

# How to Approximate a Blob with a Curve

## *An Overview of a Gradient-based Numerical Algorithm*

Forest Kobayashi (UBC)

Based on joint work with Jonathan Hayase (UW) and Young-Heon Kim (UBC)

Monday, June 30<sup>th</sup> 2025

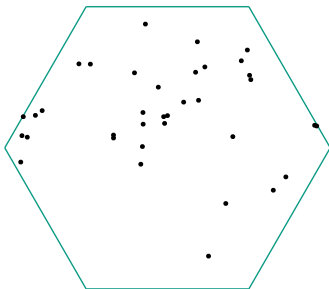
# Disclaimers

- ▶ “Just” curves?
  - Algorithm: yes, for now...
  - Theory: more general
- ▶ Many statements **informal**
  - → Focus on big picture and algorithm

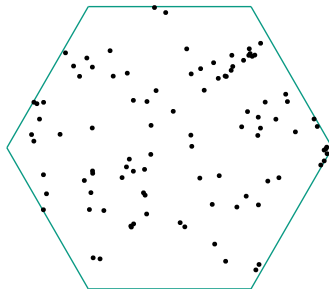
# Summary

- ▶ Goal: Efficiently approximate  $n$ -d measures via  $m$ -dimensional sets ( $m < n$ )
  - “Approximate” in what sense?
  - “Efficient” in what sense?
- ▶ Outline of sections:
  - Motivating examples
  - Define problem
  - Gradient structure
  - Algorithm

# Ex. 1: Learning noisily-embedded manifolds



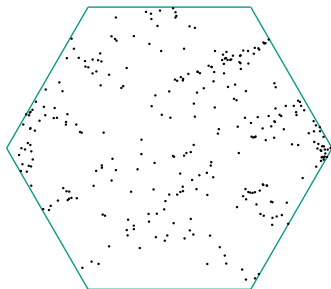
(a) A few data points...



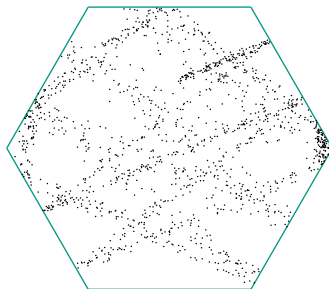
(b) ...and a few more.

Figure 1: Seemingly-unstructured data in a hexagon.

# Ex. 1 (cont.): Learning noisily-embedded manifolds



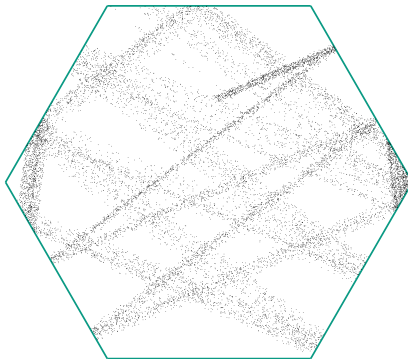
(a) More...



(b) ...and more...

Figure 1: With more points, a picture begins to emerge.

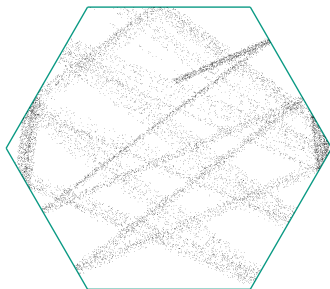
# Ex. 1 (cont.): Learning noisily-embedded manifolds



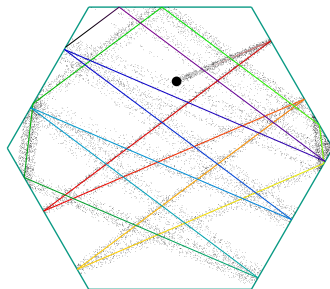
(a) ...and more!

Figure 1: With more points, a picture begins to emerge.

# Ex. 1 (cont.): Learning noisily-embedded manifolds



(a) The data.



(b) Extracted structure.

Figure 1: “Averaging” the tubular segments in the data.

## Ex. 2: Approximating a properly-higher-dim. $\rho$

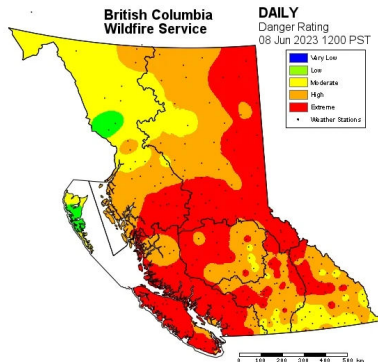


Figure 2: Wildfire danger map

## Ex. 2: Approximating a properly-higher-dim. $\rho$

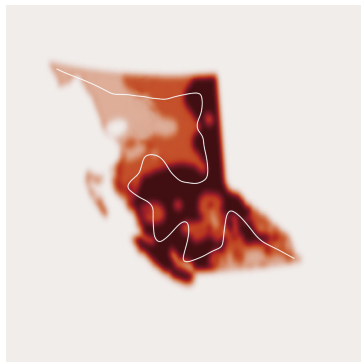
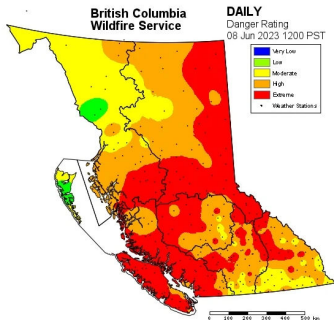


Figure 2: Example trajectory. (Is it “good?”)

