Unsupervised Signal Demixing

Optimization and Applications

20 Jan 2026 | Burnaby, BC

Nicholas Richardson

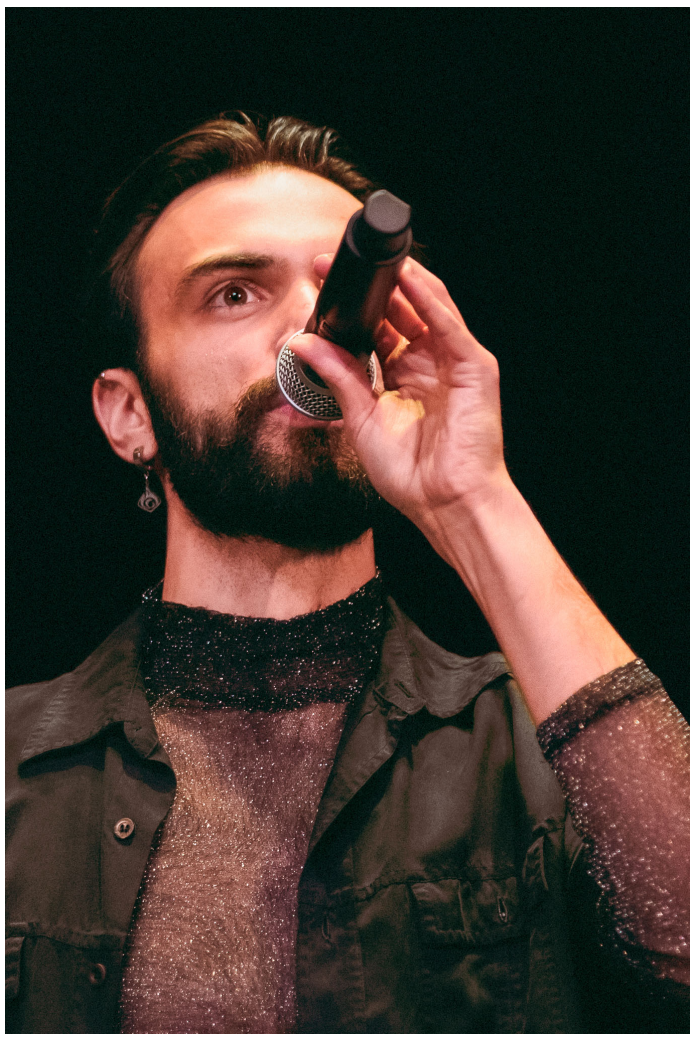
Department of Mathematics

THE UNIVERSITY OF BRITISH COLUMBIA

SFU SIMON FRASER UNIVERSITY

INSTITUTE OF APPLIED MATHEMATICS
University of British Columbia

About Me



- BMath & MMath at the University of Waterloo (2020 & 2022)
- PhD Candidate supervised by Michael P. Friedlander at UBC
- Research in Optimization, Signal Processing, Tensor Factorization
- *Fun fact*: I sing & beatbox with The Northwest Collective!
- *Website*: https://njericha.github.io

Overview

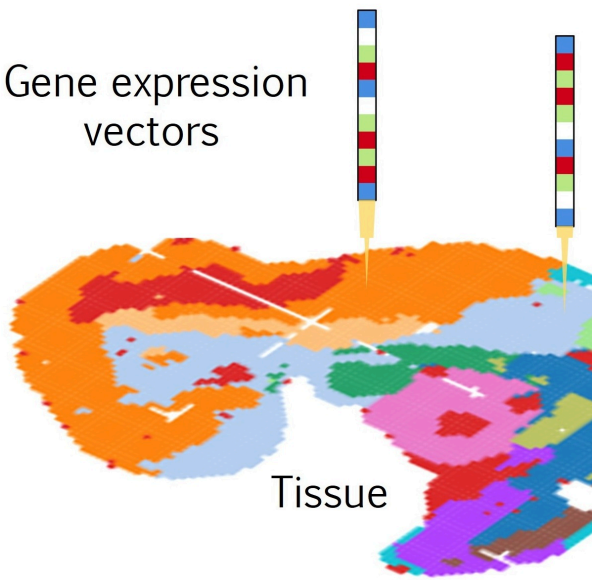
- The unmixing problem
- Applications like geology, biology, and music
- Example method one: sparse random features
- Example method two: constrained tensor factorization

The Problem

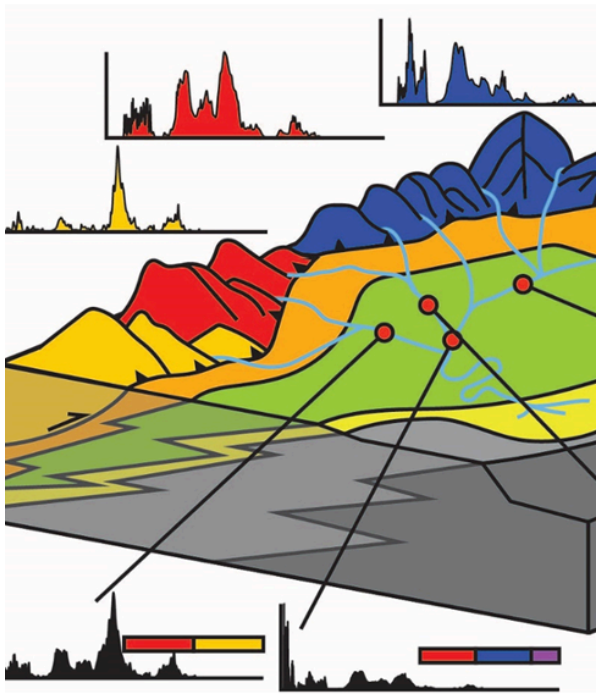
I’m getting mixed signals…

Many real world applications involve mixtures of data

Songs are mixtures of different instruments



Gene counts are influenced by cell types

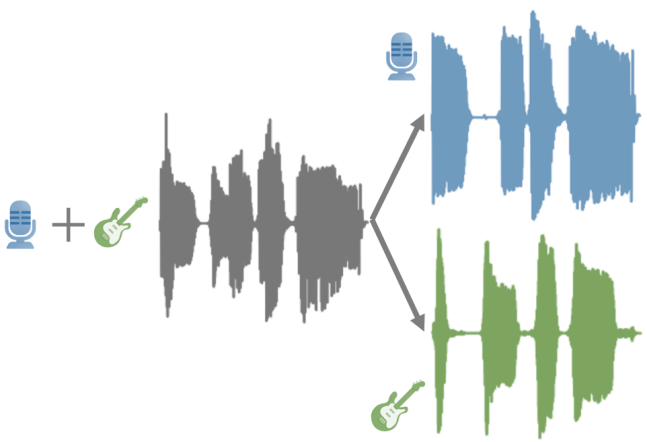


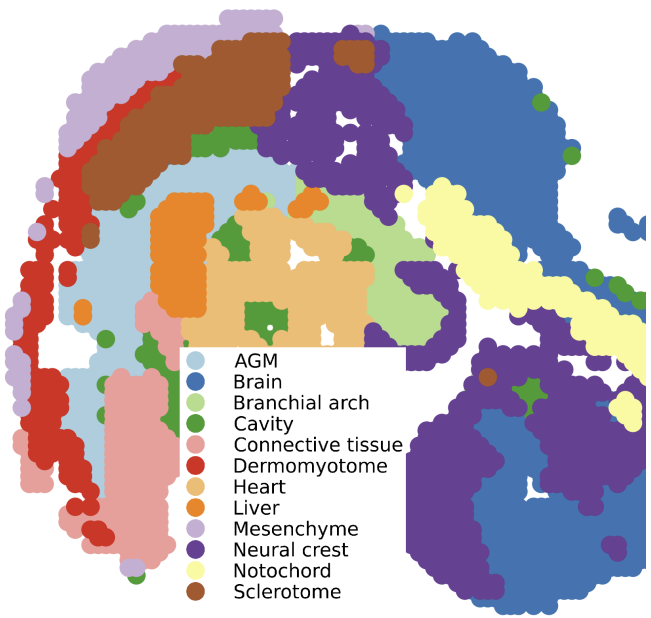
Rocks are mixtures of mineral sources

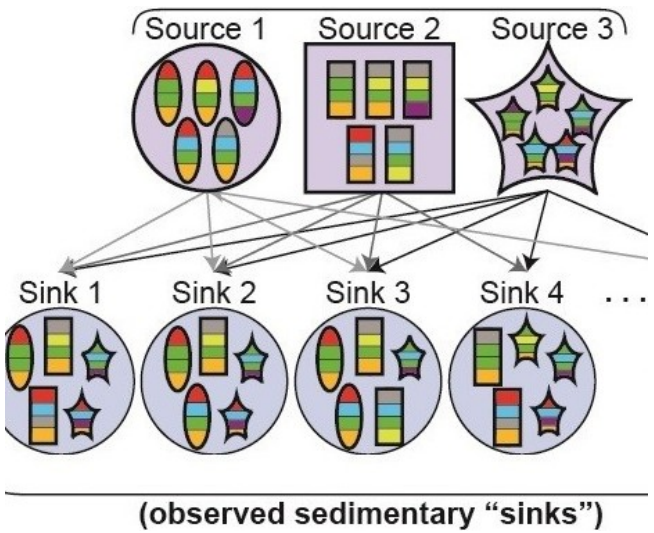
What do we want?

Goal: Given data *without labeled examples*, determine the underlying sources.

Why? For IDing sources, pre-processing, de-noising







Setting 1: Single Signal Separation

Unmix a signal $\mathbf{y} \in \mathbb{R}^n$ into sources $\mathbf{s}_r \in \mathbb{R}^n$ for $r = 1, \ldots, R$:

$$\mathbf{y} = \mathbf{s}_1 + \mathbf{s}_2 + \cdots + \mathbf{s}_R$$

Ill-defined without assumptions on what $\mathbf{s}_r$ should look like.

- Should they be sparse?
- Low rank (if signal is a matrix)?
- Fit a particular pattern?

