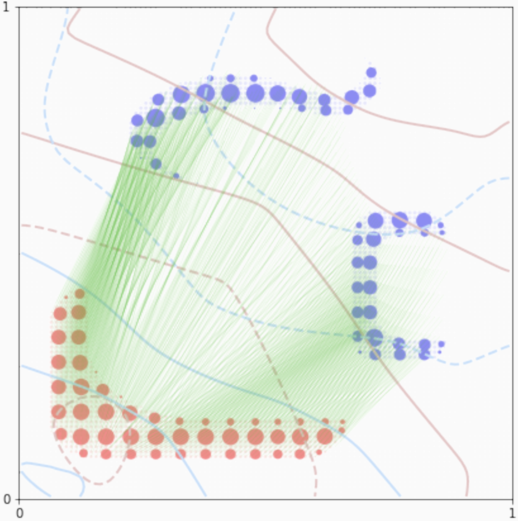
Iteration 2, blur = 2^{-2}

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha, \beta)$



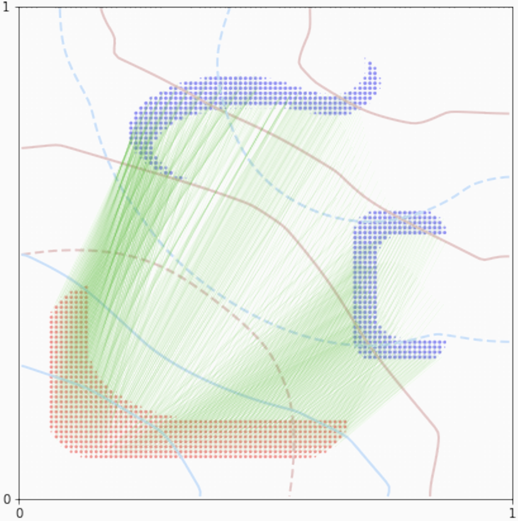
Iteration 3, blur = 2^{-3}

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha,\beta)$



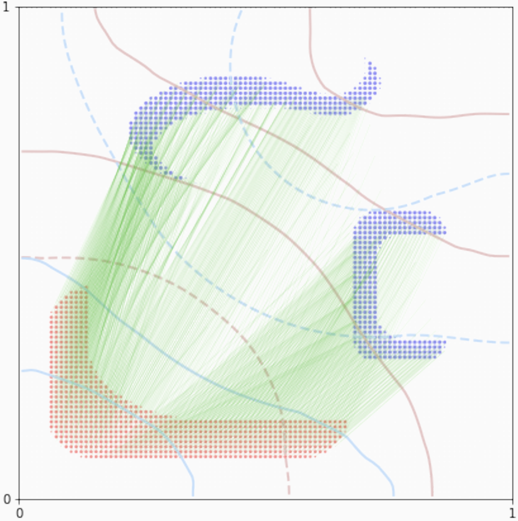
Iteration 4, blur = 2^{-4}

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha, \beta)$



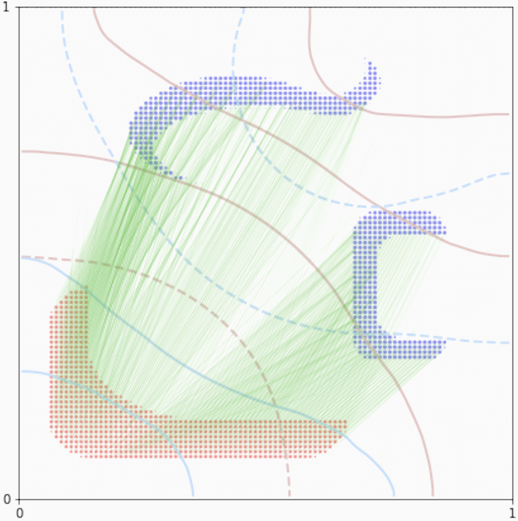
Iteration 5, blur = 2^{-5}

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha, \beta)$



Iteration 6, blur = 2^{-6}

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha, \beta)$



Iteration 7, blur = `.01`

Overview of the last decade

Progresses since [KY94, Cut13]:

+ $O(N \log N)$ instead of $O(N^2)$.

Overview of the last decade

Progresses since [KY94, Cut13]:

+ $O(N \log N)$ instead of $O(N^2)$.
+ Converge in $\leqslant 10$ iterations.

Overview of the last decade

Progresses since [KY94, Cut13]:

+ $O(N \log N)$ instead of $O(N^2)$.
+ Converge in $\leqslant 10$ iterations.
+ Fully symmetric.

Overview of the last decade

Progresses since [KY94, Cut13]:

+ $O(N \log N)$ instead of $O(N^2)$.
+ Converge in $\leqslant 10$ iterations.
+ Fully symmetric.
+ **Positive and definite** Sinkhorn loss [FSV+18].

Overview of the last decade

Progresses since [KY94, Cut13]:

+ $O(N \log N)$ instead of $O(N^2)$.
+ Converge in $\leqslant 10$ iterations.
+ Fully symmetric.
+ **Positive and definite** Sinkhorn loss [FSV+18].

− Much harder to implement, esp. on the GPU.

Overview of the last decade

Progresses since [KY94, Cut13]:

+ $O(N \log N)$ instead of $O(N^2)$.
+ Converge in $\leqslant 10$ iterations.
+ Fully symmetric.
+ **Positive and definite** Sinkhorn loss [FSV$^+$18].

− Much harder to implement, esp. on the GPU.

Overview of the last decade

Progresses since [KY94, Cut13]:

+ $O(N \log N)$ instead of $O(N^2)$.
+ Converge in $\leqslant 10$ iterations.
+ Fully symmetric.
+ **Positive and definite** Sinkhorn loss [FSV$^+$18].

– Much harder to implement, esp. on the GPU.

