Kernel norms on normal cycles and the KeOps library for linear memory reductions over datasets.

Joan Alexis Glaunès
MAP5, Université Paris Descartes

Imaging in Paris workshop - IHP - March 14th, 2019

Outline

Kernel norms on normal cycles for curve and surface registration
(joint work with Pierre Roussillon)

The KeOps library for linear memory reductions over datasets
(joint work with Benjamin Charlier and Jean Feydy)

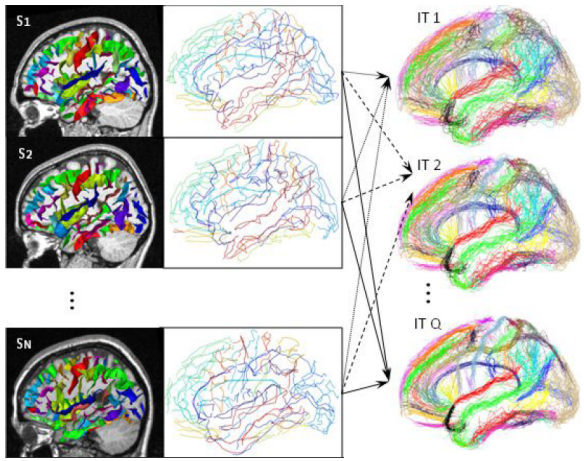
Outline

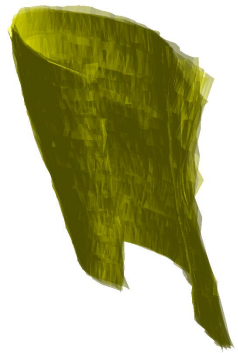
Kernel norms on normal cycles for curve and surface registration
(joint work with Pierre Roussillon)

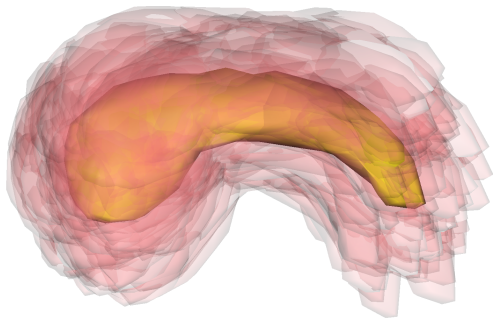
The KeOps library for linear memory reductions over datasets
(joint work with Benjamin Charlier and Jean Feydy)

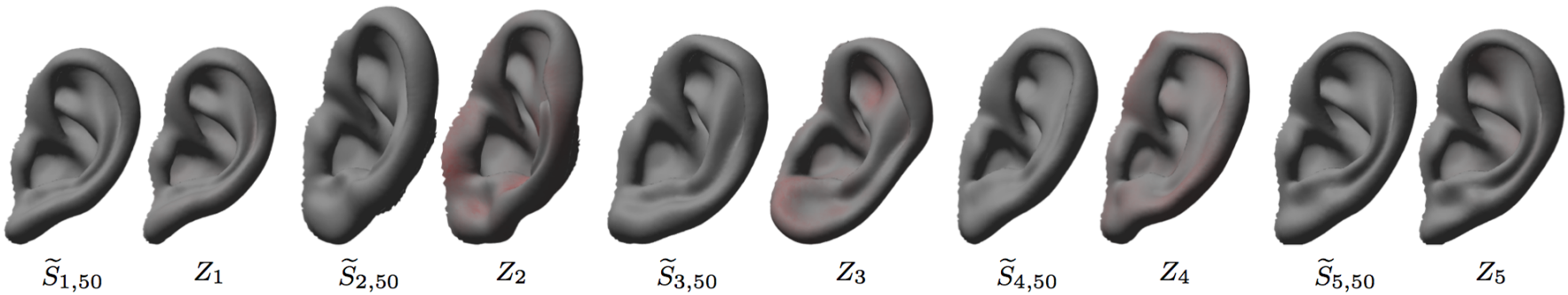
Registration and computational anatomy

- Registration : estimate an optimal deformation (rigid/non-rigid) between two shapes or images
- computational anatomy : study variability of biological shapes through induced deformations









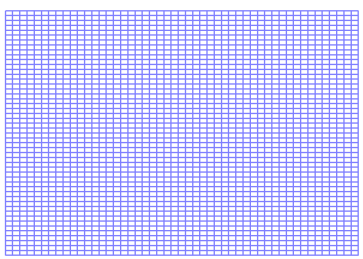
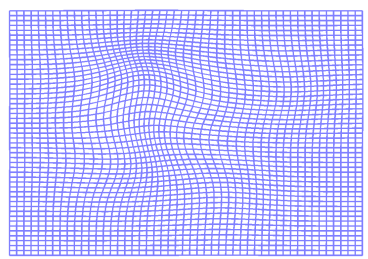
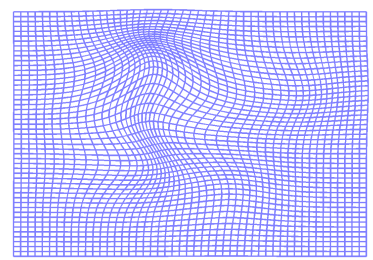
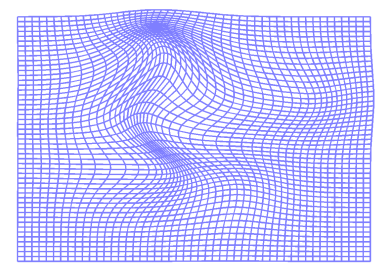
The LDDMM framework

[Trouvé, 1995, Christensen et al., 1996]

- build deformations as flows of time-dependent velocity fields in $\mathbf{R}^d$:

$$\begin{cases} \frac{\partial}{\partial t}\phi^v(t,x) = v(t,\phi^v(t,x)) \\ \phi^v(0,x) = x. \end{cases} \qquad v(t,\cdot) \in V \hookrightarrow C_0^1(\mathbf{R}^d;\mathbf{R}^d)$$

- V Hilbert space, $E(v) := \int_0^1 \|v(t,\cdot)\|_V^2 dt$ is interpreted as an energy/cost of the global deformation $\phi^v(1,\cdot)$.

Inexact matching as a variational problem

Matching from a source shape S to a target shape T

- Variational formulation :

$$\mathcal{J}(v) = \gamma E(v) + \mathcal{A}(\phi^v(1, S), T),$$

  where $\mathcal{A}$ measures dissimilarity between $\phi^v(1, S)$ and T.

- discrete formulation: when S sampled by n points x^i, we write $\mathcal{A}$ as a function of the final positions $q^i(1) = \phi^v(1, x^i)$.