Setting 2: Multiple Mixtures

Unmix $\{\mathbf{y}_i\}$ into a small number of unknown sources $\{\mathbf{b}_r\}$ with unknown weights $\{a_{ir}\}$:

$$\begin{aligned}\mathbf{y}_1 &= a_{11}\mathbf{b}_1 + a_{12}\mathbf{b}_2 + \cdots + a_{1R}\mathbf{b}_R \\ &\vdots \\ \mathbf{y}_I &= a_{I1}\mathbf{b}_1 + a_{I2}\mathbf{b}_2 + \cdots + a_{IR}\mathbf{b}_R.\end{aligned}$$

Each source s_{ir} is broken up into the product $a_{ir}b_r$.

Ok if you have multiple mixtures of the *same* sources.

If I had all the data in the world…

- Train a deep neural network $f(y;\theta) = (s_1, \ldots, s_R)$ on known pairs of mixtures y and sources $(s_1, \ldots, s_R)$
- Highly specialized to the specific task (e.g. music, images)
- Takes time, energy, and *lots* of data to train the model
- Zoo of architectures (e.g convolutional, LSTMs, transformers)

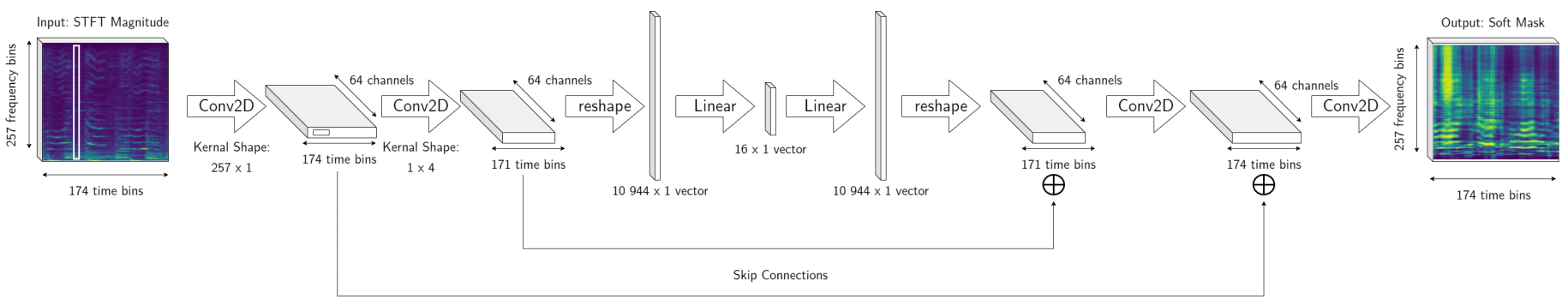


Figure 1: Example "U-net" NN for voice/instrument separation [Richardson 2022].

Solution 1: Sparse Feature Decomposition

Back to Single Signal Separation

Assume the sources $\mathbf{s}_r$ are some multiple of elements $\mathbf{b}_r$ from a "basis" $\mathcal{B}$, and R is small. Then,

$$\mathbf{y} = \sum_{r=1}^{R} a_r \mathbf{b}_r$$

Different conditions on $\mathbf{a}$ mean:

- $a_r \in \mathbb{R} \implies \mathbf{y} \in \text{span}(\mathcal{B})$
- $a_r \geq 0 \implies \mathbf{y} \in \text{cone}(\mathcal{B})$
- $\sum_r a_r = 1 \implies \mathbf{y} \in \text{aff}(\mathcal{B})$
- $a_r \geq 0$ and $\sum_r a_r = 1 \implies \mathbf{y} \in \text{conv}(\mathcal{B})$

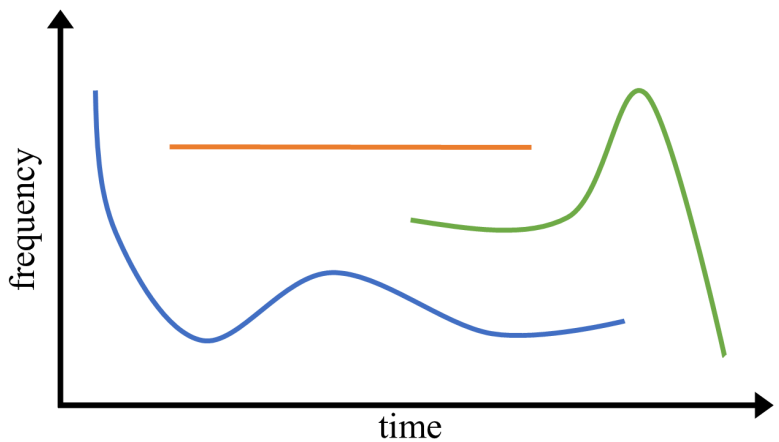
An optimization problem

In practice there might be noise, so we would like to minimize the error between the data $\mathbf{y}$ and our model:

$$\min_{a_r} \left\| \mathbf{y} - \sum_{r=1}^{R} a_r \mathbf{b}_r \right\| .$$

Can add the previously mentioned conditions on a_r as needed.

Sparse Random Mode Decomposition



- Assume sources are intrinsic mode functions $s_r(t) = a_r(t)\sin(\phi_r(t))$
- Varying amplitude $a_r(t)$ and frequency $\omega_r(t) = \phi_r'(t)$
- Look like curves on a time-frequency graph

- Two step process [Richardson *et al.* 2023]
  1. Write your signal as a sparse sum of a wavelet-like features
  2. Cluster nearby features into sources

Part one: Representation

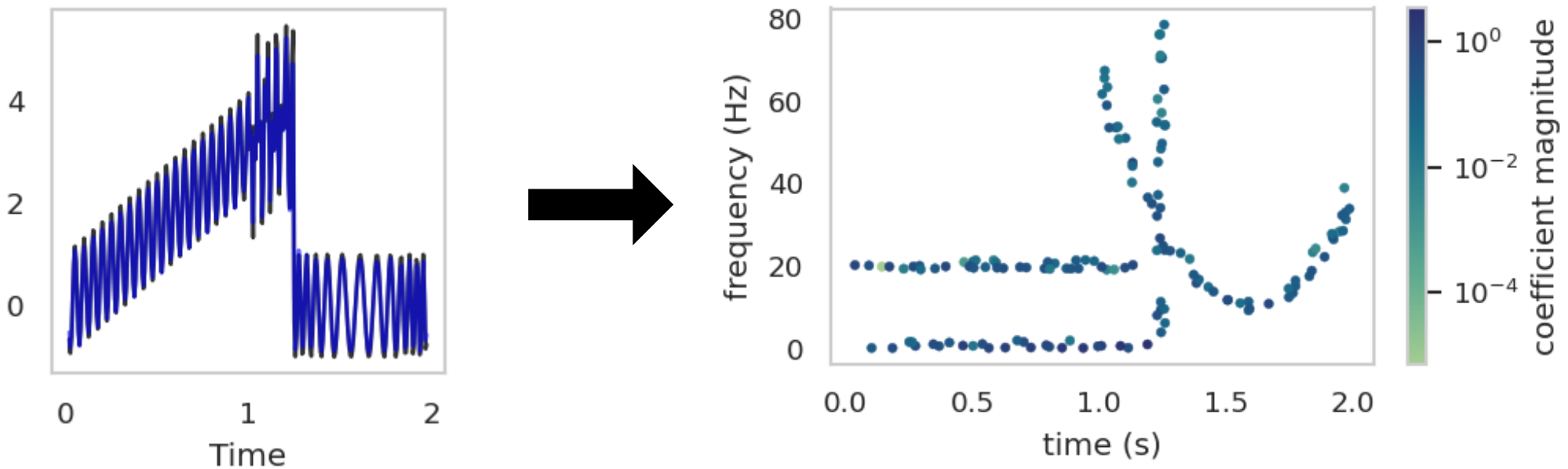


Figure 2: Representing a signal sparsely in time-frequency localized features with SRMD.

Each dot (τ_j, ω_j) corresponds to one feature $b_j(t)$ where

$$b_j(t) = e^{-(t-\tau_j)^2/2w^2} \sin(2\pi\omega_j t + \phi_j).$$

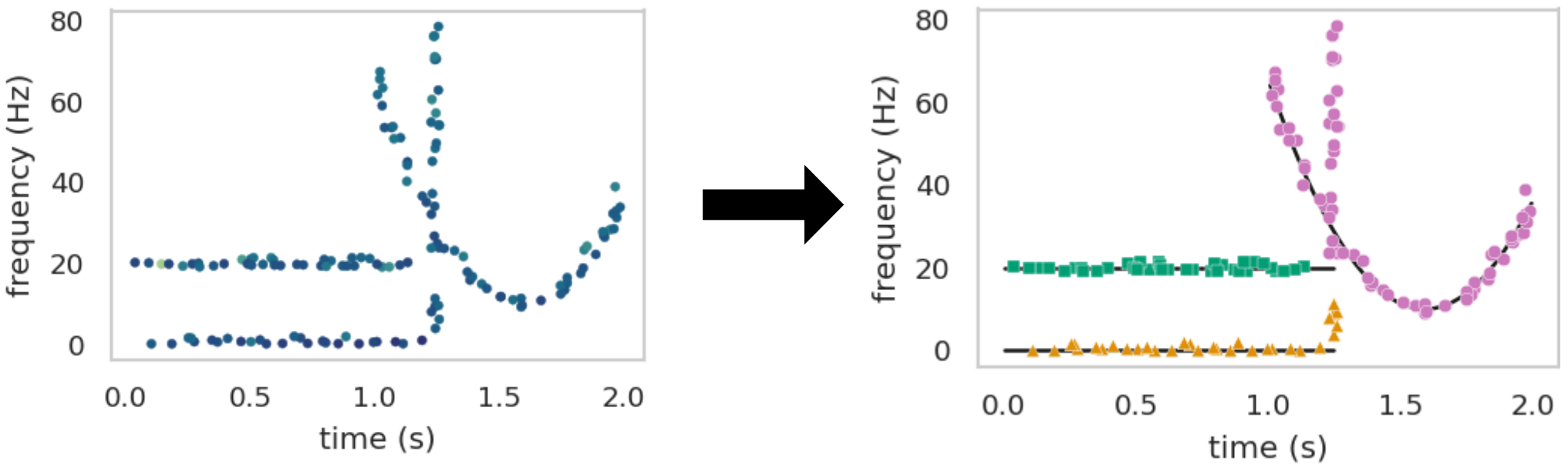
Part one: Representation Details

- Generate many random features $\mathbf{b}_j$
- Express y as the sum of as few "basis" elements as possible
- Ensure the model and data remain close

$$\min_{\mathbf{a}} \|\mathbf{a}\|_1 \quad \text{s.t} \quad \|\mathbf{y} - \mathbf{B}\mathbf{a}\|_2 \leq \epsilon \|\mathbf{y}\|_2.$$