$\Longrightarrow$ Use the **KeOps** and **GeomLoss** libraries! $\Longleftarrow$

Scaling up to millions of samples on the GPU

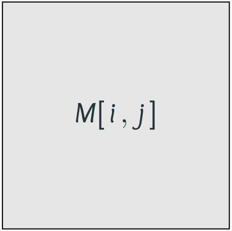
Introducing KeOps LazyTensors

$M[\,i\,,\,j\,]$

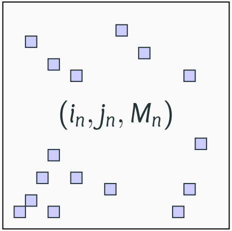
Array

Coefficients only

Introducing KeOps LazyTensors

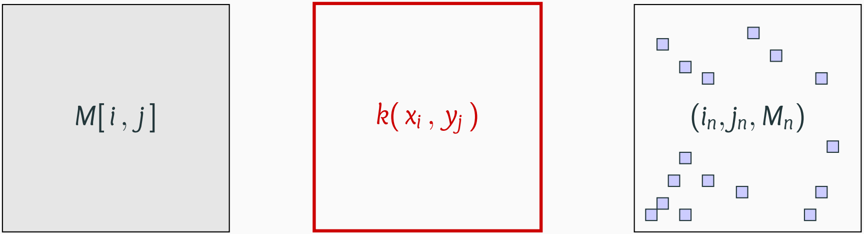


Array
Coefficients only



Sparse matrix
Coordinates + coeffs

Introducing KeOps LazyTensors

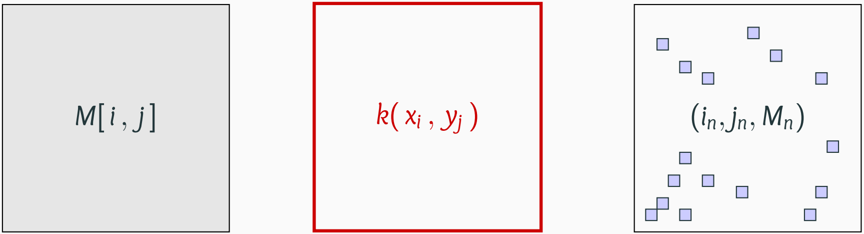


Array
Coefficients only

Symbolic LazyTensor
Formula + point clouds

Sparse matrix
Coordinates + coeffs

Introducing KeOps LazyTensors



Array
Coefficients only

Symbolic LazyTensor
Formula + point clouds

Sparse matrix
Coordinates + coeffs

```
pip install pykeops
```

It works!

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(1000000, 3, requires_grad=True).cuda()
y = torch.rand(2000000, 3).cuda()
```

It works!

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(1000000, 3, requires_grad=True).cuda()
y = torch.rand(2000000, 3).cuda()

# Turn our Tensors into KeOps symbolic variables:
from pykeops.torch import LazyTensor
x_i = LazyTensor(x[:,None,:])  # x_i.shape = (1e6, 1, 3)
y_j = LazyTensor(y[None,:,:])  # y_j.shape = ( 1, 2e6,3)
```

It works!

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(1000000, 3, requires_grad=True).cuda()
y = torch.rand(2000000, 3).cuda()

# Turn our Tensors into KeOps symbolic variables:
from pykeops.torch import LazyTensor
x_i = LazyTensor(x[:,None,:])  # x_i.shape = (1e6, 1, 3)
y_j = LazyTensor(y[None,:,:])  # y_j.shape = ( 1, 2e6,3)

# Perform large-scale computations:
D_ij = ((x_i - y_j)**2).sum(dim=2)  # (1e6,2e6,1)
```

It works!

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(1000000, 3, requires_grad=True).cuda()
y = torch.rand(2000000, 3).cuda()

# Turn our Tensors into KeOps symbolic variables:
from pykeops.torch import LazyTensor
x_i = LazyTensor(x[:,None,:])  # x_i.shape = (1e6, 1, 3)
y_j = LazyTensor(y[None,:,:])  # y_j.shape = ( 1, 2e6,3)

# Perform large-scale computations:
D_ij = ((x_i - y_j)**2).sum(dim=2)  # (1e6,2e6,1)
K_ij = (- D_ij).exp()               # (1e6,2e6,1)
```

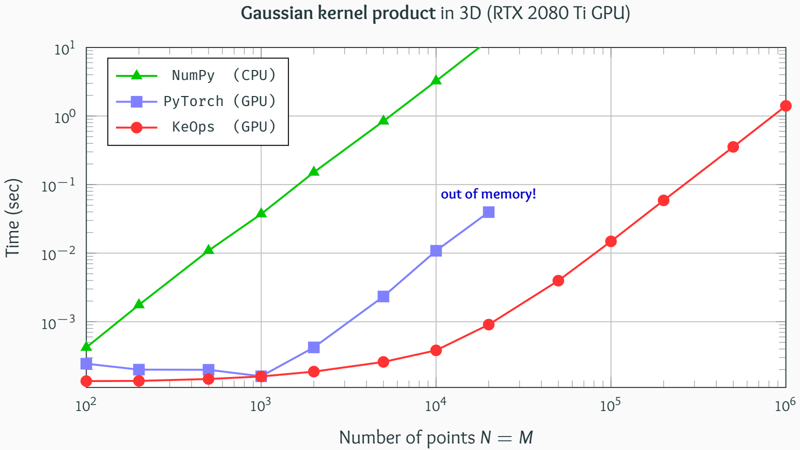
It works!

```python
# Large point clouds in [0,1]^3
import torch
x = torch.rand(1000000, 3, requires_grad=True).cuda()
y = torch.rand(2000000, 3).cuda()

# Turn our Tensors into KeOps symbolic variables:
from pykeops.torch import LazyTensor
x_i = LazyTensor(x[:,None,:])  # x_i.shape = (1e6, 1, 3)
y_j = LazyTensor(y[None,:,:])  # y_j.shape = ( 1, 2e6,3)

# Perform large-scale computations:
D_ij = ((x_i - y_j)**2).sum(dim=2)  # (1e6,2e6,1)
K_ij = (- D_ij).exp()               # (1e6,2e6,1)

# Reduction: symbolic LazyTensor -> genuine torch Tensor
a_i = K_ij.sum(dim=1)  # (1e6, 1) torch.cuda.FloatTensor
```

It works!