## Summary of some applications

- ▶  $m = 1, n \in \{2, 3\}$ : Routing problems (wildfire drone, package delivery routes, “continuous” TSP)
- ▶  $m = 1, n \geq 2$ : Trajectory inference (e.g. cryo-EM, scRNA-seq)
- ▶  $m = 2, n = 3$ : Catalytic surface design
- ▶  $1 \leq m \ll n$ : Certain generative learning problems

# Setting up the Problem

► Source:

■  $X \subseteq \mathbb{R}^m$  (compact)

► Target:

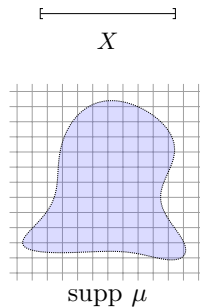
■  $\mu \in \mathcal{P}_{\text{cpt}}(\mathbb{R}^n)$

► Optimization variable:

■  $f : X \rightarrow \mathbb{R}^n$  (cont.)

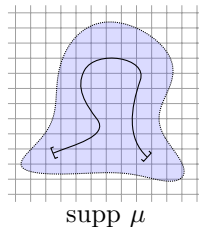
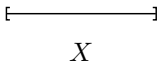
# Setting up the Problem

- ▶ Source:
  - $X \subseteq \mathbb{R}^m$  (compact)
- ▶ Target:
  - $\mu \in \mathcal{P}_{\text{cpt}}(\mathbb{R}^n)$
- ▶ Optimization variable:
  - $f : X \rightarrow \mathbb{R}^n$  (cont.)



# Setting up the Problem

- ▶  $X \subseteq \mathbb{R}^m$
- ▶  $\mu \in \mathcal{P}_{\text{cpt}}(\mathbb{R}^n)$
- ▶  $f : X \rightarrow \mathbb{R}^n$



## How “close” is $f(X)$ to $\mu$ ?

- ▶ Want OT cost, but  $f(X)$  a set
- ▶ Idea:

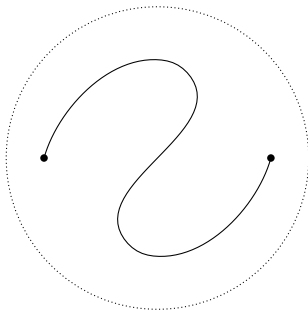
$$\mathcal{G}_p(f; \mu) = \inf_{\nu \in \mathcal{P}(f(X))} \mathbb{W}_p^p(\mu, \nu)$$

- ▶ Equivalently [7, Prop. 2.7]:

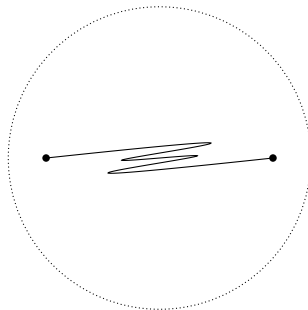
$$\begin{aligned} \mathcal{G}_p(f; \mu) &= \int_{\mathbb{R}^n} d^p(\omega, f(X)) \, d\mu(\omega) \\ &= \mathbb{E}_\mu[d^p(\omega, f(X))] \end{aligned}$$

# Visualization

Q: Which shape better “fills” unit disk?



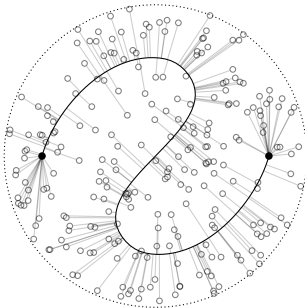
(a) One candidate



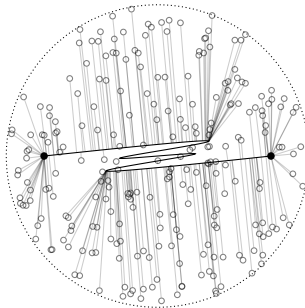
(b) ...And another

# Visualization

Q: Which shape better “fills” unit disk?



(a) A “good” filling




(b) A “bad” filling

# Quantifying complexity of $f$

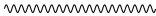
- ▶ No complexity constraint  $\rightarrow$  degeneracy
- ▶ We use:

$$\begin{aligned}\mathcal{C}(f) &:= \|f\|_{W^{k,q}(X,\mathbb{R}^n)} \\ &= \left( \sum_{j=1}^n \sum_{|\alpha| \leq k} \|D^\alpha f_j\|_{L^q}^q \right)^{1/q}.\end{aligned}$$


- ▶ Examples:



(a) Cheap  $f$



(b) Expensive  $f$



(c) Expensive  $f$

# The “best” $f$

- ▶ For  $\lambda > 0$ , minimize

$$\mathcal{G}_\lambda(f; \mu) := \mathcal{G}_p(f; \mu) + \lambda \mathcal{C}(f).$$

- ▶ Computing global solutions:
  - “Probably” NP-Hard
  - For  $k, q = 1$ , recover TSP as  $\lambda \rightarrow 0$ .
- ▶ How about local improvements?

# Improving an $f$

## Theorem (Informal)

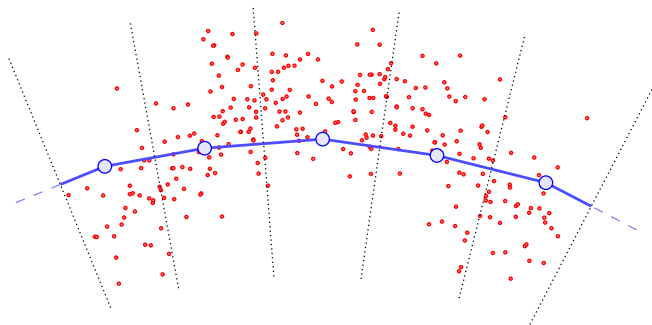
*With some technical hypotheses, can get a vector field  $F$  and measure  $\nu \in \mathcal{P}(f(X))$  such that for all  $\xi \in C(f(X); \mathbb{R}^n)$ ,*

$$\lim_{\varepsilon \rightarrow 0} \frac{\mathcal{I}_p(f + \varepsilon \xi; \rho) - \mathcal{I}_p(f; \rho)}{\varepsilon} = \int_{f(X)} -\langle F, \xi \rangle d\nu.$$