- $\mathbf{B}_{i,r} = b_j(t_i)$ where $b_j(t) = e^{-(t-\tau_j)^2/2w^2} \sin(2\pi\omega_j t + \phi_j)$.
- Time-shifts τ_j, frequencies ω_j, and phases ϕ_j are random
- Solve using SPGL1 (Van Den Berg & Friedlander 2009)

Part two: Clustering



- **D**ensity **B**ased **S**patial **C**lustering of **A**pplications with **N**oise DBSCAN (Ester *et al.* 1996)
- Recursively groups points within a given neighbourhood

Part two: Clustering Details

- Collect non-zero coefficients $\mathcal{C} = \{(\tau_j, \omega_j) \mid a_j \neq 0\}$
- Partition $\mathcal{C}$ into clusters $\{\mathcal{C}_r\}$
  - Use DBSCAN (Ester *et al.* 1996)
  - Groups points (τ_j, ω_j) that are close together
- Reconstruct one source per cluster
  - $\mathbf{s}_r = \sum_{j \in J_r} a_j \mathbf{b}_j$ where $J_r = \{j \mid (\tau_j, \omega_j) \in \mathcal{C}_r\}$

Synthetic Numerical Example

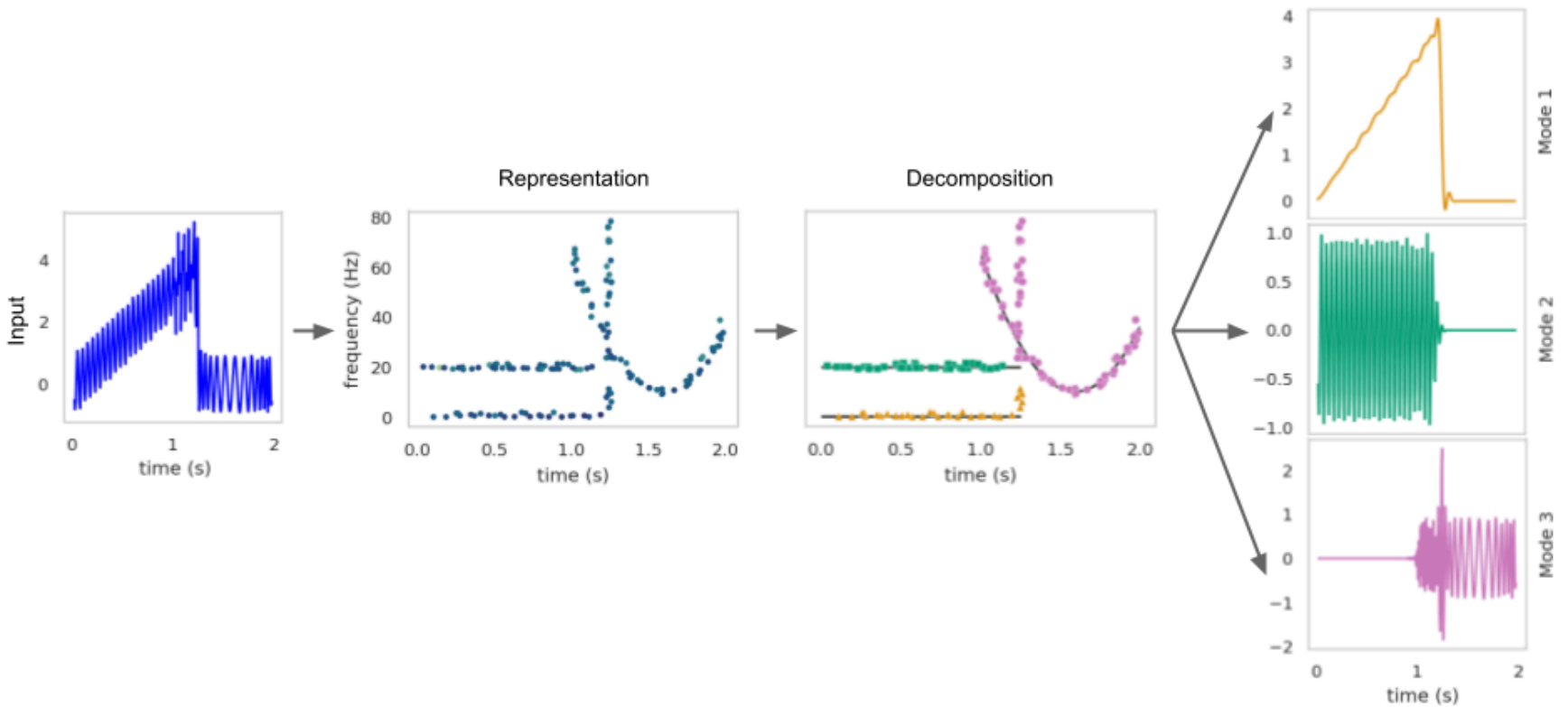
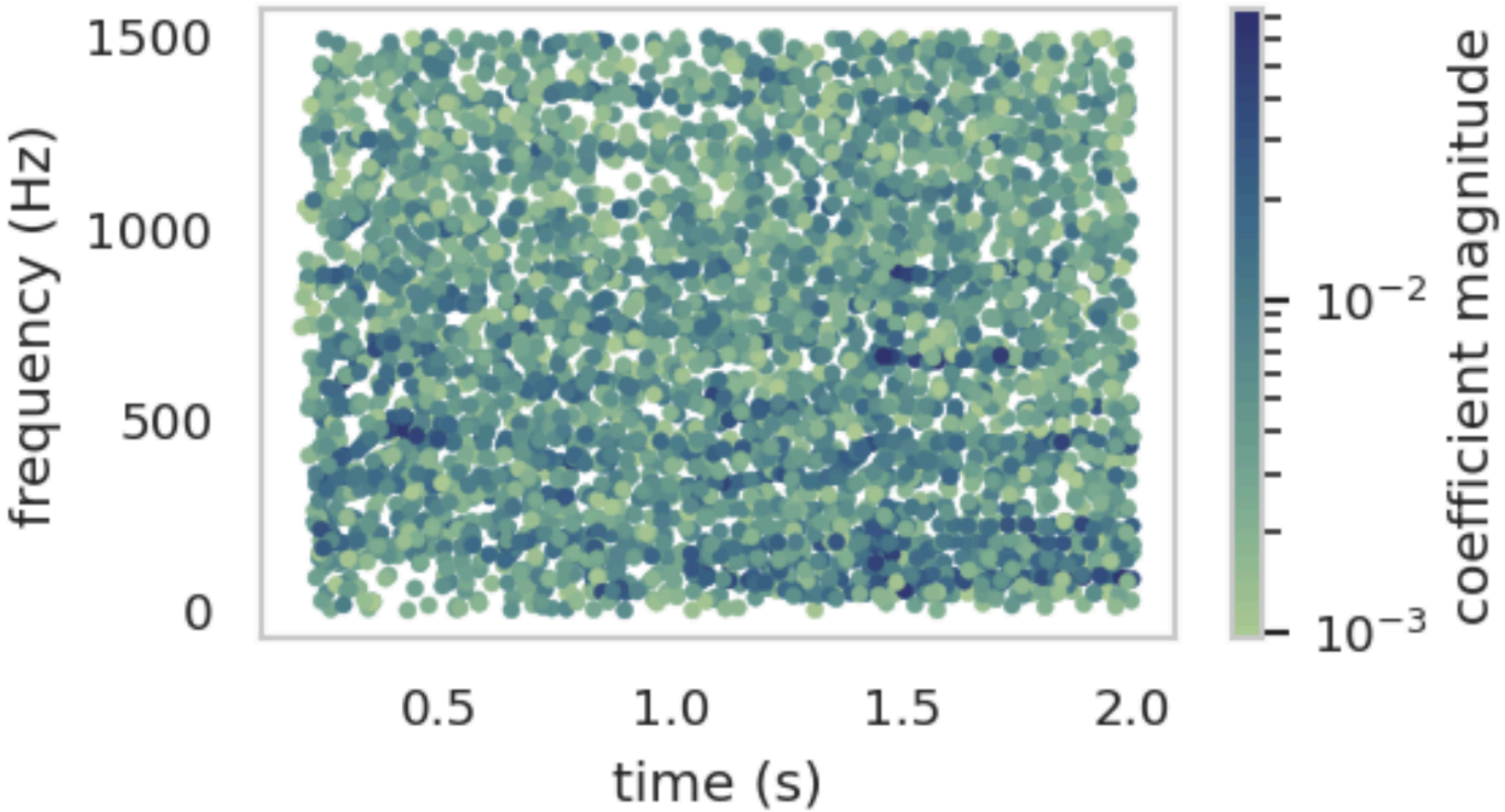


Figure 3: Flowchart of SRMD on the sum of the sources:

$$s_1(t) = \pi t\, 1_{[0,5/4)}(t), \quad s_2(t) = \cos(40\pi t)\, 1_{[0,5/4)}(t),$$
$$s_3(t) = \cos\left(\tfrac{4}{3}\left((2\pi t - 10)^3 - (2\pi - 10)^3\right) + 20\pi(t-1)\right) 1_{(1,2]}(t)$$

Fails for complicated songs…



Solution 2: Constrained Tensor Decomposition

Setting 2: Multiple Mixtures

- If the first setting was a single recording of a band,
- This example would be like setting up 10 microphones in different locations
- **Major Advantage**: No assumptions on the types of sources!
- Only need each mixture be a combination of the *same* sources

Setting 2: Multiple Mixtures

$$\mathbf{y}_1 = a_{11}\mathbf{b}_1 + a_{12}\mathbf{b}_2 + \cdots + a_{1R}\mathbf{b}_R$$
$$\vdots$$
$$\mathbf{y}_I = a_{I1}\mathbf{b}_1 + a_{I2}\mathbf{b}_2 + \cdots + a_{IR}\mathbf{b}_R$$

Package data into a tensor $\mathbf{Y}$ by sampling the mixtures $\{\mathbf{y}_i\}$ or stacking the observations

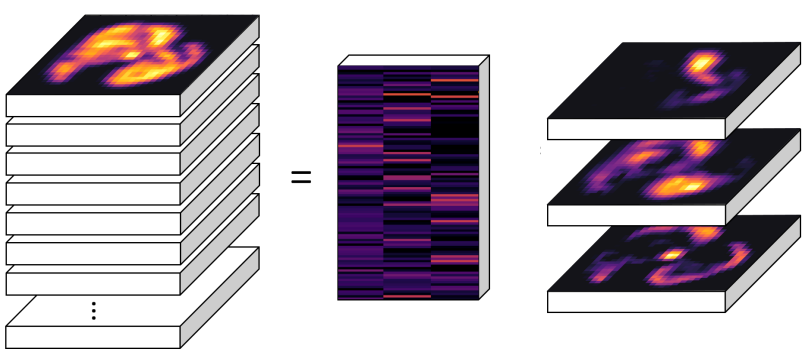


Figure 4: Example data tensor $\mathbf{Y}$ and it's decomposition into $\mathbf{A}$ and $\mathbf{B}$.