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(1000000, 3, requires_grad=True).cuda()
y = torch.rand(2000000, 3).cuda()

# Turn our Tensors into KeOps symbolic variables:
from pykeops.torch import LazyTensor
x_i = LazyTensor(x[:,None,:])  # x_i.shape = (1e6, 1, 3)
y_j = LazyTensor(y[None,:,:])  # y_j.shape = ( 1, 2e6,3)

# Perform large-scale computations:
D_ij = ((x_i - y_j)**2).sum(dim=2)  # (1e6,2e6,1)
K_ij = (- D_ij).exp()               # (1e6,2e6,1)

# Reduction: symbolic LazyTensor -> genuine torch Tensor
a_i = K_ij.sum(dim=1)  # (1e6, 1) torch.cuda.FloatTensor
g_x = torch.autograd.grad((a_i ** 2).sum(), [x])
```

Scaling up to real data



- 2 years of work with **Joan Glaunès** and **Benjamin Charlier**.

The KeOps library

- 2 years of work with **Joan Glaunès** and **Benjamin Charlier**.
- **Cross-platform**: R, Matlab, NumPy *and* PyTorch.

The KeOps library

- 2 years of work with **Joan Glaunès** and **Benjamin Charlier**.
- **Cross-platform**: R, Matlab, NumPy *and* PyTorch.
- **Versatile**: many operations, variables, reductions.

The KeOps library

- 2 years of work with **Joan Glaunès** and **Benjamin Charlier**.
- **Cross-platform**: R, Matlab, NumPy *and* PyTorch.
- **Versatile**: many operations, variables, reductions.
- **Efficient**: $O(N)$ memory, competitive runtimes.

The KeOps library

- 2 years of work with **Joan Glaunès** and **Benjamin Charlier**.
- **Cross-platform**: R, Matlab, NumPy *and* PyTorch.
- **Versatile**: many operations, variables, reductions.
- **Efficient**: $O(N)$ memory, competitive runtimes.
- **Powerful**: autodiff, block-sparsity, etc.

The KeOps library

- 2 years of work with **Joan Glaunès** and **Benjamin Charlier**.
- **Cross-platform**: R, Matlab, NumPy *and* PyTorch.
- **Versatile**: many operations, variables, reductions.
- **Efficient**: $O(N)$ memory, competitive runtimes.
- **Powerful**: autodiff, block-sparsity, etc.
- **Transparent**: interface with **SciPy**, GPytorch, etc.

The KeOps library

- 2 years of work with **Joan Glaunès** and **Benjamin Charlier**.
- **Cross-platform**: R, Matlab, NumPy *and* PyTorch.
- **Versatile**: many operations, variables, reductions.
- **Efficient**: $O(N)$ memory, competitive runtimes.
- **Powerful**: autodiff, block-sparsity, etc.
- **Transparent**: interface with **SciPy**, GPytorch, etc.

- Scikit-learn-like **documentation**:

`www.kernel-operations.io`

Geometric Loss functions for PyTorch

Our website: `www.kernel-operations.io/geomloss`

$\Longrightarrow$ `pip install geomloss` $\Longleftarrow$

Geometric Loss functions for PyTorch

Our website: `www.kernel-operations.io/geomloss`

$\Longrightarrow$ `pip install geomloss` $\Longleftarrow$

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(100000, 3, requires_grad=True).cuda()
y = torch.rand(200000, 3).cuda()
```

Geometric Loss functions for PyTorch

Our website: `www.kernel-operations.io/geomloss`

$\Longrightarrow$ `pip install geomloss` $\Longleftarrow$

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(100000, 3, requires_grad=True).cuda()
y = torch.rand(200000, 3).cuda()

# Define a Wasserstein loss between sampled measures
from geomloss import SamplesLoss  # See also ImagesLoss...
loss = SamplesLoss(loss="sinkhorn", p=2, blur=.05)
```

Geometric Loss functions for PyTorch

Our website: www.kernel-operations.io/geomloss

$\Longrightarrow$ pip install geomloss $\Longleftarrow$

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(100000, 3, requires_grad=True).cuda()
y = torch.rand(200000, 3).cuda()

# Define a Wasserstein loss between sampled measures
from geomloss import SamplesLoss  # See also ImagesLoss...
loss = SamplesLoss(loss="sinkhorn", p=2, blur=.05)

L = loss(x, y)  # By default, use constant weights
```

Geometric Loss functions for PyTorch

Our website: www.kernel-operations.io/geomloss

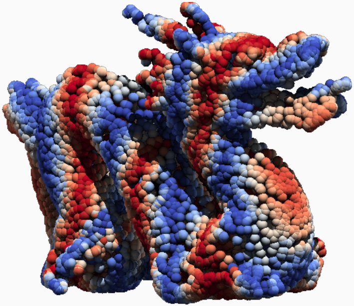
$\Longrightarrow$ pip install geomloss $\Longleftarrow$

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(100000, 3, requires_grad=True).cuda()
y = torch.rand(200000, 3).cuda()

# Define a Wasserstein loss between sampled measures
from geomloss import SamplesLoss  # See also ImagesLoss...
loss = SamplesLoss(loss="sinkhorn", p=2, blur=.05)