- ▶  $F$  like (negative) “gradient” (sans regularity issues)
- ▶  $-\langle F, \xi \rangle_{L^2(\nu)}$  like directional derivative
- ▶  $\nu$  very simple;  $F$  more complicated

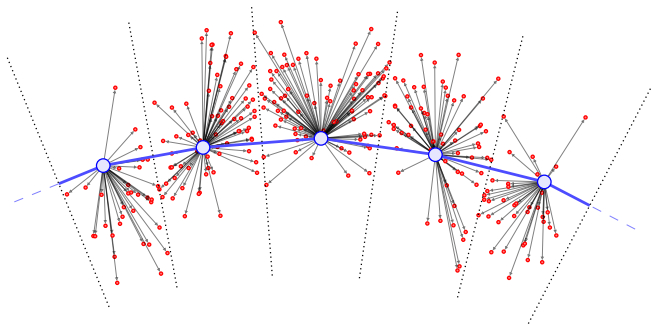
# What is $F$ ?

Discrete case easier to visualize:



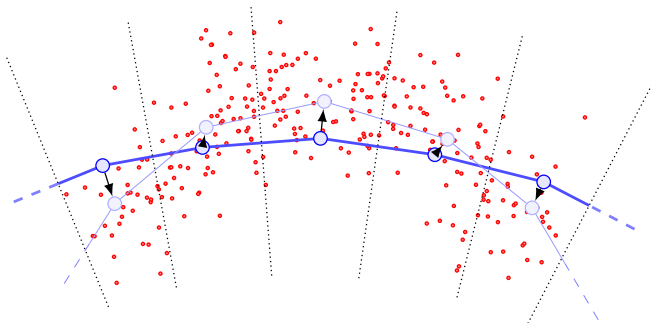
# What is $F$ ?

Discrete case easier to visualize: Compute  $(p - 1)$  barycenter



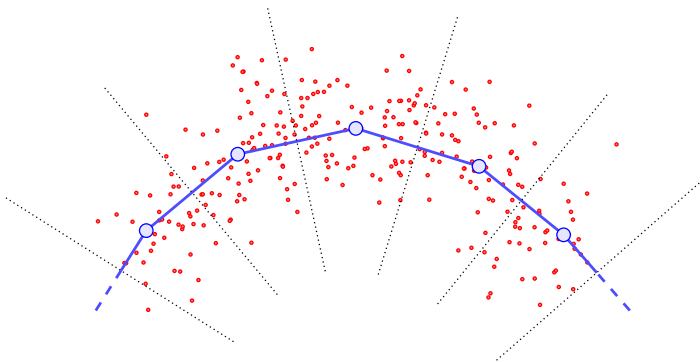
# What is $F$ ?

Discrete case easier to visualize: Compute  $(p - 1)$  barycenter



# What is $F$ ?

Discrete case easier to visualize:



# Visualizing $F$ in cont. case

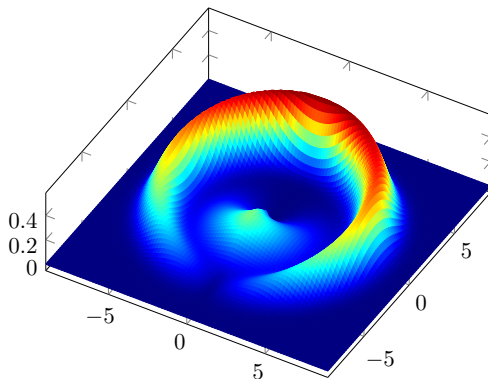


Figure 5: An example  $\mu$

# Visualizing $F$ in cont. case

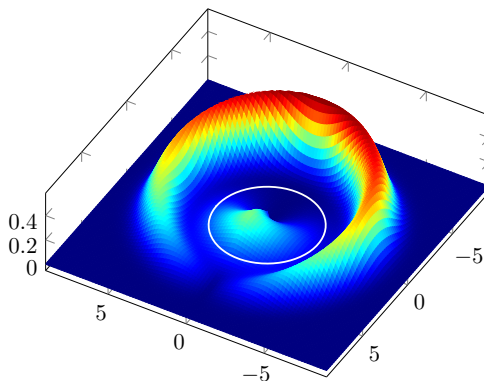


Figure 5: An example  $f$  approximating  $\mu$

# Visualizing $F$ in cont. case

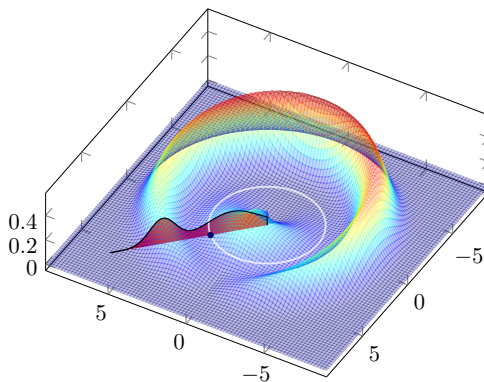
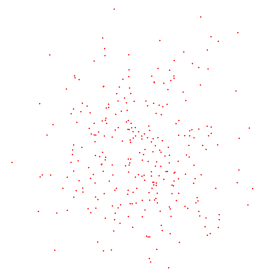
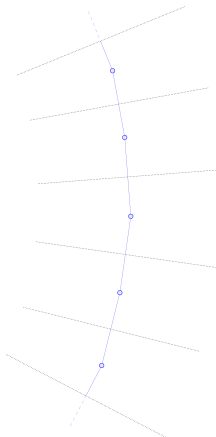