Model: Tucker-1 Tensor Factorization

- Factorize $\mathbf{Y}$ into a mixing matrix $\mathbf{A}$ times a source tensor B using the Tucker-1 model (Kolda and Bader 2009)
- $\mathbf{Y} = \mathbf{B} \times_1 \mathbf{A}$ with the entry-wise equation
- $\mathbf{Y}[i, j_1, \ldots, j_N] = \sum_{r=1}^{R} \mathbf{A}[i, r] \cdot \mathbf{B}[r, j_1, \ldots, j_N]$

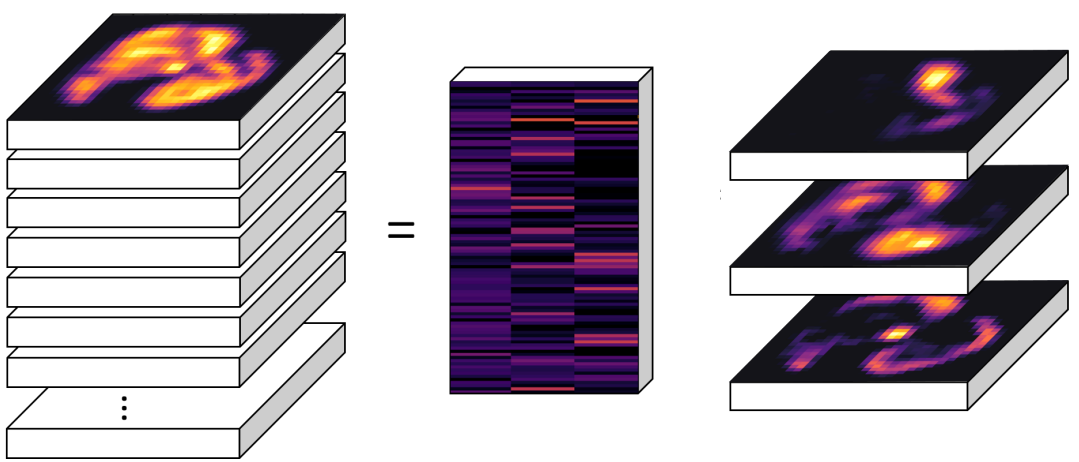


Figure 5: Example data tensor $\mathbf{Y}$ and it's decomposition into $\mathbf{A}$ and $\mathbf{B}$.

Application: Sediment Analysis

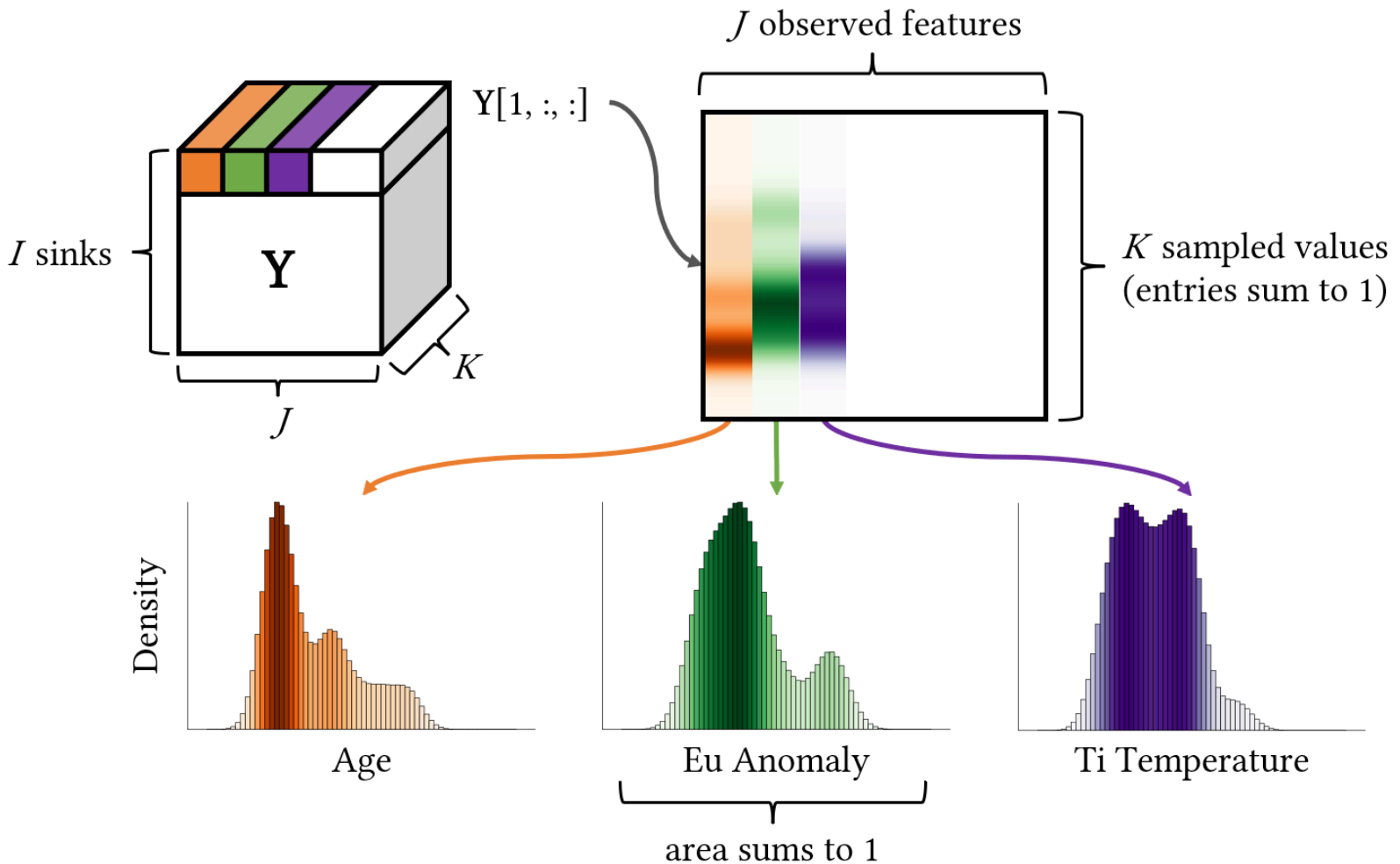


Figure 6: Input data tensor $\mathbf{Y}$. Each depth fibre $\mathbf{Y}[i, j, :]$ is a discretized probability density for a different geological feature. We decompose with [Graham *et al.* 2025].

Application: Sediment Analysis

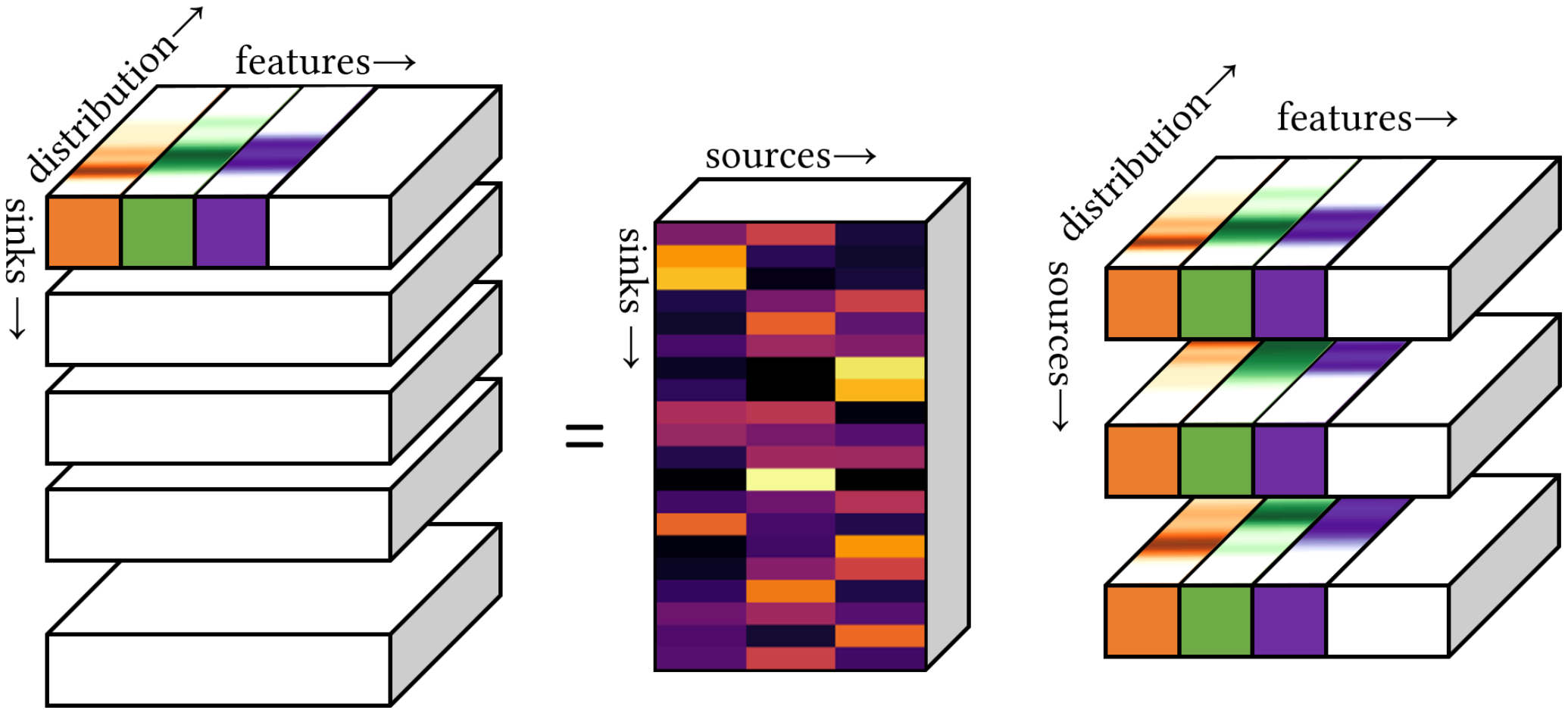


Figure 7: Decomposition of the input tensor $\mathbf{Y}$ into mixing coefficients $\mathbf{A}$ and source distributions $\mathbf{B}$.