L = loss(x, y)  # By default, use constant weights
# GeomLoss supports autograd, batch processing, etc.
g_x, = torch.autograd.grad(L, [x])
```

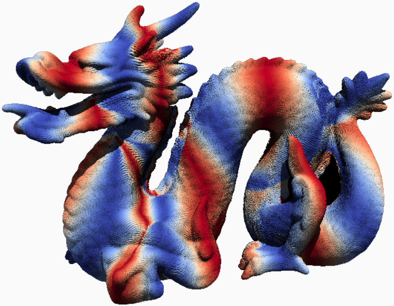
Scaling up optimal transport

Precision controlled by the ratio $\frac{\text{blur}}{\text{diameter}}$.

With a precision of 1%, on a modern gaming GPU – RTX 2080 Ti:



10k points in 30-50ms



100k points in 100-200ms

Conclusion

Wasserstein distance = Multi-dimensional sorting problem ?

The **three regimes** of Optimal Transport:

Wasserstein distance = Multi-dimensional sorting problem ?

The **three regimes** of Optimal Transport:

- α, β live in **dimension 1**
  $\Longrightarrow$ Simple sorting problem
  $\Longrightarrow$ Quicksort in $\mathbf{O(N \log N)}$.

Wasserstein distance = Multi-dimensional sorting problem ?

The **three regimes** of Optimal Transport:

- α, β live in **dimension 1**
  $\Longrightarrow$ Simple sorting problem
  $\Longrightarrow$ Quicksort in $\mathbf{O}(\mathbf{N} \log \mathbf{N})$.

- α, β live in dimension **10+**
  $\Longrightarrow$ $C(x_i, y_j)$ has very little structure
  $\Longrightarrow$ Compute all pairs in $\geqslant \mathbf{O}(\mathbf{N}^2)$.

Wasserstein distance = Multi-dimensional sorting problem ?

The **three regimes** of Optimal Transport:

- α, β live in **dimension 1**
  $\Longrightarrow$ Simple sorting problem
  $\Longrightarrow$ Quicksort in $\mathbf{O(N \log N)}$.

- α, β have a **small** intrinsic **dimension**

- α, β live in dimension **10+**
  $\Longrightarrow$ $C(x_i, y_j)$ has very little structure
  $\Longrightarrow$ Compute all pairs in $\geqslant \mathbf{O(N^2)}$.

Wasserstein distance = Multi-dimensional sorting problem ?

The **three regimes** of Optimal Transport:

- α, β live in **dimension 1**
  $\Longrightarrow$ Simple sorting problem
  $\Longrightarrow$ Quicksort in $\mathbf{O}(\mathbf{N} \log \mathbf{N})$.

- α, β have a **small** intrinsic **dimension**
  $\Longrightarrow$ Rely on multiscale strategies

- α, β live in dimension **10+**
  $\Longrightarrow$ $C(x_i, y_j)$ has very little structure
  $\Longrightarrow$ Compute all pairs in $\geqslant \mathbf{O}(\mathbf{N}^2)$.

Wasserstein distance = Multi-dimensional sorting problem ?

The **three regimes** of Optimal Transport:

- α, β live in **dimension 1**
  $\Longrightarrow$ Simple sorting problem
  $\Longrightarrow$ Quicksort in $\mathbf{O(N \log N)}$.

- α, β have a **small** intrinsic **dimension**
  $\Longrightarrow$ Rely on multiscale strategies
  $\Longrightarrow$ Multiscale Sinkhorn in $\mathbf{O(N \log N)}$ **on the GPU.**

- α, β live in dimension **10+**
  $\Longrightarrow$ $C(x_i, y_j)$ has very little structure
  $\Longrightarrow$ Compute all pairs in $\geqslant \mathbf{O(N^2)}$.

Wasserstein distance = Multi-dimensional sorting problem ?

The **three regimes** of Optimal Transport:

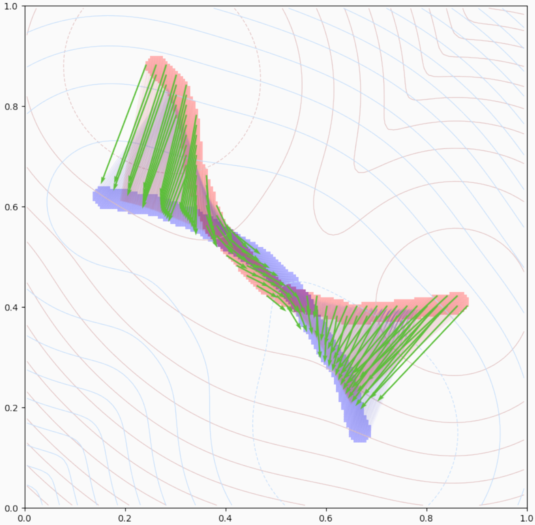
- α, β live in **dimension 1**
  $\Longrightarrow$ Simple sorting problem
  $\Longrightarrow$ Quicksort in $\mathbf{O(N \log N)}$.

- α, β have a **small** intrinsic **dimension**
  $\Longrightarrow$ Rely on multiscale strategies
  $\Longrightarrow$ Multiscale Sinkhorn in $\mathbf{O(N \log N)}$ **on the GPU.**

- α, β live in dimension **10+**