Figure 5: Slicing

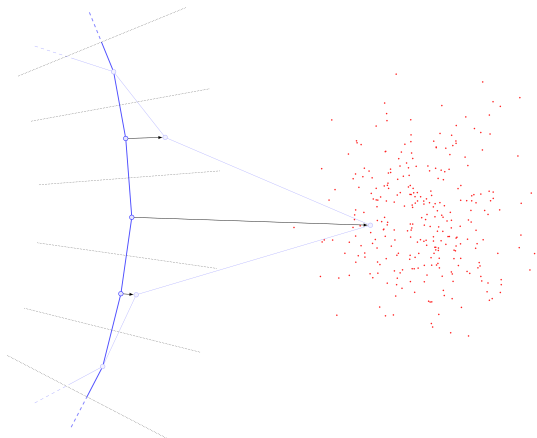
# Overview

- ▶ Big picture:
  - Discretize  $f$
  - Compute  $F$  ( $= -\nabla \mathcal{G}_p$ ) and  $-\nabla \mathcal{C}$
  - Optimize to get step size  $\eta$
  - $f \leftarrow f + \eta(F - \lambda \nabla \mathcal{C})$
- ▶ Challenges:
  - C1. Discretization of  $f$  is delicate
  - C2. Efficient Voronoi cell assignments for  $F$
  - C3. How to compute  $-\nabla \mathcal{C}$ ?

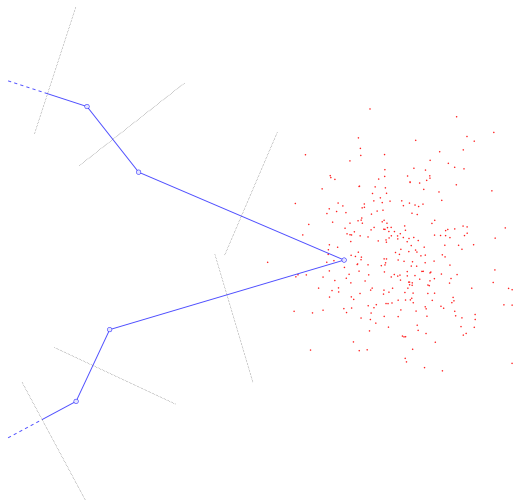
# C1: Need evenly-spaced samples of $f$



# C1: Need evenly-spaced samples of $f$



# C1: Need evenly-spaced samples of $f$



# C1: Evenly-spaced samples of $f$ —our method

- ▶ After each iteration:
  - Fit cubic spline  $f$  to samples
  - Define: Arclength functional

$$L(f; a, b) = \int_a^b \sqrt{\sum_{j=1}^n (\dot{f}_j)^2} dt$$

- Compute total length  $\ell$
- Get evenly-spaced values  $l_i$  on  $[0, \ell]$
- For given  $l_i$ , want  $t_i$  such that  $L(f; 0, t_i) \approx l_i \dots$
- Binary search then polish w/ Newton's method

# C1: Evenly-spaced samples of $f$ , cont.

- ▶ Need: Efficient query method for arclength  $L(\mathbf{f}; 0, t)$ 
  - Quadrature a bit slow
  - [5] reduces  $L(\mathbf{f}; 0, t)$  to an elliptic integral explicitly solved in [3]
  - Reduction: need factorization into real quadratics

$$\sum_{j=1} (\dot{\mathbf{f}}_j)^2 = Q_1(t)Q_2(t),$$

(exists if  $\dot{\mathbf{f}} \neq 0$ ; use fast root-finding algorithms)

- When  $n = 2$ : Get a stable, analytic formula via trick
$$q_1^2 + q_2^2 = (q_1 - iq_2)(q_1 + iq_2)$$
- ▶ End result: Fast resampling of  $f$

## C2: Quickly computing $F$

- ▶ Continuous data: Special case  $\mu$  uniform,  $\text{supp } \mu$  PL
  1. Explicitly compute Voronoi diagram
  2. Fast triangular decomposition of  $\text{supp } \mu$
  3. Triangles:  $F$  has analytic formula in terms of hypergeometric function
  4. Sum results
- ▶ Discrete data: Need faster Voronoi assignments
  - CPU-method: spatial-acceleration datastructures
  - GPU-method: brute-force

## C2: CPU method, acceleration datastructures

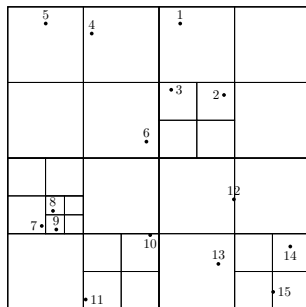


Figure 6: Example orthant subdivision for  $M = 15$  samples of  $\text{Unif}([0, 1]^2)$ .

## C2: CPU method, acceleration datastructures

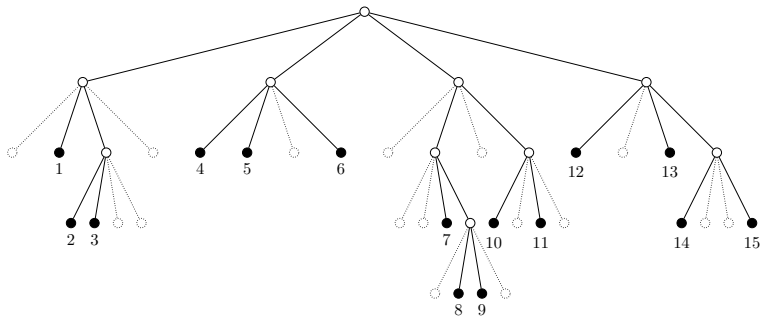


Figure 6: The tree datastructure encoding the quadrants shown previously.