Application: Sediment Analysis

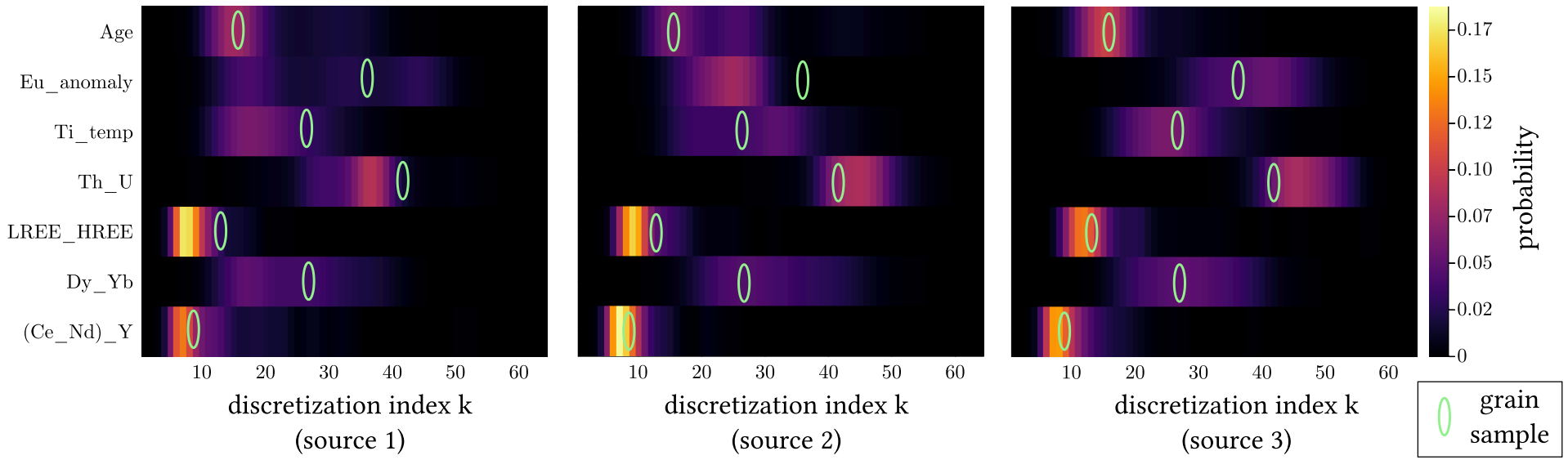


Figure 8: Three learned density sources. Example grain sample highlighted in green ovals.

- Estimate probability grain $\mathbf{g}$ came from source r:
  $p_r = \mathbb{P}(\mathbf{g} \in \mathcal{V}_{\mathbf{g}} \mid \mathbf{g} \sim \mathbf{B}_r) \approx \prod_{j=1}^{J} \mathbf{B}[r, j, \hat{k}_j]$
- $\mathcal{V}_{\mathbf{g}} = [x_{1\hat{k}_1}, x_{1(\hat{k}_1+1)}] \times \cdots \times [x_{J\hat{k}_J}, x_{J(\hat{k}_J+1)}]$ is the box that contains the measured grain
- Label based on the most likely probability $\hat{r} = \operatorname{argmax}_r p_r$
- Assign confidence score $= \log_{10}\left(p_{(1)}/p_{(2)}\right)$

Application: Spatial Transcriptomics

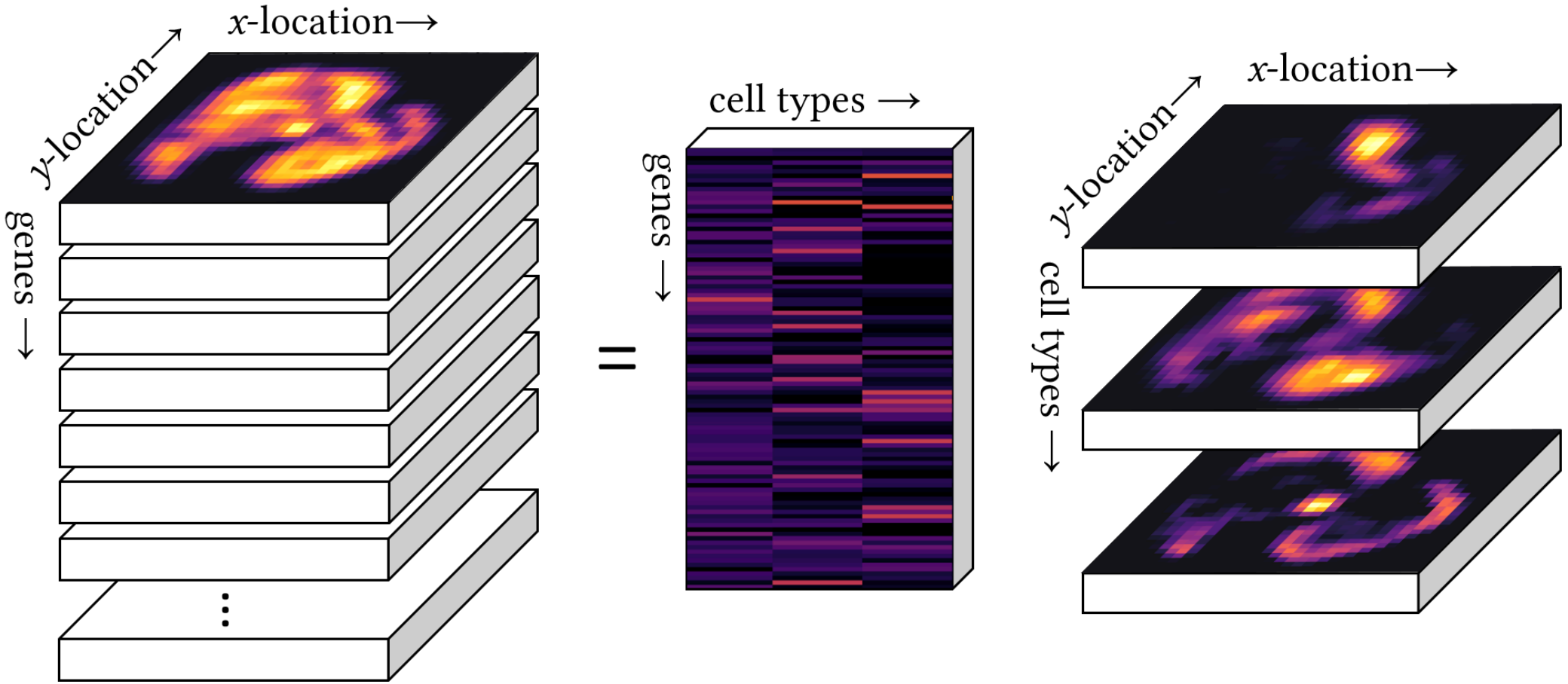


Figure 9: Spatial transcriptomics factorization model. Spatial distribution of many genes can be decomposed into few cell types. We uncover the gene expression and spatial distribution of these cell types, and can label distinct regions accordingly.

Application: Spatial Transcriptomics

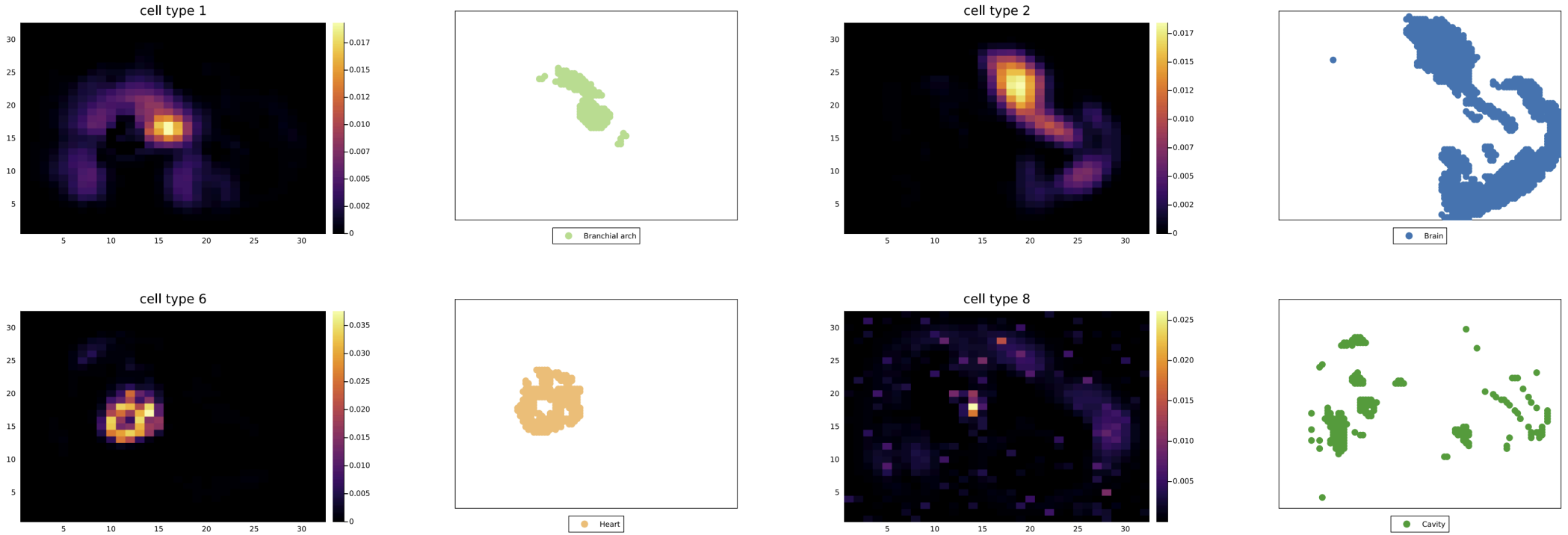


Figure 10: Learn cell types and their spatial presence (heatmaps) matched with known cell types (Branchial Arch, Brain, Heart, and Cavity).