  $\Longrightarrow$ $C(x_i, y_j)$ has very little structure
  $\Longrightarrow$ Compute all pairs in $\geqslant \mathbf{O(N^2)}$.

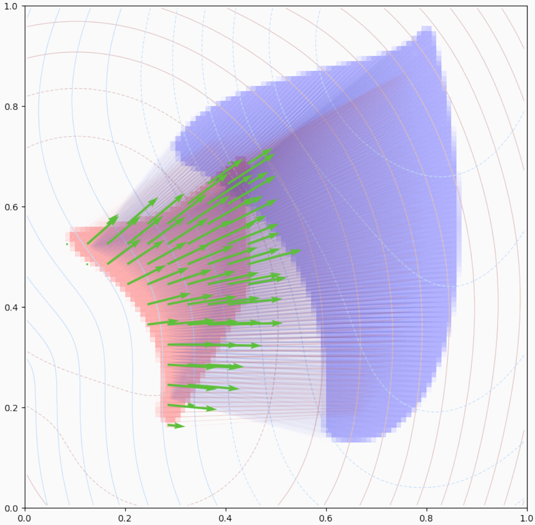
$\Longrightarrow$ Multiscale Sinkhorn algorithm $\simeq$ Multi-dimensional **Quicksort.**

A robust loss function



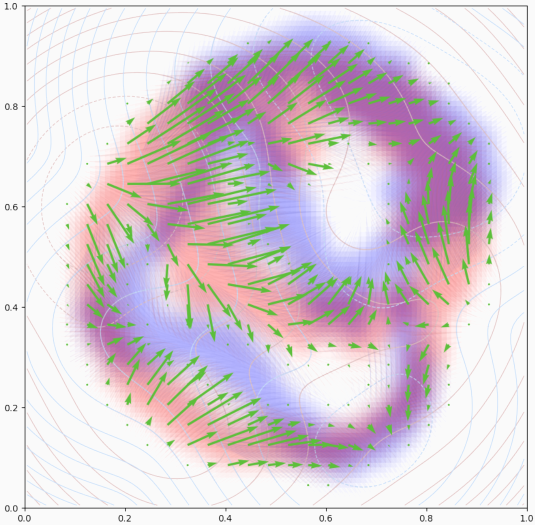
A high-quality gradient…

A robust loss function



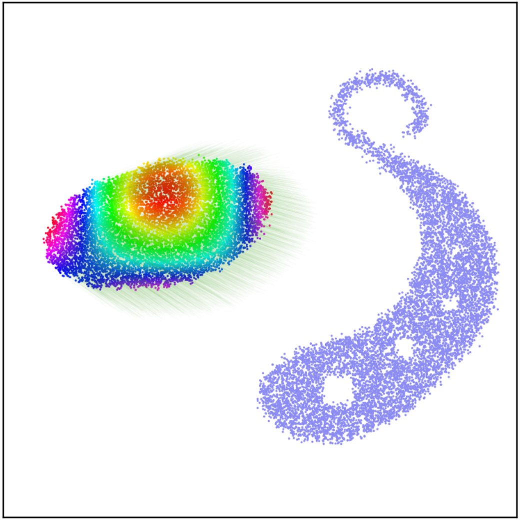
A high-quality gradient...

A robust loss function



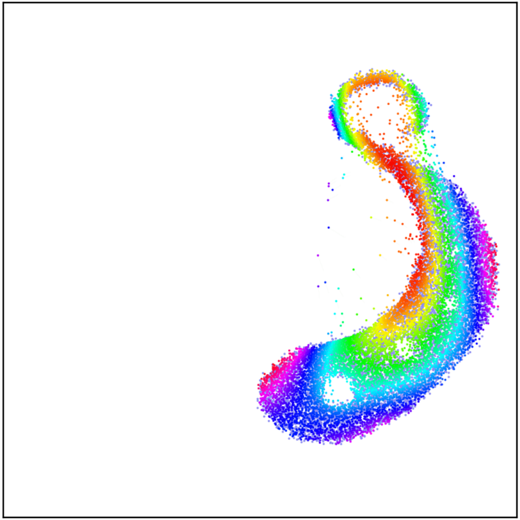
A high-quality gradient... But no preservation of topology!

Gradient descent with OT: cheap'n easy registration?



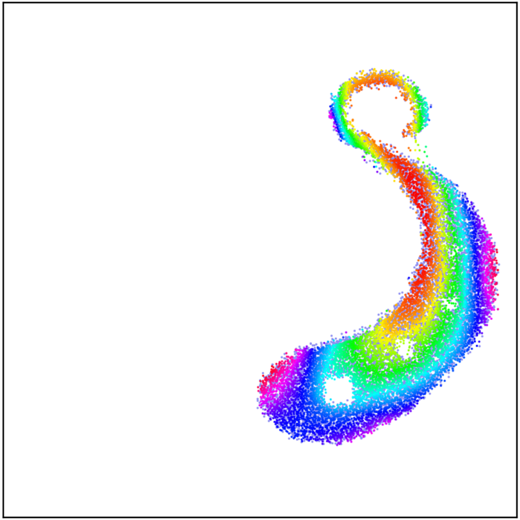
Iteration 0

Gradient descent with OT: cheap'n easy registration?



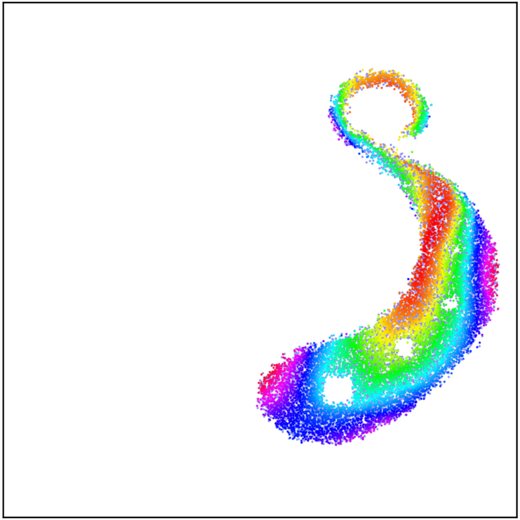
Iteration 1

Gradient descent with OT: cheap'n easy registration?



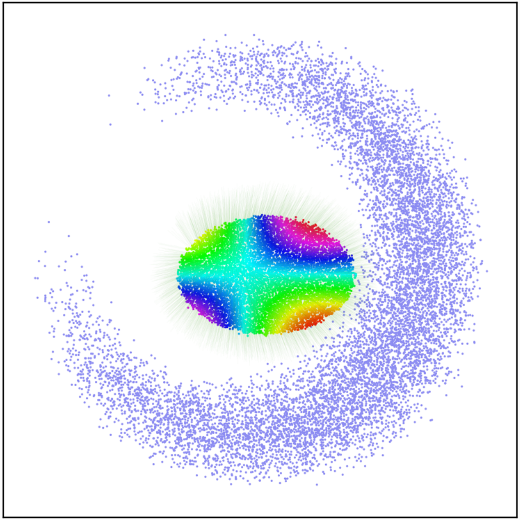
Iteration 2

Gradient descent with OT: cheap'n easy registration?



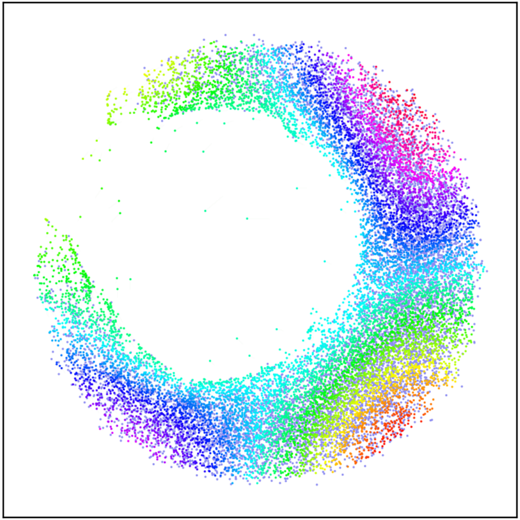
Iteration 10

Gradient descent with OT: cheap'n easy registration?



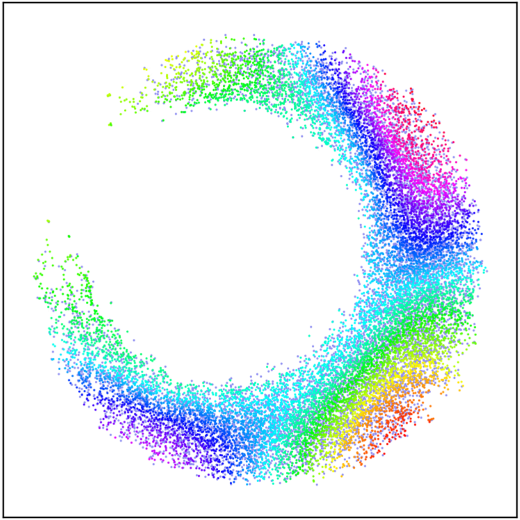
Iteration 0

Gradient descent with OT: cheap'n easy registration?



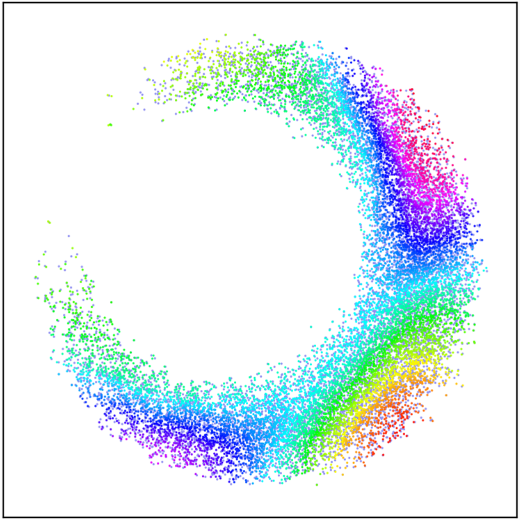
Iteration 1

Gradient descent with OT: cheap'n easy registration?



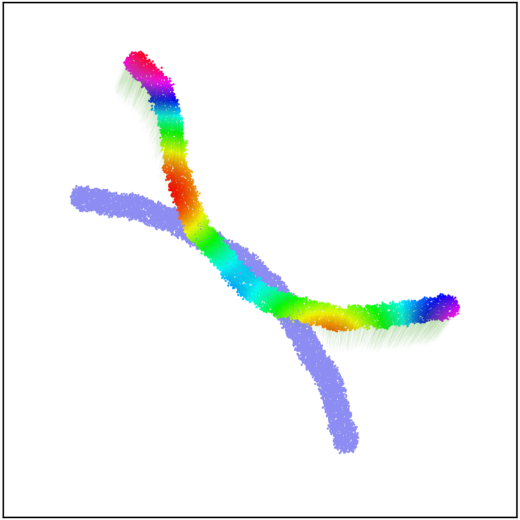
Iteration 2

Gradient descent with OT: cheap'n easy registration?



Iteration 10

Gradient descent with OT: cheap'n easy registration?



Iteration 0

Gradient descent with OT: cheap'n easy registration?

Iteration 1

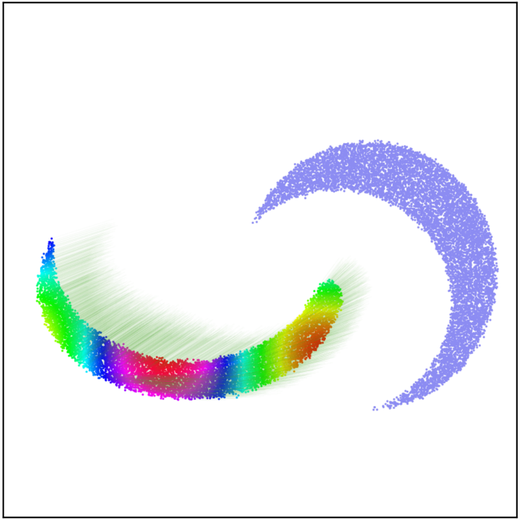
Gradient descent with OT: cheap’n easy registration?

Iteration 2

Gradient descent with OT: cheap'n easy registration?

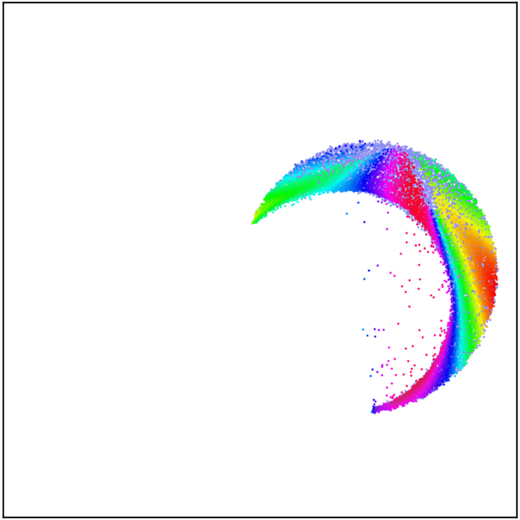
Iteration 10

Gradient descent with OT: cheap'n easy registration? Beware!



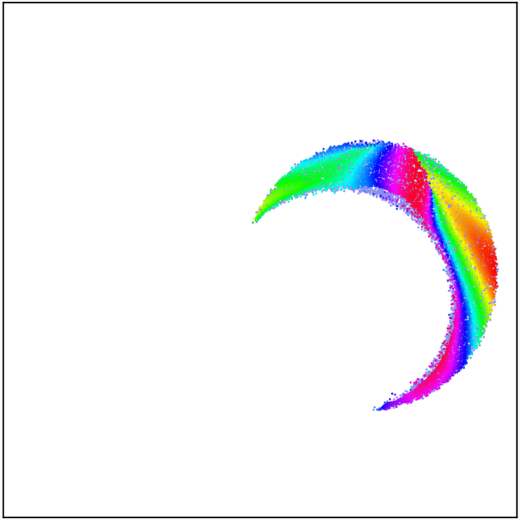
Iteration 0

Gradient descent with OT: cheap'n easy registration? Beware!



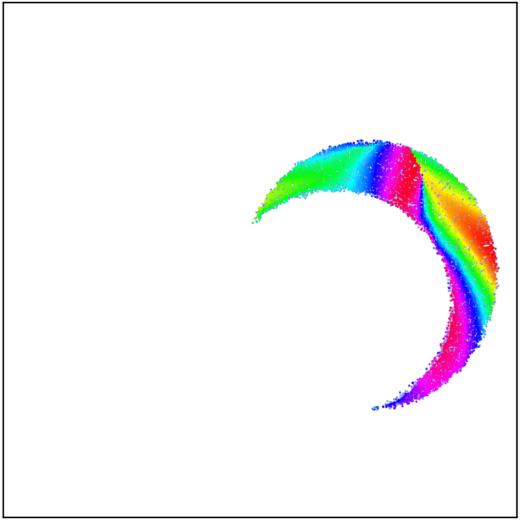
Iteration 1

Gradient descent with OT: cheap'n easy registration? Beware!



Iteration 2

Gradient descent with OT: cheap'n easy registration? Beware!



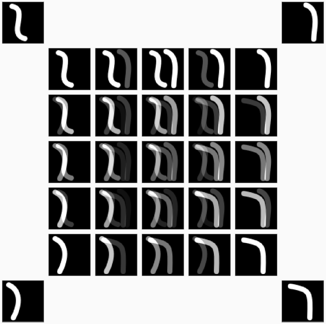
Iteration 10

Affordable geometric barycenters

$$\text{Barycenter } \alpha^* \;=\; \arg\min_{\alpha} \sum_{i=1}^{N} \lambda_i \,\text{Loss}(\,\alpha\,,\,\beta_i\,)\,.$$

Affordable geometric barycenters

$$\text{Barycenter } \alpha^* = \arg\min_{\alpha} \sum_{i=1}^{N} \lambda_i \, \text{Loss}(\alpha, \beta_i) .$$

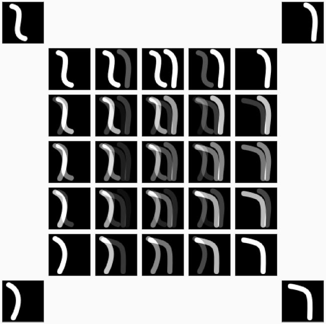


Linear barycenters

$$\text{Loss}(\alpha, \beta) = \|\alpha - \beta\|_{L^2}^2$$

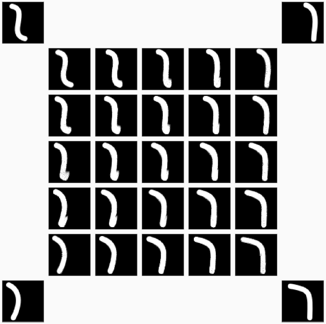
Affordable geometric barycenters

$$\text{Barycenter } \alpha^* = \arg\min_{\alpha} \sum_{i=1}^{N} \lambda_i \, \text{Loss}(\alpha, \beta_i) \,.$$



Linear barycenters

$$\text{Loss}(\alpha, \beta) = \|\alpha - \beta\|^2_{L^2}$$



Wasserstein barycenters

$$\text{Loss}(\alpha, \beta) = \text{OT}(\alpha, \beta)$$

Key points

- **Symbolic LazyTensors** are key to performances:
  $\longrightarrow$ KeOps, x30 speed-up vs PyTorch and TF.

Key points

- **Symbolic LazyTensors** are key to performances:
  $\longrightarrow$ KeOps, x30 speed-up vs PyTorch and TF.