- At the optimum :

$$\begin{cases} v(t,x) = \sum_{i=1}^{n} K(x, q^i(t)) p^i(t) \quad (=: K(x, \mathbf{q})\mathbf{p}) \\ \|v(t,\cdot)\|_V^2 = \sum_{i,j=1}^{n} \left\langle p^j(t) \, , K(q^j(t), q^i(t)) p^i(t) \right\rangle \quad (=: \langle \mathbf{p} \, , K(\mathbf{q}, \mathbf{q})\mathbf{p} \rangle) \end{cases}$$

  where K is the reproducing kernel of space V, e.g.

$$K(x,y) = e^{-\frac{\|x-y\|^2}{\sigma_V^2}}, \qquad K(x,y) = \frac{1}{1 + \|x-y\|^2/\sigma_V^2}$$

Geodesic equations in the space of landmarks

- for discrete problems at the optimum,

$$\|v(t,\cdot)\|_V^2 = \langle \mathbf{p}\,, K(\mathbf{q},\mathbf{q})\mathbf{p}\rangle = \left\langle \dot{\mathbf{q}}\,, K(\mathbf{q},\mathbf{q})^{-1}\dot{\mathbf{q}}\right\rangle$$

  which gives a Riemannian metric on $\mathcal{L}_n(\mathbf{R}^d)$ (space of n distinct points in R^d)

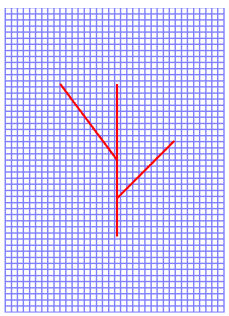
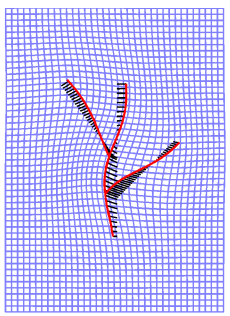
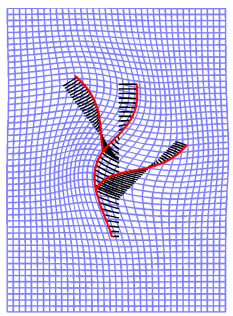
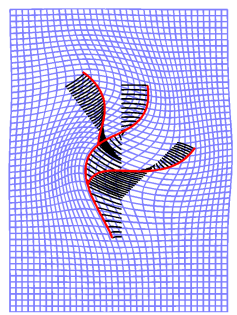
Hamiltonian formulation of geodesics:

- Hamiltonian

$$H(\mathbf{p},\mathbf{q}) = \frac{1}{2}\langle \mathbf{p}\,, K(\mathbf{q},\mathbf{q})\mathbf{p}\rangle\,.$$

- Hamiltonian system of ODEs:

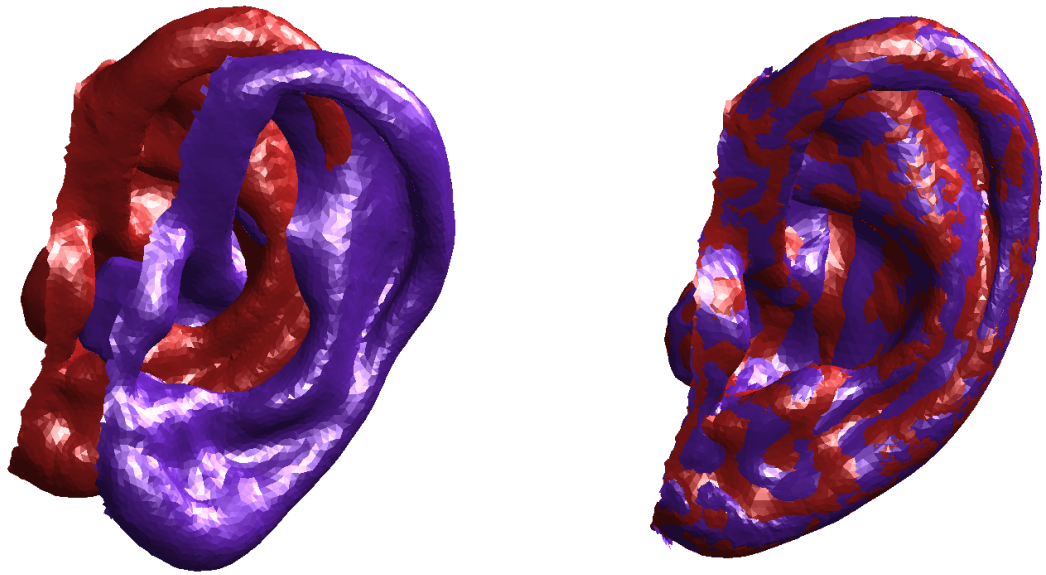
$$\begin{cases} \dot{\mathbf{p}} = -\nabla_{\mathbf{q}} H(\mathbf{p},\mathbf{q}) = -\frac{1}{2}\nabla_{\mathbf{q}}\langle \mathbf{p}\,, K(\mathbf{q},\mathbf{q})\mathbf{p}\rangle \\ \dot{\mathbf{q}} = \nabla_{\mathbf{p}} H(\mathbf{p},\mathbf{q}) = K(\mathbf{q},\mathbf{q})\mathbf{p}, \end{cases}$$

Kernel norms on currents

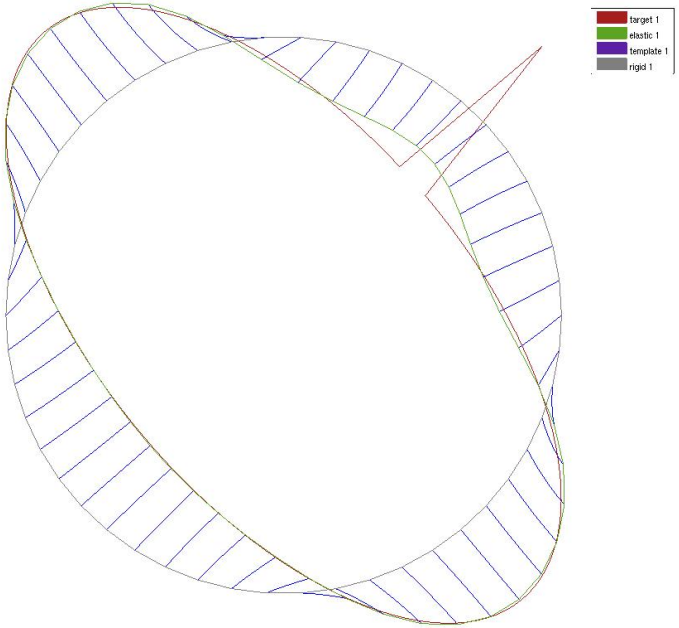
- main goal : propose data fidelity terms for curves and surfaces
- first idea: model these objects as **currents**, i.e. as linear forms over the space of differential forms, then define a Hilbert dual norm
- If $S \subset \mathbf{R}^d$ is m-rectifiable and oriented :
  - associated current :
$$[S](\omega) := \int_S \langle \omega(x) \mid \tau_S(x) \rangle \, d\mathcal{H}^m(x)$$
    where $\omega \in \Omega_0^m(\mathbf{R}^d)$ (m-differential form on $\mathbf{R}^d$)
  - dual norm :
$$\|[S]\|_{W'}^2 = \int_S \int_S \langle K_W(x,y)\tau_S(y) \mid \tau_S(x) \rangle \, d\mathcal{H}^m(y) d\mathcal{H}^m(x)$$