## C2: CPU method, acceleration datastructures

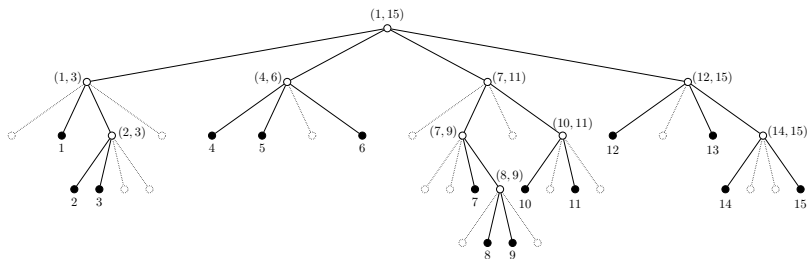


Figure 6: Annotate nodes of tree with range of descendants (also, diameter).

## C2: CPU method, Performing the Voronoi Assignments

- ▶ With trees above, construct further  $k$ -d tree
- ▶ Core loop:

1. Take data pt.  $v_i$
2. Query  $k$ -d tree for closest samples  $y_i^0, y_i^1$  of  $f$
3. Let  $r_i = |d(v_i, y_i^0) - d(v_i, y_i^1)|$  and compute

$$j_i := \lfloor \log_2 (r_i / \text{diam}(\text{node for } v_i)) \rfloor$$

4. Slice trailing  $j_i$  entries off `lex_code`( $v_i$ ); yields ancestor  $a_i$
5. Fact: All descendants  $(v_i, \dots, v_{i'})$  of  $a_i$  in same Voronoi cell as  $v_i$
6. Assign all  $(v_i, \dots, v_{i'})$ ; let  $i = i' + 1$  and loop

## C2: CPU method, Pictures

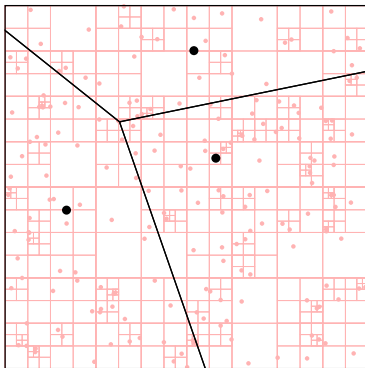


Figure 7: Starting configuration.

## C2: CPU method, Pictures

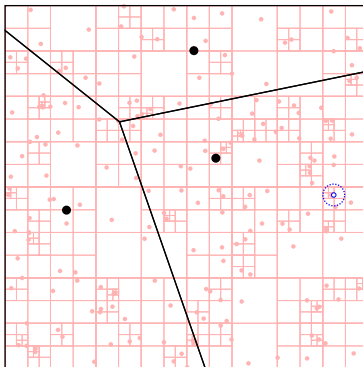


Figure 7: Select  $v_i$ .

## C2: CPU method, Pictures

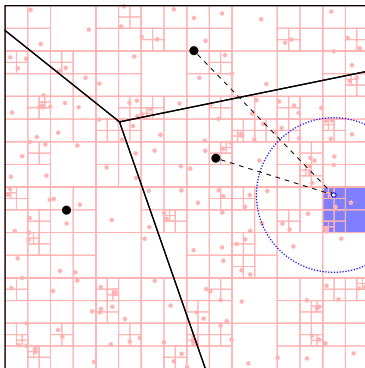


Figure 7: Compute ancestor  $a_i \subseteq B_{r_i}(v_i)$

## C2: CPU method, Benchmarking

M	Mean exec. time $\pm \sigma$ (ms)		Median (ms)		Mem. est. (MiB)		Allocs. est. (#)	
	Our alg.	$k$ -d tree	Our alg.	$k$ -d tree	Our alg.	$k$ -d tree	Our alg.	$k$ -d tree
$2.16 \cdot 10^5$	$43 \pm 4$	$45 \pm 11$	43	39	21.03	32.96	521697	864014
$5.12 \cdot 10^5$	$76 \pm 7$	$116 \pm 25$	76	107	36.70	78.13	879072	2048014
$8.00 \cdot 10^5$	$102 \pm 5$	$190 \pm 35$	103	184	48.83	122.07	1143060	3200014
$1.00 \cdot 10^6$	$114 \pm 6$	$242 \pm 92$	115	221	56.16	152.59	1297350	4000014
$1.20 \cdot 10^6$	$126 \pm 7$	$307 \pm 148$	127	281	62.81	181.11	1433556	4800014
$1.50 \cdot 10^6$	$152 \pm 13$	$398 \pm 172$	149	383	72.39	228.88	1627029	6000014
$1.80 \cdot 10^6$	$161 \pm 9$	$493 \pm 273$	164	461	81.83	274.66	1798515	7200014
$2.00 \cdot 10^6$	$168 \pm 10$	$550 \pm 316$	169	514	86.72	305.18	1906332	8000014
$2.50 \cdot 10^6$	$196 \pm 12$	$711 \pm 477$	198	619	99.70	381.47	2150112	10000014
$5.00 \cdot 10^6$	$300 \pm 23$	$1587 \pm 1201$	304	1304	155.41	762.94	3123312	20000014
$1.00 \cdot 10^7$	$467 \pm 30$	$3370 \pm 2805$	471	2838	245.37	1490	4500033	40000014

**Table 1:** Benchmarking for  $\mu = \text{Unif}([0, 1]^2)$ .  $M$ : # input data pts.  
 $N = 100$ : # Voronoi cell roots (also uniform).