Application: Musical Instrument Separation

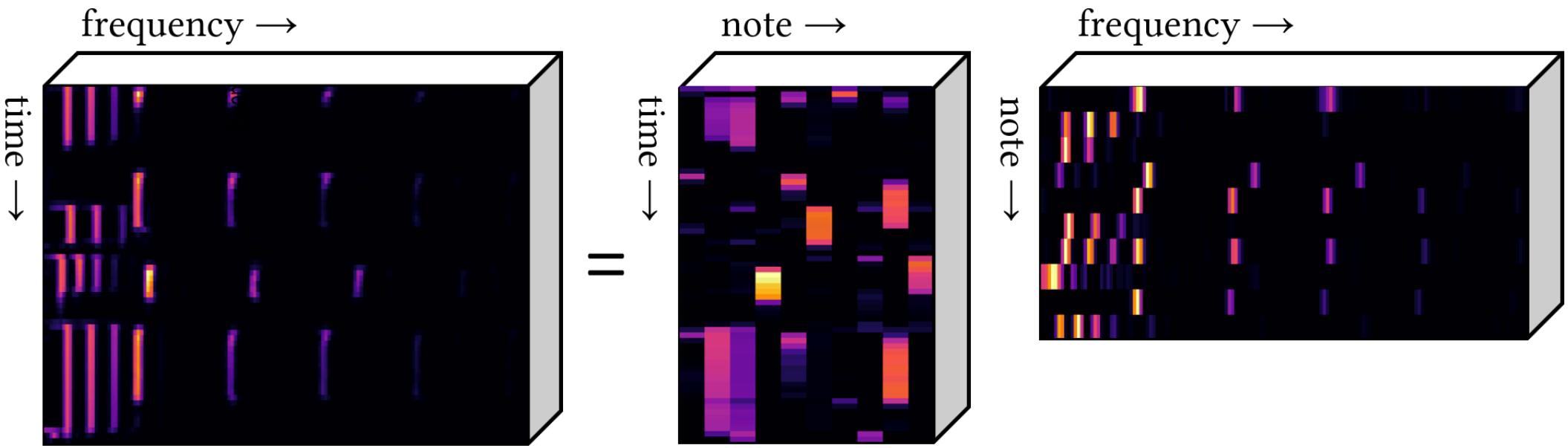


Figure 11: Audio source separation model. The short-time Fourier transform of a mixture can be separated into harmonically distinct notes. These can be grouped by their spectral similarity to recover instrument sources.

Application: Musical Instrument Separation

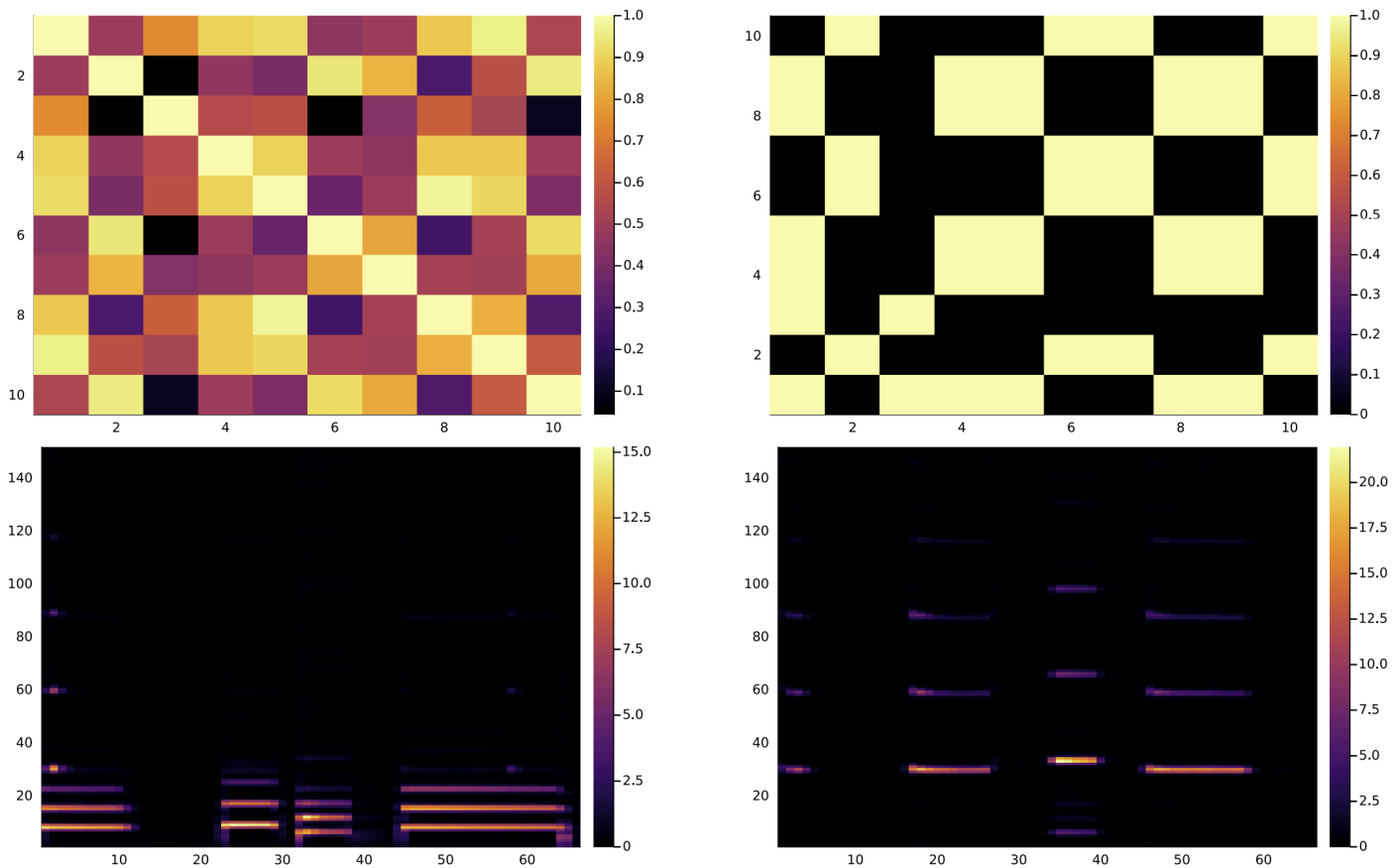


Figure 12: (Top) Cross-correlation matrix between notes' frequencies before and after threshold. (Bottom) Separated guitar and flute spectrograms.

Computing Tensor Factorizations

Constrained Optimization

Minimize the error between the model $B \times_1 A$ and the data Y:

$$\min_{\mathbf{A},\mathbf{B}} \ell(\mathbf{A}, \mathbf{B}) \quad \text{s.t} \quad \mathbf{A} \in \mathcal{C}_A, \ \mathbf{B} \in \mathcal{C}_B.$$

Implementation in Julia [Richardson *et al.* 2025]:

```julia
using BlockTensorFactorization
Y = load_data()

options = (rank=3, model=Tucker1, objective=L2(), constraints=nonnegative!)
decomposition, stats, kwargs = factorize(Y; options...)

(B, A) = factors(decomposition)
```

Algorithm:

Block Projected Gradient Descent

Cyclically update factors with descent updates (Xu *et al.* 2023)

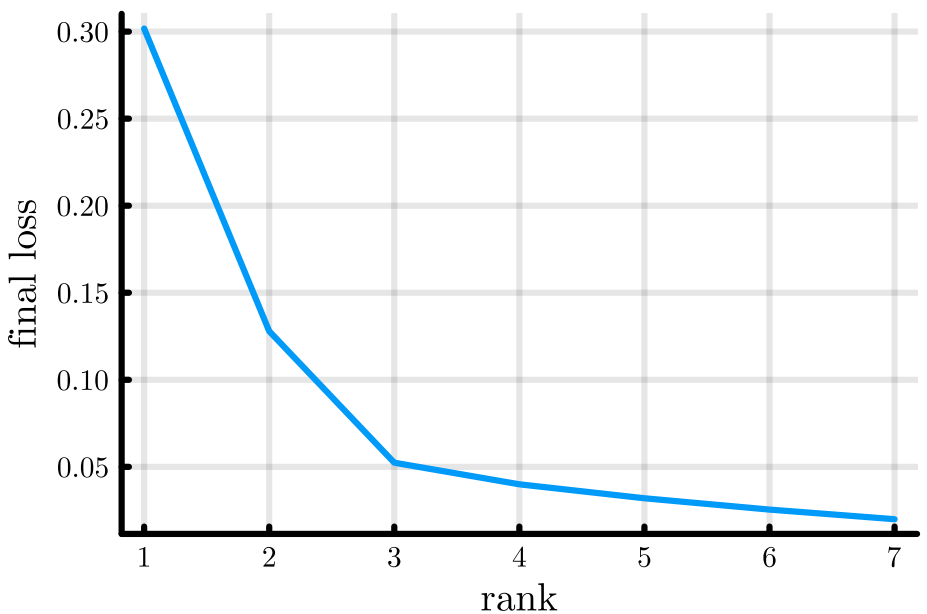
$$\mathbf{A}^{t+1} = P_{\mathcal{C}_A}\left(\mathbf{A}^t - \alpha \nabla_A \ell(\mathbf{A}^t, \mathbf{B}^t)\right).$$