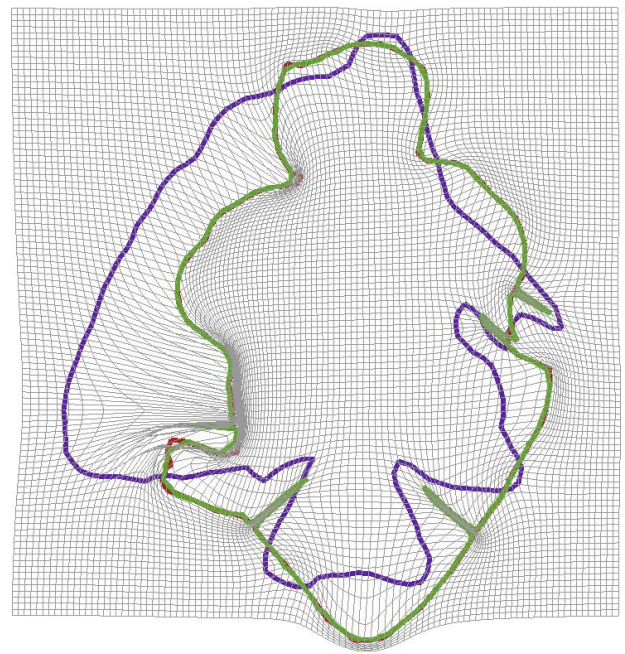
LDDMM registration examples with kernel norms on currents



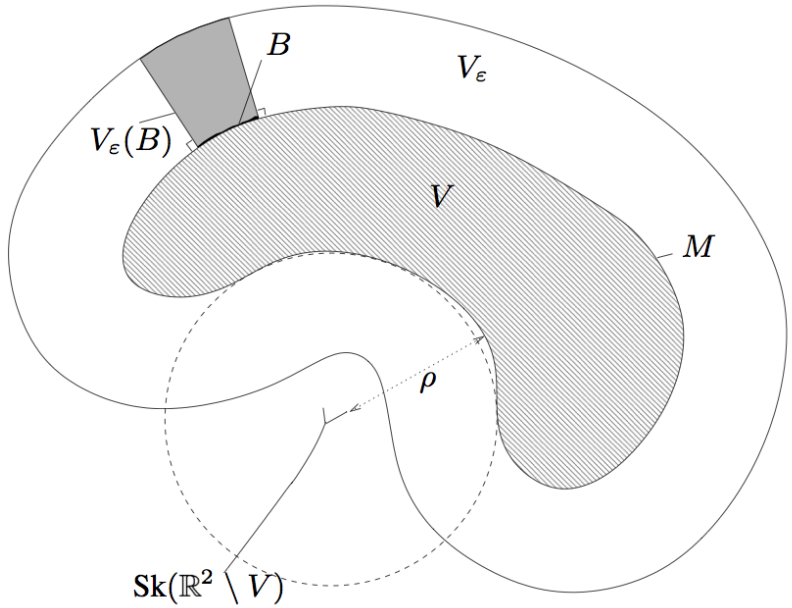
The orientation problem

- Currents model oriented sub-manifolds.
- $\Rightarrow$ spike structures count for almost nothing for the kernel norm, so that they may be disregarded for small σ_W parameter ; or even worse, spikes may be created by the deformation model for small σ_V scale parameter.





Normal cycles : Tube formula, curvature and unit normal bundle



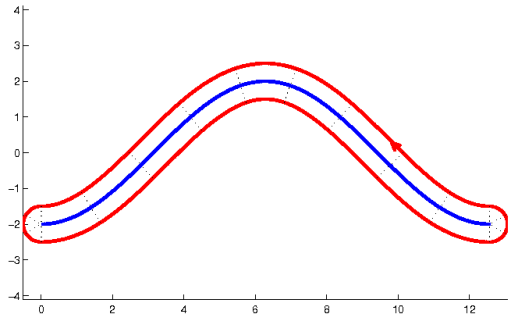
- Weyl's tube formula for $V \subset \mathbf{R}^3$ (with $M := \partial V$) :

$$Vol(V_\varepsilon) = Vol(V) + Area(M)\varepsilon + H(M)\frac{\varepsilon^2}{2} + G(M)\frac{\varepsilon^3}{3},$$

- Extension to "positive reach" subsets (PR) ([Federer, 1959])
- Bijection $\mathcal{N}_V \to V_\varepsilon$, $(x, n) \mapsto x + \varepsilon n$, where $\mathcal{N}_V$ is the unit normal bundle.

Definitions

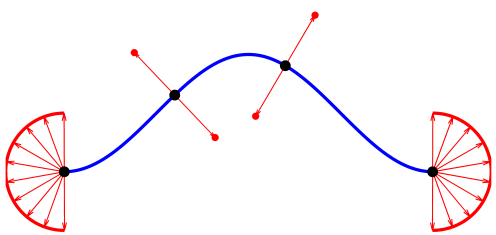
- ε-**tube** around a compact subset $C \subset \mathbf{R}^d$: $C_\varepsilon = \{x \in \mathbf{R}^d, d(x, C) \leq \varepsilon\}$.

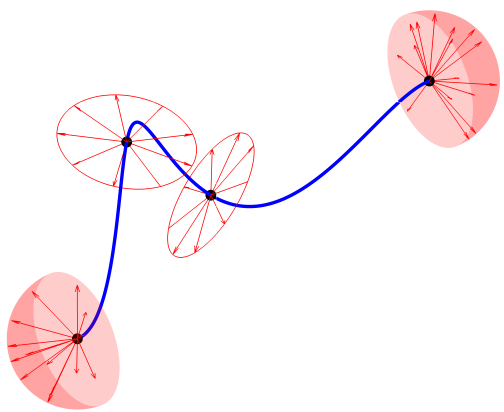


- **Normal cone** at $x \in C$:

$$\hat{\mathcal{N}}(C, x) = \{u \in \mathbf{R}^d, \exists \varepsilon > 0, \forall y \in C \cap B(x, \varepsilon), \langle x - y \, , u \rangle \leq 0\}.$$

- **Unit normal vectors** at $x \in C$: $\mathcal{N}(C, x) = \hat{\mathcal{N}}(C, x) \cap S^{d-1}$.