## C3: Computing Sobolev gradient $-\nabla\mathcal{C}$

- ▶ In C1 we fit a cubic spline  $f$
- ▶ In general:  $f \notin W^{k,q}(X; \mathbb{R}^n)$
- ▶  $\Rightarrow$  need new spline of order  $k + 2$ .
- ▶ Approach:
  1. Get B-Spline basis; compute  $C^k(X; \mathbb{R}^n)$  spline interpolating resampled  $f$
  2. Derivatives of all orders linear in spline coefficients
  3. Spline coefficients linear in fitted points
  4. Linear relationships can be precomputed very quickly (inverse of  $(k + 2)$ -width banded matrix) *once* per loop
  5.  $\Rightarrow$  fast computation of derivatives of all orders

# Bottom Line

## ► Runtime:

### ■ Resampling $f$ evenly:

- Fast for  $n = 2$
- Good for  $n > 2$

### ■ Computing $F = -\nabla \mathcal{G}_p$ :

- Special continuous case: Very fast if  $\text{supp } \mu$  convex; decent otherwise
- Discrete case: Acceptable on CPU; extremely fast on GPU

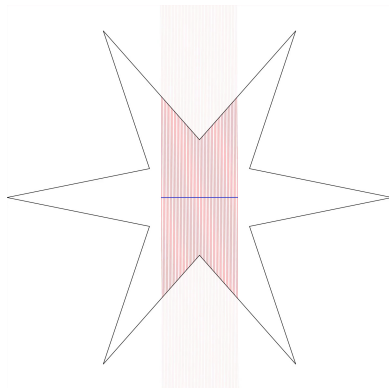
### ■ Approximating $-\nabla \mathcal{C}$ :

- *Very fast*

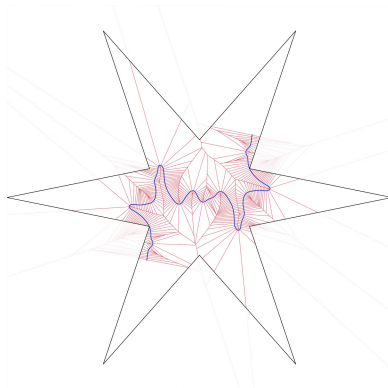
## ► Memory:

### ■ Lightweight except discrete CPU method for $F = -\nabla \mathcal{G}_p$

# Example numerics

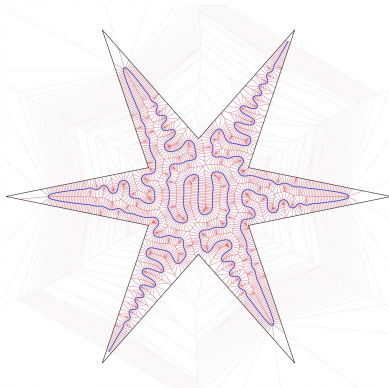


(a)  $i = 1$

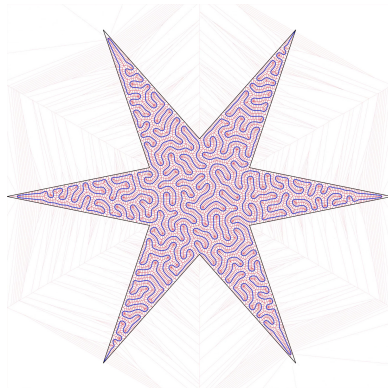


(b)  $i = 150$

# Example numerics



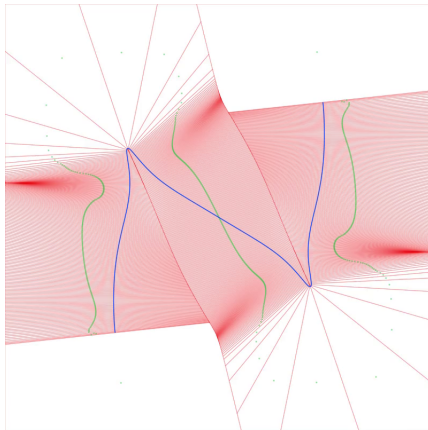
(a)  $i = 300$



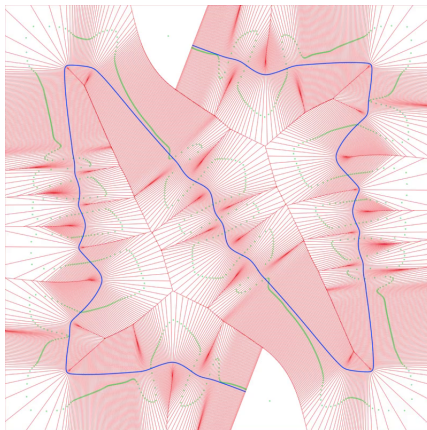
(b)  $i = 1000$

## More simulations

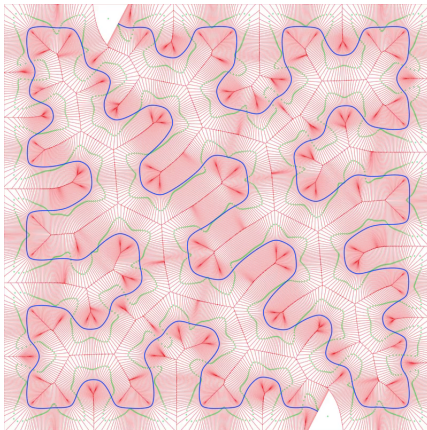
## More simulations



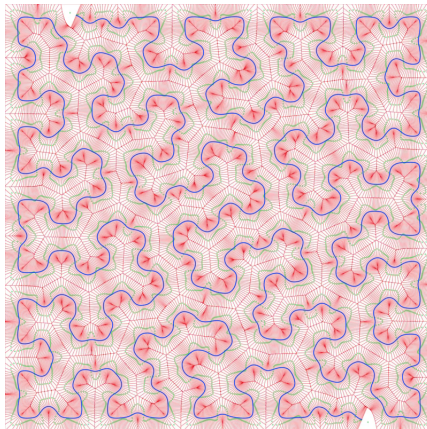
## More simulations



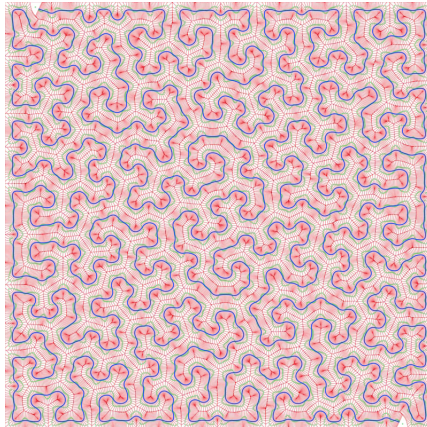
## More simulations



## More simulations



## More simulations



Thank you for your attention!