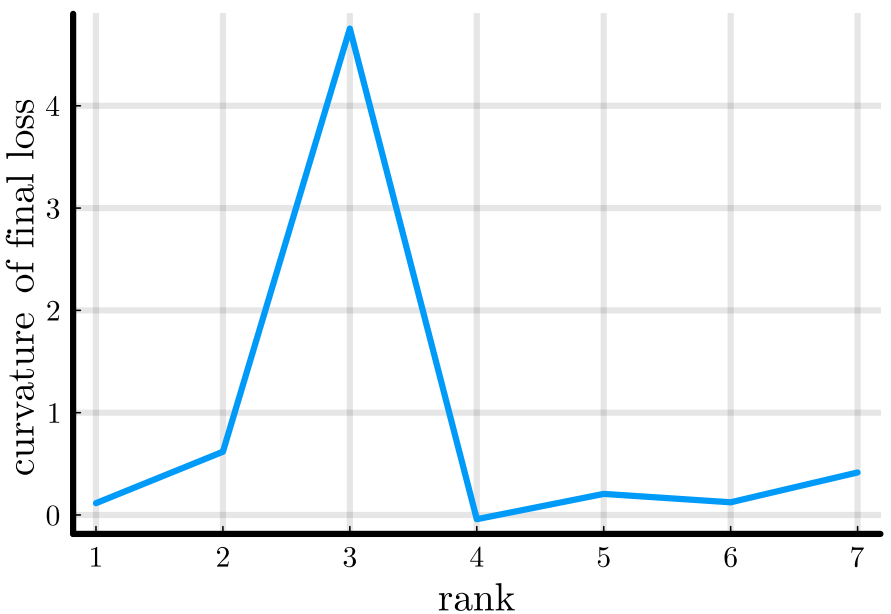
Converges to a *block-wise* minimum and stationary point:

$$\ell(\mathbf{A}^*, \mathbf{B}^*) \leq \min_{\mathbf{A}\in\mathcal{C}_A, \mathbf{B}\in\mathcal{C}_B} \left\{\ell(\mathbf{A}^*, \mathbf{B}), \ell(\mathbf{A}, \mathbf{B}^*)\right\}.$$

Estimating Rank

- Usually R needs to be known in advance, but can be estimated
- Let $f(r) = \|\mathbf{X}_r^* - \mathbf{Y}\|_F / \|\mathbf{Y}\|_F$ be final relative error with a rank r factorization X_r^*
- Pick the r at the maximum curvature (Satopaa *et al.* 2011)





Multi-Scaled Decomposition

Optimize over multiple scales for faster convergence.

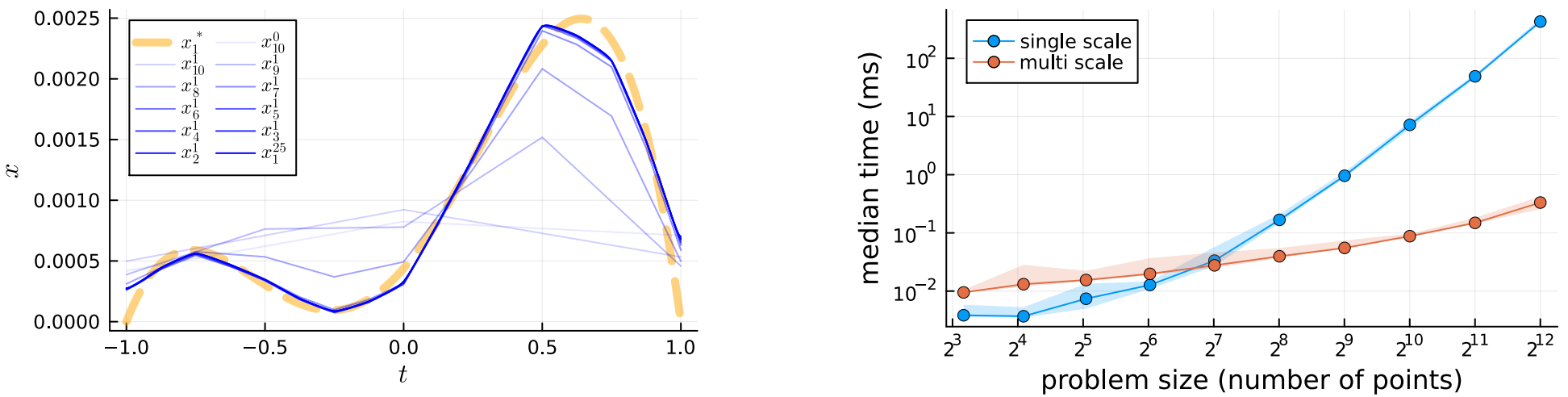


Figure 13: (Left) Optimize over cheaper, coarse discretization with fewer points and refine. (Right) Performance scales better with problem size [Richardson *et al.* 2025].

Conclusion

- Real world data is often a mixture of individual sources
- Want to separate these sources for analysis, denoising, etc.
- Looked at two techniques when you have limited data
  1. Single Source Separation via Sparse Feature Decomposition
  2. Multiple Mixture Separation via Tensor Factorization

Contact & References

Website: https://njericha.github.io
Email: njericha@math.ubc.ca

[1] N. J. E. Richardson, "A Sparse Random Feature Model for Signal Decomposition," Master's thesis, University of Waterloo, 2022.

[2] N. Richardson, H. Schaeffer, and G. Tran, "SRMD: Sparse Random Mode Decomposition," *Communications on Applied Mathematics and Computation*, Jun. 2023.

[3] N. Graham, N. Richardson, M. P. Friedlander, and J. Saylor, "Tracing Sedimentary Origins in Multivariate Geochronology via Constrained Tensor Factorization," *Mathematical Geosciences*, Feb. 2025.

[4] N. Richardson, N. Marusenko, and M. P. Friedlander, "BlockTensorFactorization.jl," *GitHub*. https://github.com/MPF-Optimization-Laboratory/BlockTensorFactorization.jl; GitHub, 2025.

[5] N. J. E. Richardson, N. Marusenko, and M. P. Friedlander, "Multiple Scale Methods For Optimization Of Discretized Continuous Functions." arXiv, Dec. 2025.

Extra slides

Under the hood: SPGL1

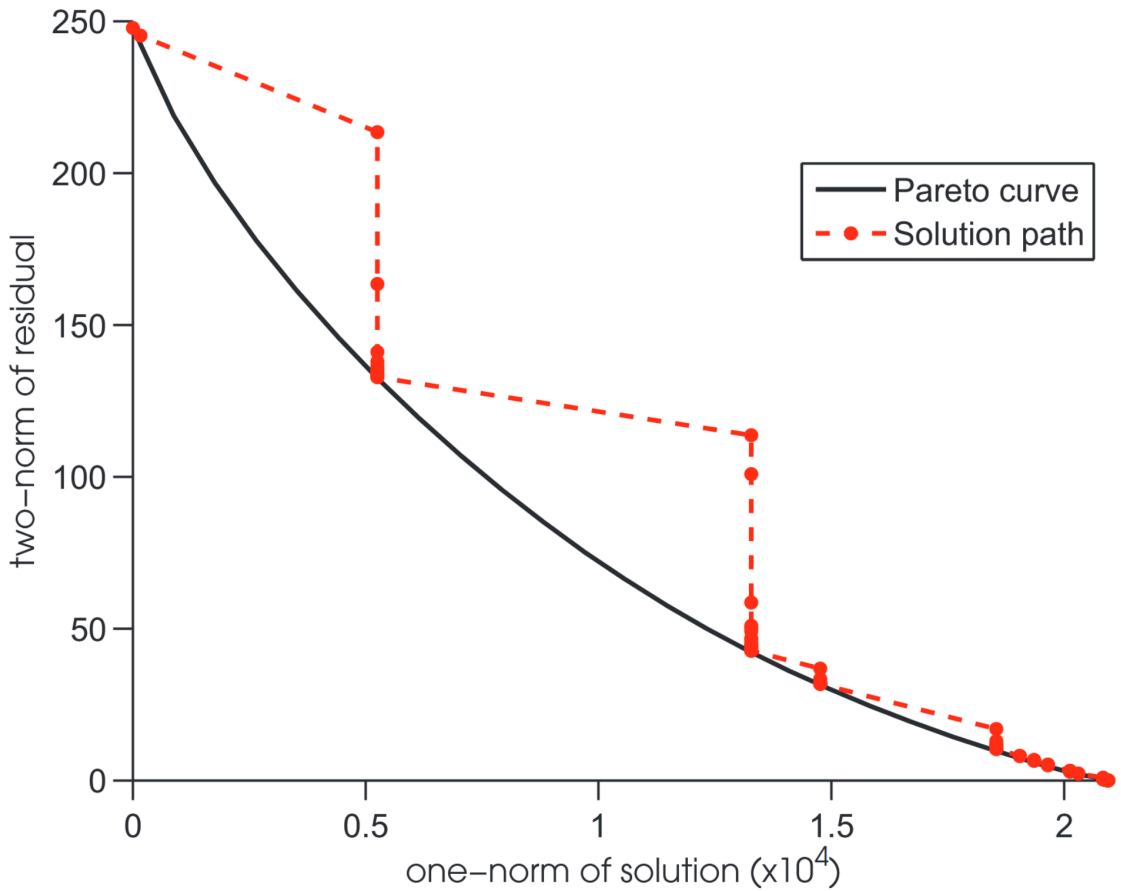


Figure 14: **S**pectral **P**rojected-**G**radient with **L1**-norm solves LASSO problems for growing sparsity tolerances until the residual is small enough (Van Den Berg & Friedlander 2009).

$$\min_{\mathbf{a}} \|\mathbf{y} - \mathbf{B}\mathbf{a}\|_2 \quad \text{s.t} \quad \|\mathbf{a}\|_1 \leq \tau$$