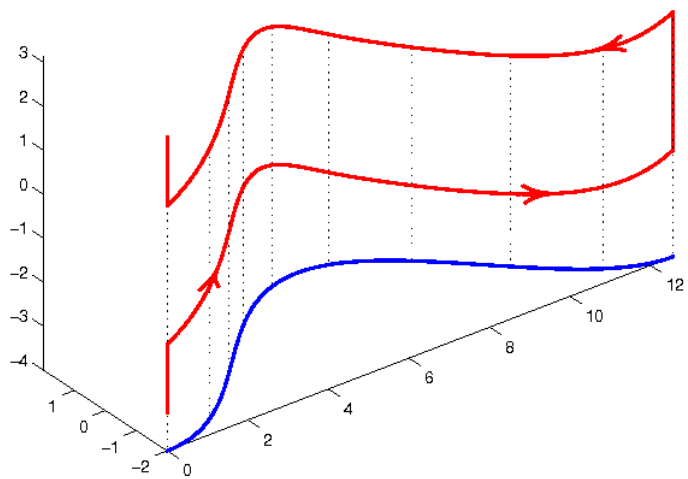


Definitions

- **Unit normal bundle** associated with a subset:

$$\mathcal{N}(C) = \{(x, \xi) \in C \times S^{d-1}, \xi \in \mathcal{N}(C, x)\}.$$

- $\mathcal{N}(C)$ is a closed sub-manifold of dimension $d-1$ in $\mathbf{R}^d \times S^{d-1}$.
- The **normal cycle** associated with C is the current $N(C) := [\mathcal{N}(C)]$ associated with $\mathcal{N}(C)$ (which is canonically oriented).



Link with curvature measures

- Lipschitz-Killing canonical forms: $\omega_0, \ldots, \omega_{d-1}$

$$Vol((C \cap B)_\varepsilon) = \sum_{k=0}^{d} \alpha_k \mathcal{C}_{d-k}(C, B) \varepsilon^k,$$

where $\mathcal{C}_{d-k}(C, B)$ is the generalized curvature measure (integral of the k-th curvature of C over B in the smooth case). One has

$$\mathcal{C}_{d-k}(C, B) = N(C) \lfloor 1_{B \times S^{d-1}}(\omega_k).$$

- Canonical decomposition of the space of differential forms:

$$\Lambda^{d-1}(\mathbf{R}^d \times S^{d-1}) = F_0 \oplus \cdots \oplus F_{d-1},$$

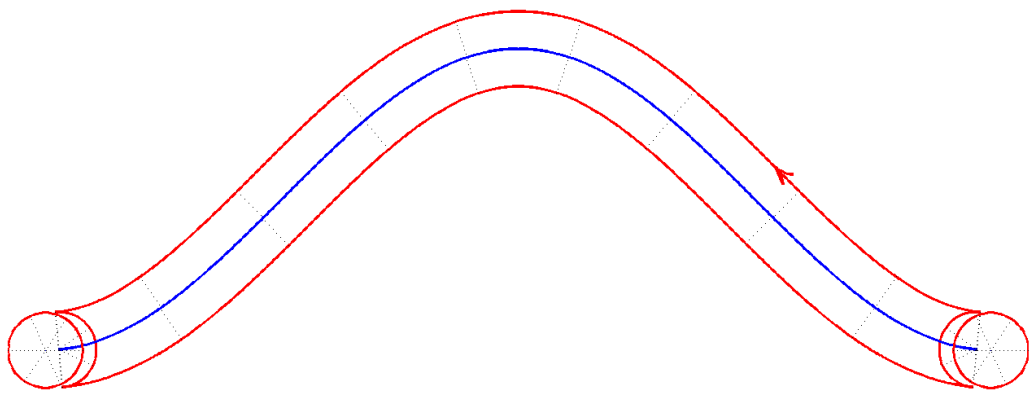
where
$F_k = Span\left\{(u_1, 0) \wedge \cdots \wedge (u_k, 0) \wedge (0, v_1) \wedge \cdots \wedge (0, v_{d-1-k}), \quad u_i, v_j \in \mathbf{R}^d\right\}$
We have that ω_k takes values in F_k.

Addition formula

- The following formula is satisfied as soon as the considered normal cycles are defined:

$$N(C_1 \cup C_2) = N(C_1) + N(C_2) - N(C_1 \cap C_2).$$

- $\Rightarrow$ extension of normal cycles to unions of sets of positive reach $\mathcal{U}_{PR}$.



Properties

- The normal cycle is a second-order model; it encodes curvature information of the set. By computing specific integrals of the normal cycle over a small area, one gets the exact integrated values of the curvature of C on this area.
- The normal cycle does not depend on any choice of orientation on the subset, and there is no need to specify any,
- Since "spikes" are parts of high curvature; they get highly weighted in the model.
- Normal cycles are in fact a model for subsets of $\mathbf{R}^d$ and not for submanifolds of a specific dimension. Hence one can think about comparing a curve with a surface, or to model "hybrid" objects.

Hilbert norms on normal cycles

- Since $[\mathcal{N}(C)]$ is a current in the product space $\mathbf{R}^d \times S^{d-1}$, one must define a kernel on $\mathbf{R}^d \times S^{d-1}$. This can be done by considering a product of two kernels

$$k(\mathbf{x}, \mathbf{y}) = k((x, u), (y, v)) = k_p(x, y) k_n(u, v),$$

where $k_p(x, y)$ is a reproducing kernel in $\mathbf{R}^d$ (e.g. $k_p(x, y) = \frac{1}{1+\|x-y\|^2/\sigma^2}$), and $k_n(u, v)$ is a reproducing kernel in S^{d-1} (e.g. the kernel induced by a Sobolev metric on S^{d-1})

- Let $T(\mathbf{x}) = \tau_1(\mathbf{x}) \wedge \cdots \wedge \tau_{d-1}(\mathbf{x})$, where $(\tau_i(\mathbf{x}))_{1 \leq i \leq d-1}$ is an orthonormal basis on the tangent space $T_{\mathbf{x}}\mathcal{N}(C)$ for any $\mathbf{x} \in \mathcal{N}(C)$. Then one has

$$\|\vec{\mu}_{\mathcal{N}(C)}\|^2_{W'} = \int_{\mathcal{N}(C)} \int_{\mathcal{N}(C)} k(\mathbf{x}, \mathbf{y}) \langle T(\mathbf{x}), T(\mathbf{y}) \rangle \, d\sigma_{\mathcal{N}(C)}(\mathbf{x}) \, d\sigma_{\mathcal{N}(C)}(\mathbf{y}),$$

where $d\sigma_{\mathcal{N}(C)}(\mathbf{x})$ is the volume form on the sub-manifold $\mathcal{N}(C)(\mathbf{x})$

Application to discrete shapes

- Addition formula for normal cycles :

$$N(X \cup Y) := N(X) + N(Y) - N(X \cap Y)$$

- $\Rightarrow$ decompose the computation of the dual norm for unions of segments or triangles.