- Optimal Transport = **generalized sorting**:
  $\longrightarrow$ Geometric gradients.
  $\longrightarrow$ Super-fast $O(N \log N)$ solvers.

Key points

- **Symbolic LazyTensors** are key to performances:
  $\longrightarrow$ KeOps, x30 speed-up vs PyTorch and TF.

- Optimal Transport = **generalized sorting**:
  $\longrightarrow$ Geometric gradients.
  $\longrightarrow$ Super-fast $O(N \log N)$ solvers.

Key points

- **Symbolic LazyTensors** are key to performances:
  $\longrightarrow$ KeOps, x30 speed-up vs PyTorch and TF.

- Optimal Transport = **generalized sorting**:
  $\longrightarrow$ Geometric gradients.
  $\longrightarrow$ Super-fast $O(N \log N)$ solvers.

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

Key points

- **Symbolic LazyTensors** are key to performances:
  $\longrightarrow$ KeOps, x30 speed-up vs PyTorch and TF.

- Optimal Transport = **generalized sorting**:
  $\longrightarrow$ Geometric gradients.
  $\longrightarrow$ Super-fast $O(N \log N)$ solvers.

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

Main reference: my **PhD thesis**.

Key points

For **users**: reliable, efficient `python` toolboxes:

- Fluid mechanics: `github.com/sd-ot/pysdot`
- Machine Learning on the CPU: `pot.readthedocs.io`
- Graphics, large-scale ML on the GPU:
  `www.kernel-operations.io/geomloss`

Key points

For **users**: reliable, efficient `python` toolboxes:

- Fluid mechanics: `github.com/sd-ot/pysdot`
- Machine Learning on the CPU: `pot.readthedocs.io`
- Graphics, large-scale ML on the GPU:
  `www.kernel-operations.io/geomloss`

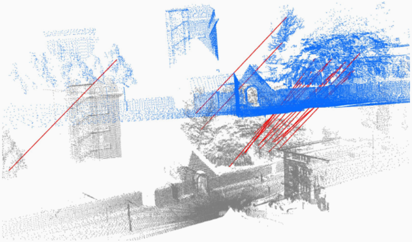
For **us**: new interesting questions:

- Link between S_ε and a **blurred Wasserstein** distance?
- What about **graphs**?

Thank you for your attention.

Any questions ?

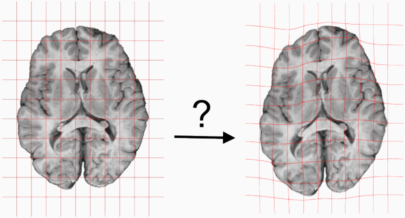
First setting: processing of point clouds



From the documentation of the Point Cloud Library.

- φ is **rigid** or affine
- Occlusions
- Outliers

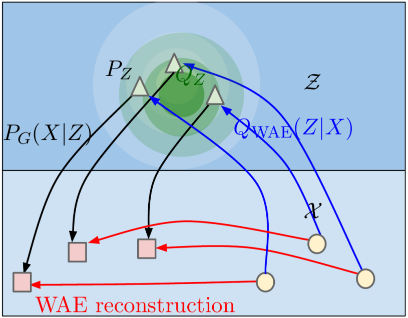
Second setting: medical imaging



From Marc Niethammer's Quicksilver slides.

- φ is a spline or a **diffeomorphism**
- Ill-posed problem
- Some occlusions

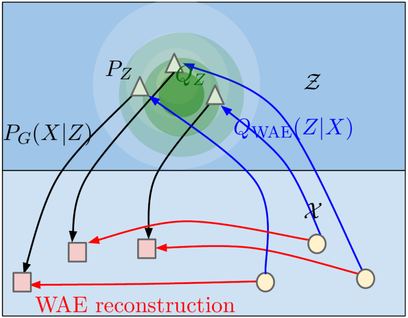
Third setting: training a generative model



Wasserstein Auto-Encoders,
Tolstikhin et al., 2018.

- φ is a **neural network**
- Very weak regularization
- High-dimensional space

Third setting: training a generative model



Wasserstein Auto-Encoders,
Tolstikhin et al., 2018.

- φ is a **neural network**
- Very weak regularization
- High-dimensional space

Which **Loss** function should we use?

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) = \max_{f \in B} \langle \alpha - \beta , f \rangle,$$

$$\text{look for } \theta^* = \arg\min_{\theta} \max_{f \in B} \langle \alpha(\theta) - \beta , f \rangle$$

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \,\rangle,$$

$$\text{look for } \theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \,\rangle$$

- $B = \{\, \| f \|_\infty \leqslant 1 \,\} \implies$ Loss = TV norm:
  - zero geometry
  - **too many** test functions

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) = \max_{f \in B} \langle \alpha - \beta, f \rangle,$$