# References I

- [1] Giuseppe Buttazzo and Eugene Stepanov. “Optimal transportation networks as free Dirichlet regions for the Monge-Kantorovich problem”. In: *Annali della Scuola Normale Superiore di Pisa - Classe di Scienze* (2003).
- [2] Giuseppe Buttazzo, Edouard Oudet, and Eugene Stepanov. “Optimal Transportation Problems with Free Dirichlet Regions”. In: *Variational Methods for Discontinuous Structures*. 2002.
- [3] BC Carlson. “A table of elliptic integrals: Two quadratic factors”. In: *mathematics of computation* (1992).
- [4] Sylvain Delattre and Aurélie Fischer. “On principal curves with a length constraint”. In: *Annales de l’Institut Henri Poincaré, Probabilités et Statistiques* (Aug. 2020).
- [5] Hartmut Henkel. *Calculating the Cubic Bézier Arc Length by Elliptic Integrals*. 2014.

## References II

- [6] Slav Kirov and Dejan Slepčev. “Multiple Penalized Principal Curves: Analysis and Computation”. en. In: *Journal of Mathematical Imaging and Vision* (Oct. 2017).
- [7] Forest Kobayashi, Jonathan Hayase, and Young-Heon Kim. “Monge-Kantorovich Fitting With Sobolev Budgets”. In: *arXiv* (Sept. 2024). eprint: 2409.16541.
- [8] A. Lemenant. “A presentation of the average distance minimizing problem”. In: *J. Math. Sci.* (Mar. 2012).
- [9] Xin Yang Lu and Dejan Slepčev. “Average-distance problem with curvature penalization for data parameterization: regularity of minimizers”. en. In: *ESAIM: Control, Optimisation and Calculus of Variations* (2021).
- [10] E. Paolini and E. Stepanov. “Qualitative Properties of Maximum Distance Minimizers and Average Distance Minimizers in  $\mathbb{R}^n$ ”. In: *Journal of Mathematical Sciences* (July 2004).

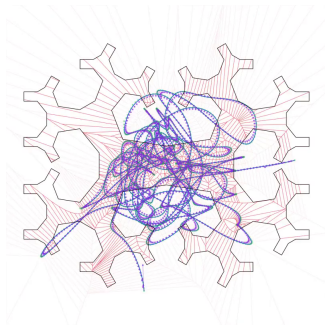
## References III

- [11] Eugene Stepanov. “Partial Geometric Regularity of Some Optimal Connected Transportation Networks”. In: *J. Math. Sci.* (Jan. 2006).
- [12] Wenping Wang, Helmut Pottmann, and Yang Liu. “Fitting B-spline curves to point clouds by curvature-based squared distance minimization”. In: *ACM Trans. Graph.* (Apr. 2006).

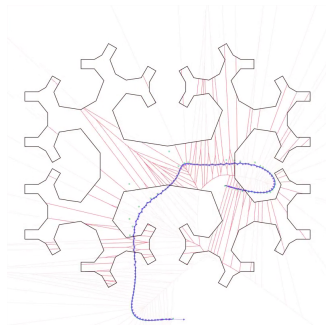
# Generative Learning

- ▶ Soft Proposal: this problem gives loosely-analogous toy problem for some black-box models
- ▶ Like (regularized) WGAN, but
  - Learns only support of approximating measure (not parametrization)
  - Regularization on *generator*, not critic
- ▶ Gives a possible interpretation for what regularization “does”