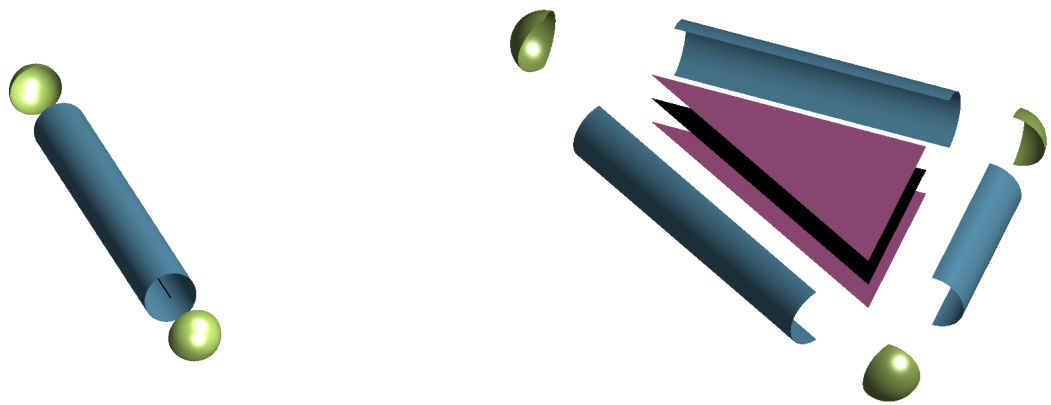
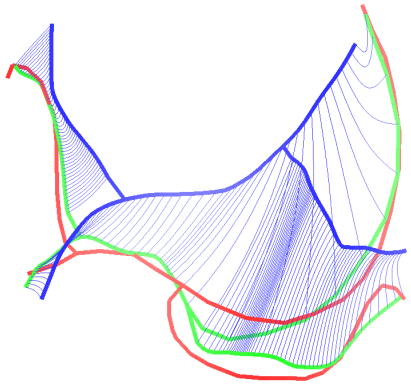
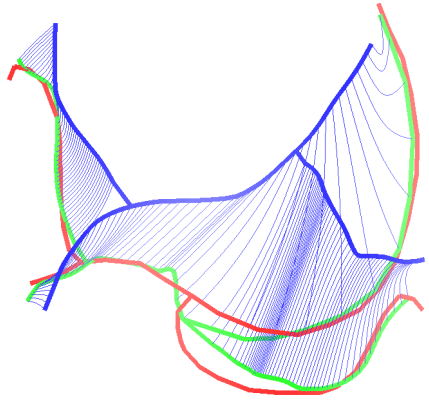


Figure: Illustration of unit normal bundles associated with a segment and a triangle.

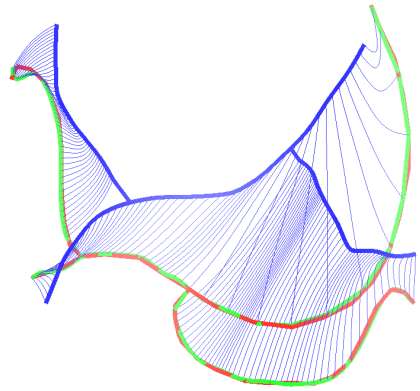
Numerical results



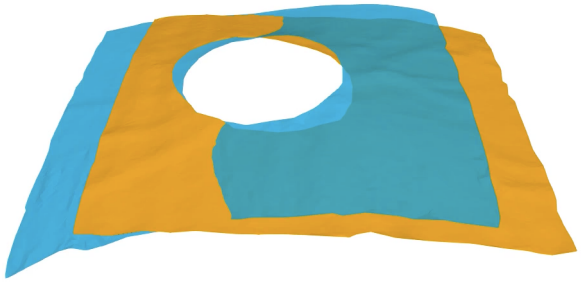
currents



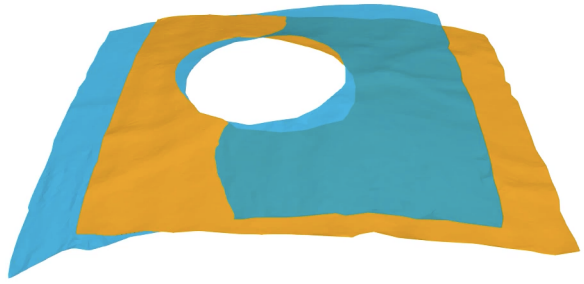
varifolds



normal cycles

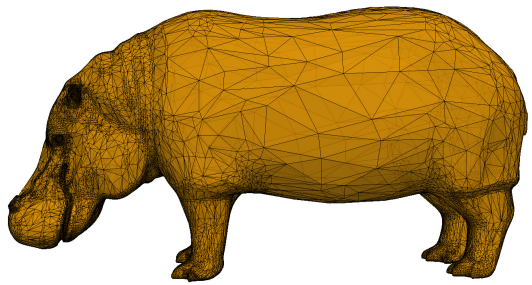


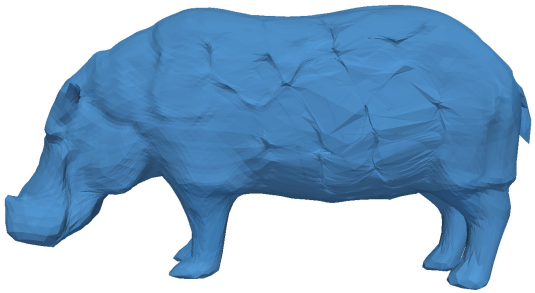
varifolds



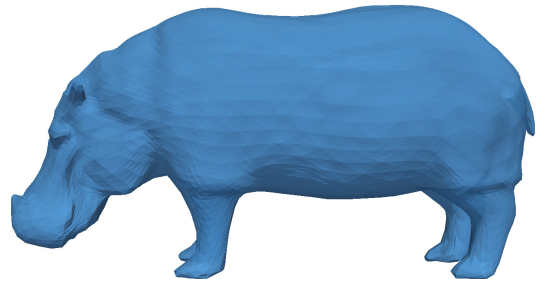
normal cycles

Numerical results





varifolds



normal cycles

Outline

Kernel norms on normal cycles for curve and surface registration
(joint work with Pierre Roussillon)