$$\text{look for } \theta^* = \arg\min_{\theta} \max_{f \in B} \langle \alpha(\theta) - \beta, f \rangle$$

- $B = \{ \| f \|_\infty \leqslant 1 \} \implies \text{Loss} = \text{TV norm:}$
    - zero geometry
    - **too many** test functions
- $B = \{ \| f \|_2^2 + \| \nabla f \|_2^2 + \cdots \leqslant 1 \} \implies \text{Loss} = \text{kernel norm:}$
    - may saturate at infinity
    - **screening** artifacts

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \; = \; \max_{f \in B} \langle \, \alpha - \beta \, , \, f \rangle,$$

$$\text{look for } \; \theta^* \; = \; \arg\min_{\theta} \max_{f \in B} \langle \, \alpha(\theta) - \beta \, , \, f \rangle$$

- $B = \{ \, f \text{ is 1-Lipschitz} \, \} \implies$ Loss $=$ Wasserstein-1 (OT_0):

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \rangle,$$

$$\text{look for } \; \theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \rangle$$

- $B = \{\, f \text{ is 1-Lipschitz} \,\} \implies$ Loss $=$ Wasserstein-1 (OT_0):
  - S_ε is nearly as efficient as a **closed formula**

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \,\rangle,$$

$$\text{look for } \;\theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \,\rangle$$

- $B = \{\, f \text{ is 1-Lipschitz} \,\} \implies$ Loss $=$ Wasserstein-1 (OT_0):
  - S_ε is nearly as efficient as a **closed formula**
  - relevant in **low dimensions**
  - **useless** in $(\mathbb{R}^{512\times 512}, \|\cdot\|_2)$: the ground cost makes no sense

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \,\rangle,$$

$$\text{look for } \;\theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \,\rangle$$

- $B = \{\, f \text{ is 1-Lipschitz} \,\} \implies$ Loss $=$ Wasserstein-1 (OT_0):
  - S_ε is nearly as efficient as a **closed formula**
  - relevant in **low dimensions**
  - **useless** in $(\mathbb{R}^{512 \times 512}, \|\cdot\|_2)$: the ground cost makes no sense

- $B \simeq \{\, f \text{ is 1-Lipschitz} \,\} \bigcap \{\, f \text{ is a } \textbf{CNN} \,\}$
  $\implies$ Loss $=$ Wasserstein GAN :

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \,\rangle,$$

$$\text{look for } \;\theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \,\rangle$$

- $B = \{\, f \text{ is 1-Lipschitz} \,\} \implies$ Loss $=$ Wasserstein-1 (OT_0):
  - S_ε is nearly as efficient as a **closed formula**
  - relevant in **low dimensions**
  - **useless** in $(\mathbb{R}^{512\times 512}, \|\cdot\|_2)$: the ground cost makes no sense

- $B \simeq \{\, f \text{ is 1-Lipschitz} \,\} \bigcap \{\, f \text{ is a } \textbf{CNN} \,\}$
  $\implies$ Loss $=$ Wasserstein GAN :
  - use **perceptually sensible** test functions

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) = \max_{f \in B} \langle\, \alpha - \beta \,, f \rangle,$$

$$\text{look for } \theta^* = \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,, f \rangle$$

- $B = \{\, f \text{ is 1-Lipschitz} \,\} \implies$ Loss $=$ Wasserstein-1 (OT_0):
  - S_ε is nearly as efficient as a **closed formula**
  - relevant in **low dimensions**
  - **useless** in $(\mathbb{R}^{512\times 512}, \|\cdot\|_2)$: the ground cost makes no sense

- $B \simeq \{\, f \text{ is 1-Lipschitz} \,\} \bigcap \{\, f \text{ is a } \textbf{CNN} \,\}$