# A regularized “generator”

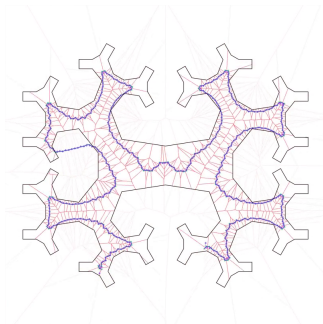


(a)  $i = 1$

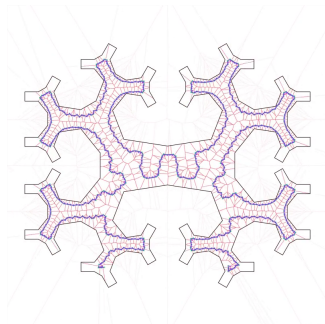


(b)  $i = 110$

# A regularized “generator”

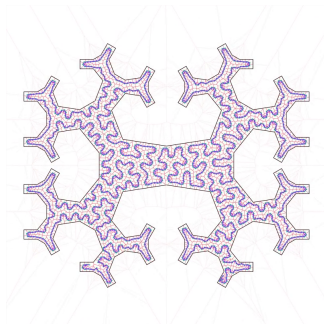


(a)  $i = 800$



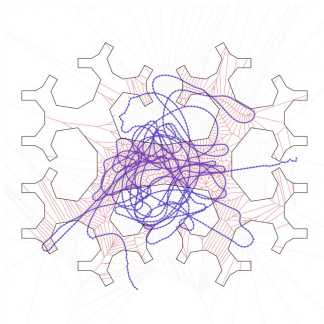
(b)  $i = 1120$

# A regularized “generator”

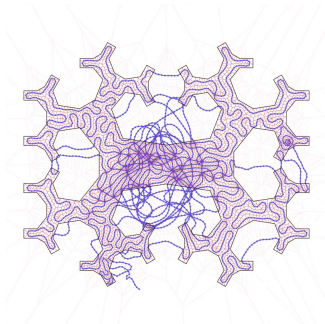


(a)  $i = 1850$

## (cont.): Unregularized Generator

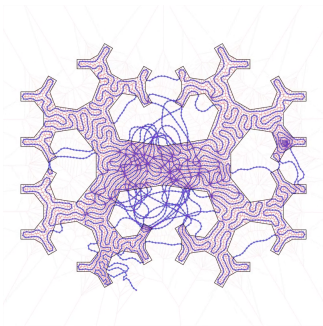


(a)  $i = 110$

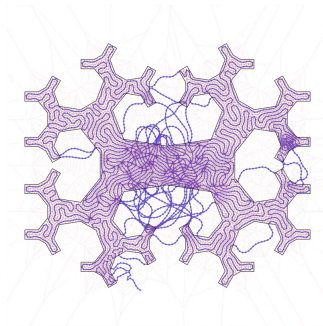


(b)  $i = 800$

## (cont.): Unregularized Generator

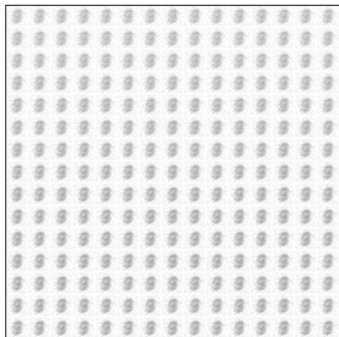


(a)  $i = 1120$

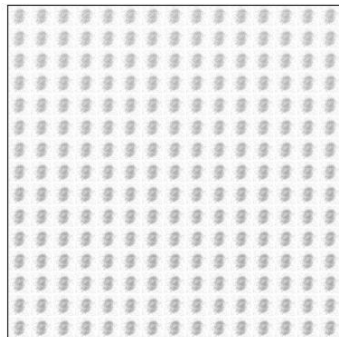


(b)  $i = 1850$

# Proof of concept: training on MNIST



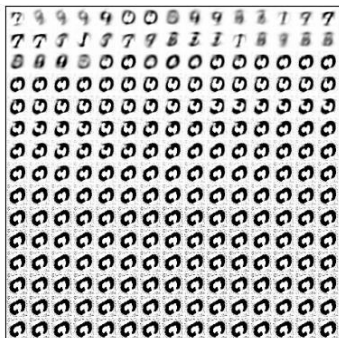
(a) 1 epoch (wd)



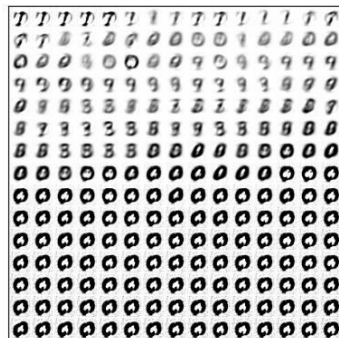
(b) 1 epoch (reg)

Figure 11:  $f$  sampled at  $15^2$  uniformly-spaced points on  $[0, 1]$

# Proof of concept: training on MNIST



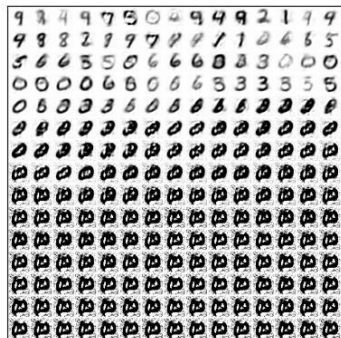
(a) 30 epochs (wd)



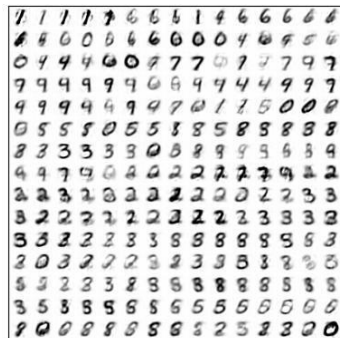
(b) 30 epochs (reg)

Figure 11:  $f$  sampled at  $15^2$  uniformly-spaced points on  $[0, 1]$

# Proof of concept: training on MNIST



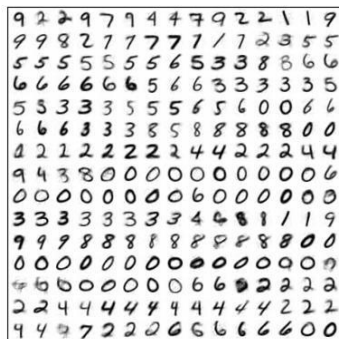
(a) 100 epoch (wd)



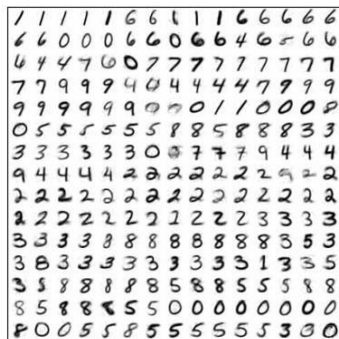
(b) 100 epoch (reg)

Figure 11:  $f$  sampled at  $15^2$  uniformly-spaced points on  $[0, 1]$

# Proof of concept: training on MNIST



(a) 1000 epochs (wd)



(b) 1000 epochs (reg)

Figure 11:  $f$  sampled at  $15^2$  uniformly-spaced points on  $[0, 1]$