The KeOps library for linear memory reductions over datasets
(joint work with Benjamin Charlier and Jean Feydy)

Motivations

- Automatic differentiation is of great help in many applications, specially for LDDMM algorithms.

$$\begin{cases} \dot{\mathbf{p}} = -\nabla_{\mathbf{q}} H(\mathbf{p},\mathbf{q}) = -\frac{1}{2}\nabla_{\mathbf{q}} \langle \mathbf{p} \, , K(\mathbf{q},\mathbf{q})\mathbf{p} \rangle \\ \dot{\mathbf{q}} = \nabla_{\mathbf{p}} H(\mathbf{p},\mathbf{q}) = K(\mathbf{q},\mathbf{q})\mathbf{p}, \end{cases}$$

  $\Rightarrow$ second order differentiation needed (one for the ODE system, one for the computation of the gradient)

- Neural networks toolboxes like PyTorch provide automatic differentiation, which can be used for general prupose computations.
- Need for fast and lightweight computation of kernel convolutions of the type

$$\gamma_i = \sum_{j=1}^{n} e^{-\frac{\|x_i - y_j\|^2}{\sigma^2}},$$

  or

$$\gamma_i = \sum_{j=1}^{n} \langle u_i \, , v_j \rangle^2 \, e^{-\frac{\|x_i - y_j\|^2}{\sigma^2}},$$

General reduction operations

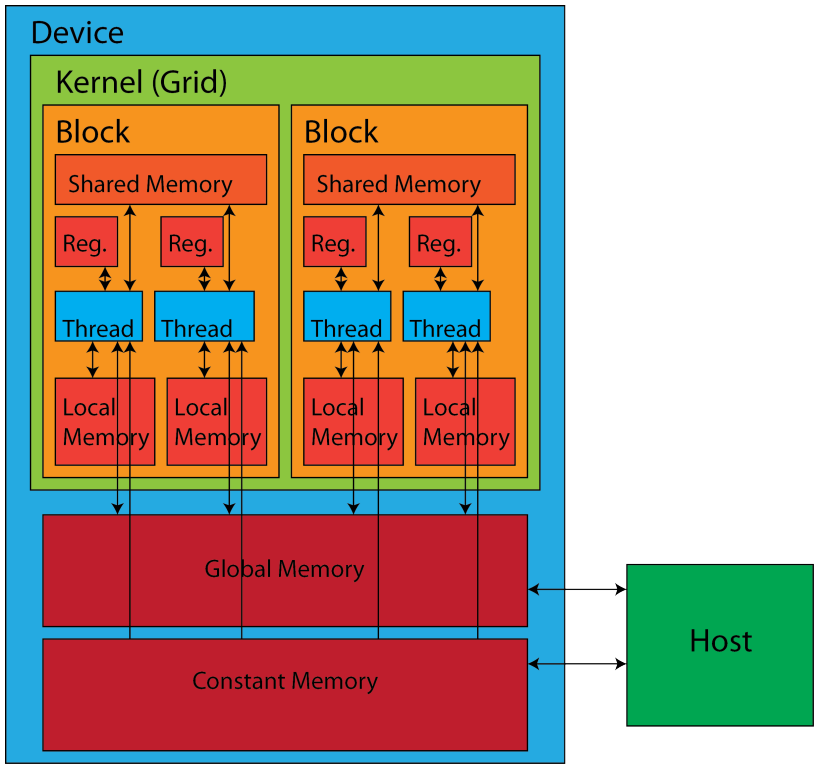
Consider a computation of this type:

$$\gamma_i = \operatorname*{Red}_{j=1}^{n_x} F(\theta, x_i^1, \ldots, x_i^p, y_j^1, \ldots, y_j^q),$$

where

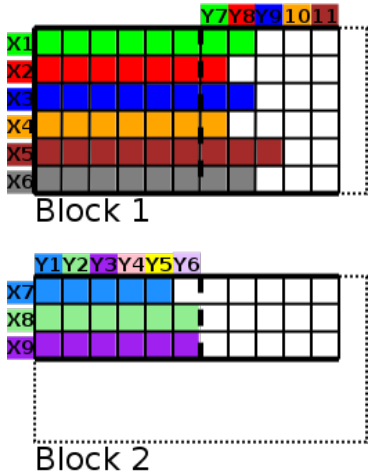
- $\theta \in \mathbf{R}^{d_\theta}$ (parameters), $x_i^k \in \mathbf{R}^{d_x^k}$ (i-indexed inputs), $y_j^l \in \mathbf{R}^{d_y^l}$ (j-indexed inputs),
- $\gamma_i \in \mathbf{R}^{d_\gamma}$ are outputs (i-indexed)
- Red is a reduction operation : Red $= \sum$, *Min*/*Argmin*, *KMin*/*ArgKmin*, *LogSumExp*, *SoftMax*,...
- $F : \mathbf{R}^{d_\theta} \times \mathbf{R}^{d_x^1} \times \cdots \times \mathbf{R}^{d_x^p} \times \mathbf{R}^{d_y^1} \times \cdots \times \mathbf{R}^{d_y^q} \to \mathbf{R}^{d_\gamma}$ is any function.
- goal: compute the reduction for any type of function, parallelized on GPU if available, and staying linear in memory, i.e. never storing the $n_x \times n_y$ evaluations of the function F.
- bonus : get automatic differentiation of these reduction operations.