  $\implies$ Loss $=$ Wasserstein GAN :
  - use **perceptually sensible** test functions
  - no simple formula: use **gradient ascent**

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \,\rangle,$$

$$\text{look for } \; \theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \,\rangle$$

- $B = \{\, f \text{ is 1-Lipschitz} \,\} \implies$ Loss $=$ Wasserstein-1 (OT_0):
  - S_ε is nearly as efficient as a **closed formula**
  - relevant in **low dimensions**
  - **useless** in $(\mathbb{R}^{512\times 512}, \|\cdot\|_2)$: the ground cost makes no sense

- $B \simeq \{\, f \text{ is 1-Lipschitz} \,\} \bigcap \{\, f \text{ is a } \textbf{CNN} \,\}$
  $\implies$ Loss $=$ Wasserstein GAN :
  - use **perceptually sensible** test functions
  - no simple formula: use **gradient ascent**
  - can we provide relevant **insights** to the ML community?

References i

- John Ashburner.
  A fast diffeomorphic image registration algorithm.
  Neuroimage, 38(1):95–113, 2007.
- Dimitri P Bertsekas.
  A distributed algorithm for the assignment problem.
  Lab. for Information and Decision Systems Working Paper, M.I.T., Cambridge, MA, 1979.
- Y. Brenier.
  Polar factorization and monotone rearrangement of vector-valued functions.
  Comm. Pure Appl. Math., 44(4):375–417, 1991.

Christophe Chnafa, Simon Mendez, and Franck Nicoud.
Image-based large-eddy simulation in a realistic left heart.
Computers & Fluids, 94:173–187, 2014.

Lénaïc Chizat, Gabriel Peyré, Bernhard Schmitzer, and François-Xavier Vialard.
Unbalanced optimal transport: Dynamic and kantorovich formulations.
Journal of Functional Analysis, 274(11):3090–3123, 2018.

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-c
2019.
MIT CSAIL.

Marco Cuturi.
Sinkhorn distances: Lightspeed computation of optimal transport.
In *Advances in Neural Information Processing Systems*, pages 2292–2300, 2013.

Olivier Ecabert, Jochen Peters, Matthew J Walker, Thomas Ivanc, Cristian Lorenz, Jens von Berg, Jonathan Lessick, Mani Vembar, and Jürgen Weese.
Segmentation of the heart and great vessels in CT images using a model-based adaptation framework.
Medical image analysis, 15(6):863–876, 2011.

Jean Feydy, Thibault Séjourné, François-Xavier Vialard, Shun-Ichi Amari, Alain Trouvé, and Gabriel Peyré.
Interpolating between optimal transport and MMD using Sinkhorn divergences.
arXiv preprint arXiv:1810.08278, 2018.

Joan Glaunes.
Transport par difféomorphismes de points, de mesures et de courants pour la comparaison de formes et l'anatomie numérique.
These de sciences, Université Paris, 13, 2005.

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.

Irene Kaltenmark, Benjamin Charlier, and Nicolas Charon.
A general framework for curve and surface comparison and registration with oriented varifolds.
In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

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.

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.

Ptrump16.
Irm picture.
https://commons.wikimedia.org/w/index.php?curid=64157788, 2019.
CC BY-SA 4.0.

Bernhard Schmitzer.
Stabilized sparse scaling algorithms for entropy regularized transport problems.
SIAM Journal on Scientific Computing, 41(3):A1443–A1481, 2019.