Compare to other methods

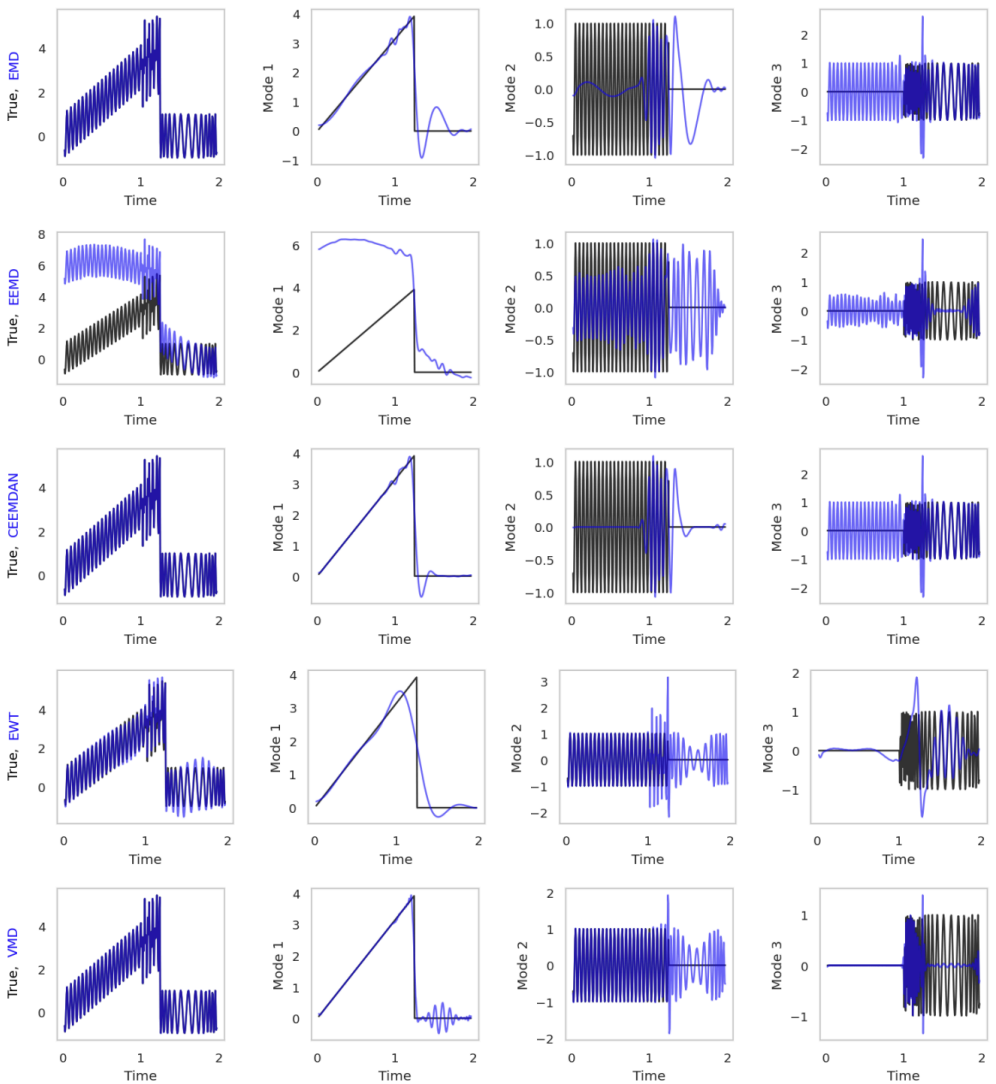


Figure 15: Alternate methods are less robust than SRMD.

Block-Lipschitz Constant

$$a_{n,r}^{t+1} \leftarrow P_{\mathcal{C}_{n,r}}\left(a_{n,r}^{t} - \frac{1}{L_{n,r}^{t}} \nabla f_{n,r}^{t}(a_{n,r}^{t})\right),$$

where
$f_{n,r}^{t}(a) = \frac{1}{2}\left\|[B; A_1^{t+1}, \ldots, A_{n-1}^{t+1}, A_{n,r}^{t}(a), A_{n+1}^{t}, \ldots, A_N^{t}] - Y\right\|_F^2$
and

$$A_{n,r}^{t}(a) = \begin{bmatrix} \uparrow & & \uparrow & \uparrow & \uparrow & & \uparrow \\ a_{n,1}^{t+1} & \cdots & a_{n,r-1}^{t+1} & a & a_{n,r+1}^{t} & \cdots & a_{n,R_n}^{t} \\ \downarrow & & \downarrow & \downarrow & \downarrow & & \downarrow \end{bmatrix}.$$

Block-Lipschitz Constant

$$A_n^{t+1} \leftarrow P_{\mathcal{C}_n}\left(A_n^t - \nabla f_n^t(A_n^t)(\hat{L}_n^t)^{-1}\right),$$

where
$f_n^t(a) = \frac{1}{2}\left\|[B; A_1^{t+1}, \ldots, A_{n-1}^{t+1}, A_n^t, A_{n+1}^t, \ldots, A_N^t] - Y\right\|_F^2$
and

$$\hat{A}_n^t = \begin{bmatrix} \uparrow & & \uparrow & \uparrow & \uparrow & & \uparrow \\ a_{n,1}^t & \cdots & a_{n,r-1}^t & a_{n,r}^t & a_{n,r+1}^t & \cdots & a_{n,R_n}^t \\ \downarrow & & \downarrow & \downarrow & \downarrow & & \downarrow \end{bmatrix}.$$

Momentum

Before a gradient step, we move A further in the direction of travel

$$\begin{aligned}\hat{A}_n^t \leftarrow & A_n^t + \omega_n^t (A_n^t - A_n^{t-1}) \\ & = A_n^t (\mathrm{id}_{R_n} + \omega_n^t) - A_n^{t-1} \omega_n^t\end{aligned}$$

where the amount of momentum is determined by

$$\omega_n^t \leftarrow \min \left(\hat{\omega}^t, \delta \sqrt{\hat{L}_n^{t-1} (\hat{L}_n^t)^{-1}} \right).$$

Examples of Constraints

```
struct GenericConstraint <: AbstractConstraint
    apply::Function
    # input a AbstractArray -> mutate it so that `check` would return true
    check::Function
end

function (C::GenericConstraint)(D::AbstractArray)
    (C.apply)(D)
end

check(C::GenericConstraint, A::AbstractArray) = (C.check)(A)
```

```
l2scale_1slices! = ScaledNormalization(l2norm;
                   whats_normalized=(x -> eachslice(x; dims=1)))

l1normalize_rows! = ProjectedNormalization(l1norm, l1project!;
                    whats_normalized=eachrow)

nonnegative! = Entrywise(ReLU, isnonnegative)

IntervalConstraint(a, b) = Entrywise(x -> clamp(x, a, b), x -> a <= x <= b)
```

Randomizing the order of updates

```
BlockedUpdate(
    MomentumUpdate(0, lipschitz, combine)
    GradientDescent(0, gradient, LipschitzStep)
    Projection(0, Entrywise(ReLU, isnonnegative))
    MomentumUpdate(1, lipschitz, combine)
    GradientDescent(1, gradient, LipschitzStep)
    Projection(1, Entrywise(ReLU, isnonnegative))
)
```

Include the boolean options;

- `group_by_factor`: Groups updates on the same factor together
- `random_order`: Updates in a new random order each iteration
- `recursive_random_order`: Inner grouped updates performed in a random order (recursively)

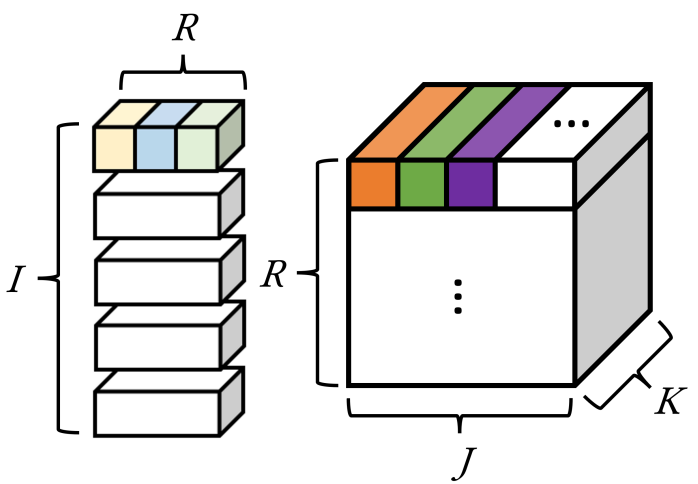
Rescaling trick

If you have simplex constraints

- $A \in \Delta_A = \left\{ A \in \mathbb{R}_+^{I \times R} \middle| \sum_{r=1}^{R} A[i,r] = 1, \forall i \right\}$
- $B \in \Delta_B = \left\{ A \in \mathbb{R}_+^{R \times J \times K} \middle| \sum_{k=1}^{K} B[r,j,k] = 1, \forall r,j \right\}$

Updates look like:

- $\mathbf{A}^{t+1} = P_{\Delta_A}(\mathbf{A}^t - \frac{1}{L_\mathbf{A}} \nabla_\mathbf{A} \ell(\mathbf{A}^t, \mathbf{B}^t))$
- $\mathbf{B}^t = P_{\Delta_A}(\mathbf{B}^t - \frac{1}{L_\mathbf{B}} \nabla_\mathbf{B} \ell(\mathbf{A}^{t+1}, \mathbf{B}^t))$



Rescaling trick

- Relax simplex constraints to $\mathbf{A} \geq 0, \mathbf{B} \geq 0$
- and $\frac{1}{J} \sum_{jk} B_{rjk} = 1$ for all r (vs $\sum_{k} B_{rjk} = 1$ for all r, j)

Updates now look like:

- $\mathbf{A}^{t+1/2} = (\mathbf{A}^t - \frac{1}{L_{\mathbf{A}}} \nabla_{\mathbf{A}} \ell(\mathbf{A}^t, \mathbf{B}^t))_+$
- $\mathbf{B}^{t+1/2} = (\mathbf{B}^t - \frac{1}{L_{\mathbf{B}}} \nabla_{\mathbf{B}} \ell(\mathbf{A}^{t+1/2}, \mathbf{B}^t))_+$
- $\mathbf{B}^{t+1} = \mathbf{C}^{-1} \mathbf{B}^{t+1/2}$ and $\mathbf{A}^{t+1} = \mathbf{A}^{t+1/2} \mathbf{C}$
- where $C_{rr} = \frac{1}{J} \sum_{jk} B_{rjk}$

Rescaling vs simplex projection

- Compare stationary condition: $\mathrm{dist}\left(0, \partial(\ell + \delta_{\geq 0})(\mathbf{A}, \mathbf{B})\right)$ at every iteration for different constraint methods