Memory architecture of GPU devices



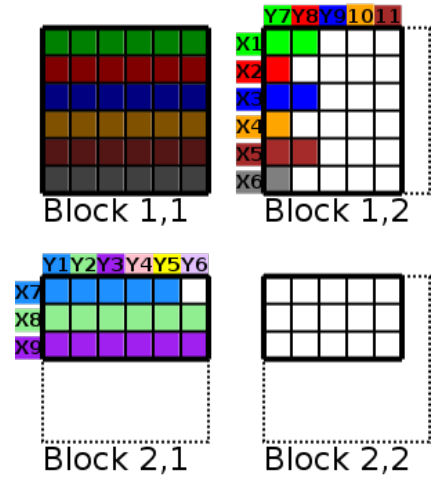
- Host memory: usual computer RAM memory, outside the device (several Gb)
- Global and constant memories : global memory of the GPU (several Gb)
- Shared memory : memory allocated to each block of threads (a few Kb)
- local memory and registers : memory allocated to each thread (a few Kb)

Tiled reductions on GPU devices: 1D and 2D schemes



1D scheme for tiled reduction, executed in parallel by all threads in block I

- load θ into local memory
- if $i < n_x$: load x_i into local memory
- initialize γ_i
- for each block J:
- for each index j in block J:
  - if $j < n_y$: load y_j into shared memory
  - synchronize threads
  - if $i < n_x$: compute $Red_{j \in J} F(x_i, y_j)$ and accumulate result into γ_i



2D scheme for tiled reduction, executed in parallel by all threads in block (I,J)

- load θ into local memory
- if $i < n_x$: load x_i into local memory
- initialize γ_i^J
- for each index j in block J:
  - if $j < n_y$: load y_j into shared memory
  - synchronize threads
  - if $i < n_x$: compute $Red_{j \in J} F(x_i, y_j)$ and accumulate result into γ_i

Eventually : needs additional step to reduce further γ_i^J from all blocks J to γ_i.

Automatic differentiation

- formula F is expressed as a combination of elementary operations, e.g. $F = Exp((x, y))$ and expanded internally in the code using C++ templates:

$$F = Exp < Scalprod < X, Y >>$$

- Each elementary operation has a corresponding gradient operation expressed via the same template construction process. So the chain rule calculus is done via composition of templates at compilation time.

$$\begin{aligned} Grad < Exp < U >, X, E > \quad &\Rightarrow \quad Grad < U, X, Mult < Exp < U >, E >> \\ &\Rightarrow \quad \text{expanded further via gradient of} \\ &\qquad\quad U = Scalprod < X, Y > \end{aligned}$$

The Log-Sum-Exp reduction

$$\gamma_i = \log \sum_{j=1}^{n_y} e^{F(\theta, x_i, y_j)}$$

- a stable reduction is obtained by keeping track of the maximal value of F during the reduction, in order to take only exponential of negative numbers. So in fact the actual reduction computed internally is

$$(m_i, s_i) = (\max_j F_{ij}, \sum_j e^{F_{ij} - m_i})$$

  which is computed through the following scheme:
  - initialize $m_i = -\infty$, $s_i = 0$,
  - loop through j :

$$\begin{cases} (m_i, s_i) \leftarrow (m_i \ , \ s_i + e^{F_{ij} - m_i}) \text{ if } F_{ij} < m_i, \\ (m_i, s_i) \leftarrow (F_{ij} \ , \ s_i e^{m_i - F_{ij}} + 1) \text{ otherwise.} \end{cases}$$

- Finally just compute $\gamma_i = m_i + \log s_i$.

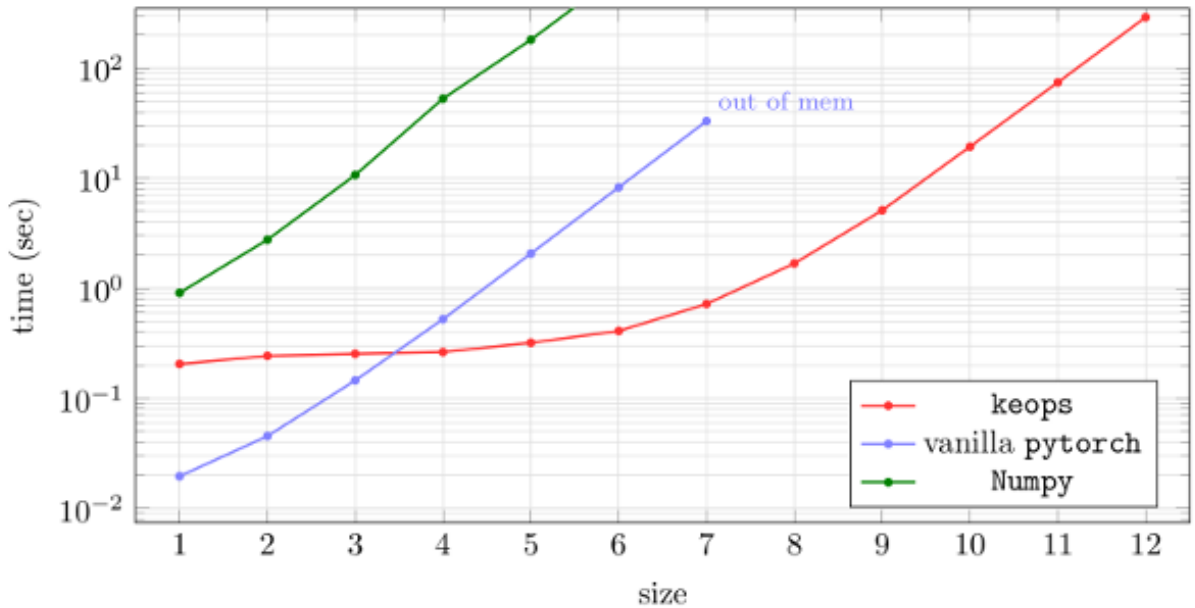
KeOps library

- code in C++/Cuda with Matlab, Numpy and PyTorch bindings
- `http://www.kernel-operations.io`
- pip install pykeops
- automatic differentiation is integrated into the automatic differentiation of PyTorch.
- operations in $O(n^2)$ (but soon : implementation for sparse reductions) but parallelized on Gpu, and $O(n)$ memory cost.

KeOps library

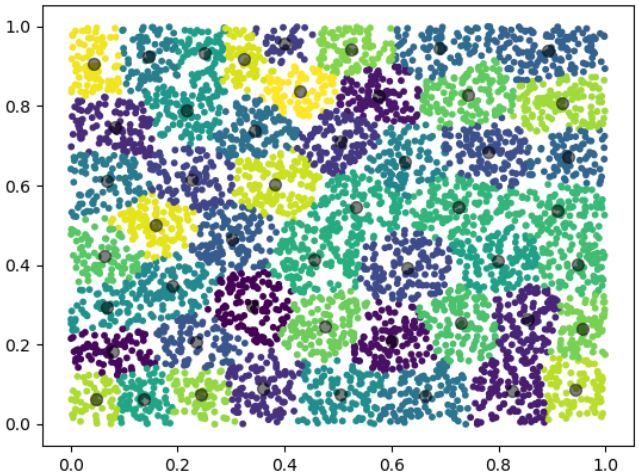
Performances for gaussian convolution

$$\gamma_i = \sum_j e^{-\frac{\|x_i - y_j\|^2}{\sigma^2}} \beta_j$$

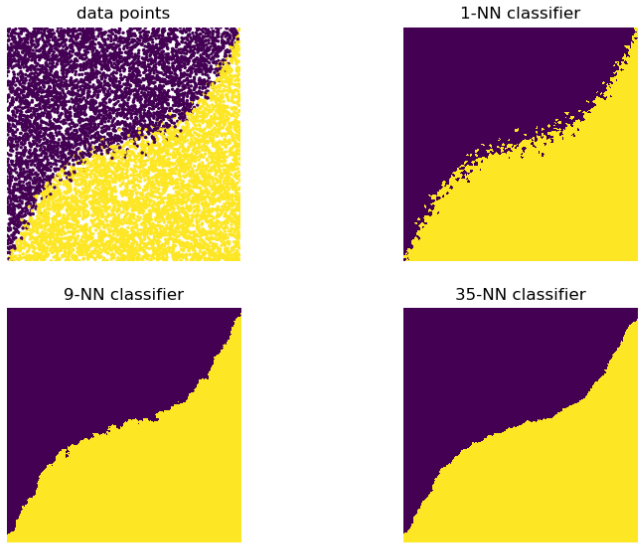


Time needed to compute 200 Gaussian kernel products on a Tesla P100 GPU.
At size i we have $M = 200 \times 2^i$ and $N = 300 \times 2^i$.

Applications to standard algorithms

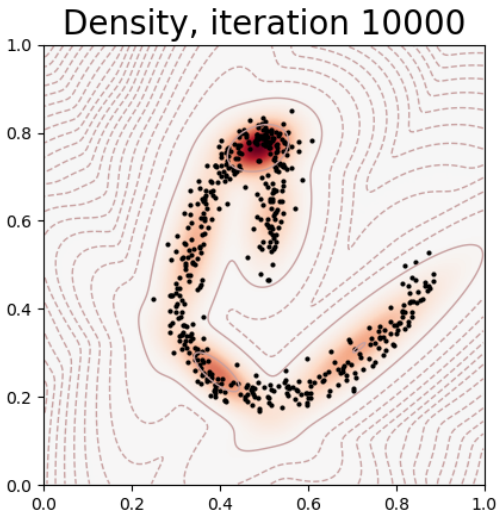


k-means clustering (reduction: ArgMin)
$K = 5.10^3, N = 5.10^5, D = 60 : 0.12s/it$

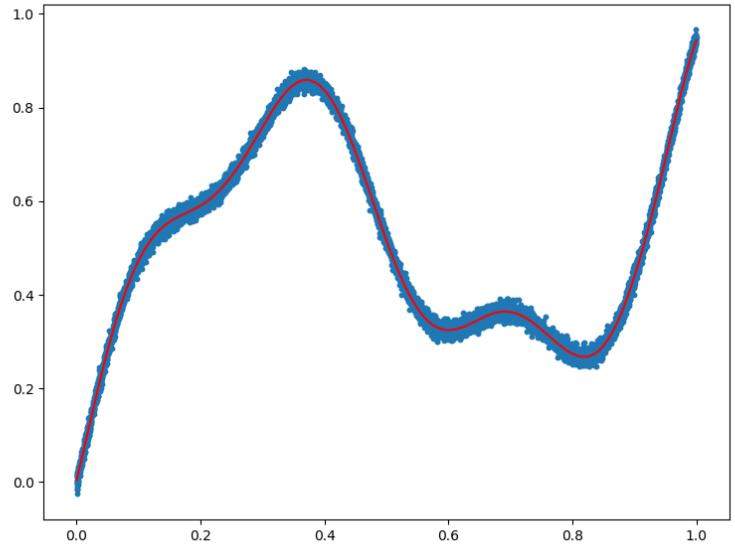


k-NN classification (reduction: ArgKMin)
$N_{train} = 10^4, N_{test} = 4.10^6 : 0.24s$

Applications to standard algorithms

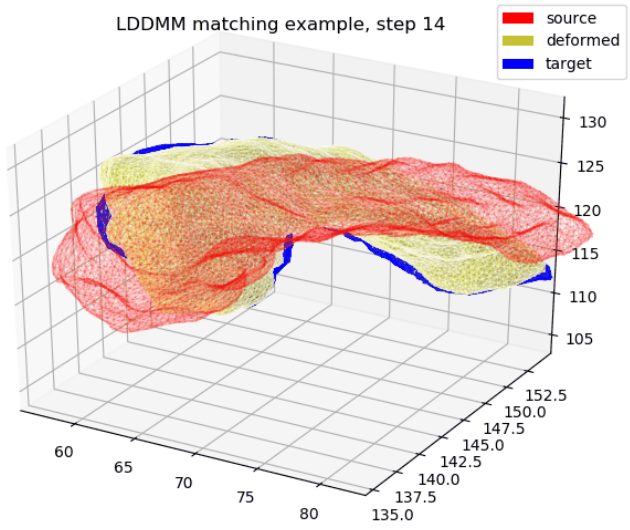


gaussian mixture fitting
(reduction: LogSumExp)
73s for 10^5 iterations

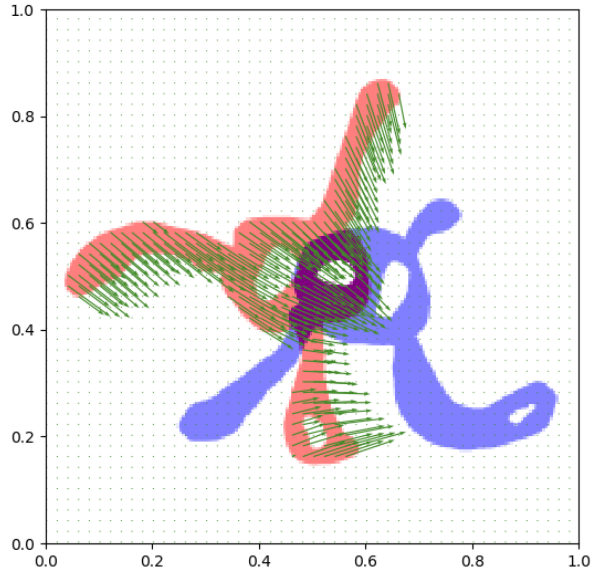


RBF interpolation
(reduction:Sum + conjugate gradient solver)
$N = 10^4$: $0.17s$ for solving

Applications to shapes



LDDMM/varifolds
(reduction: Sum)
6.10^3 points: $89s$ for optimization



regularized optimal transport
(reduction:LogSumExp)
4.10^3 points: $0.44s$ for 30 iters

- check also Jean's new toolbox : GeomLoss
  `http://www.kernel-operations.io/geomloss` for implementation and comparison of various data fidelity terms between points sets.