Computational Techniques for Analysis of Mixed Signals

23 July 2026

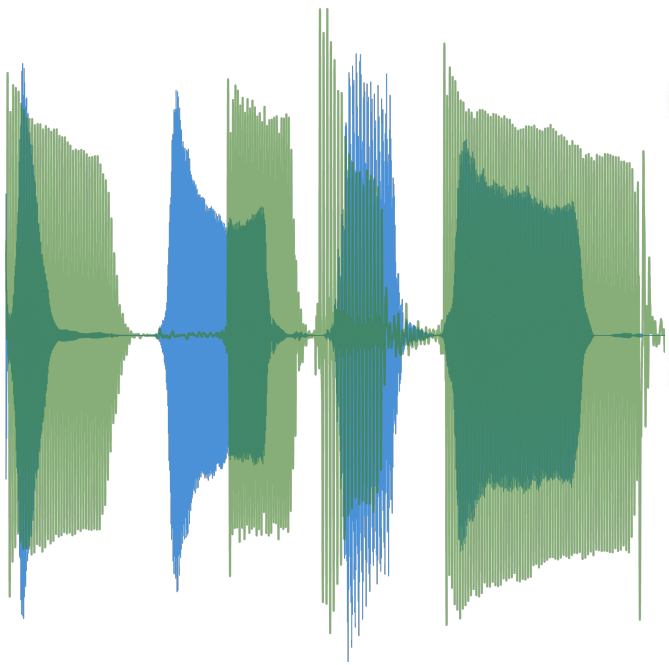
Nicholas Joseph Emile Richardson

A defense for the degree of Doctor of Philosophy in Mathematics

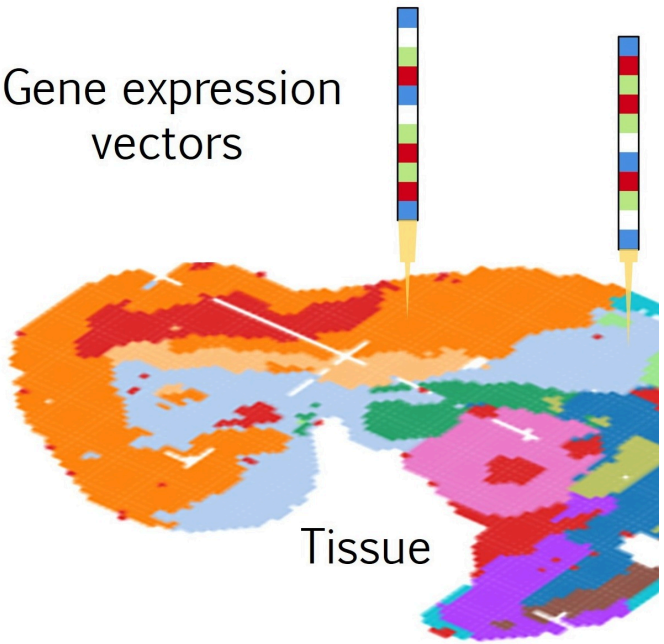
THE UNIVERSITY
OF BRITISH COLUMBIA

Introduction

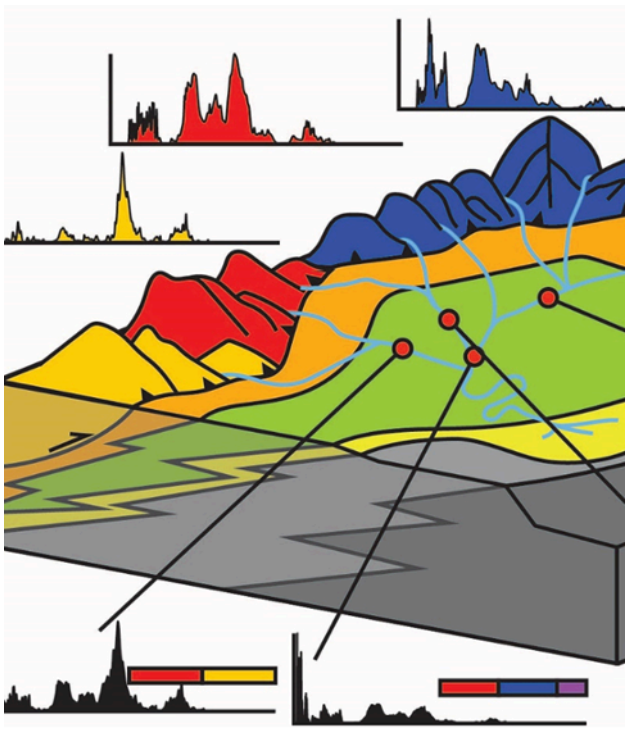
Mixed signals are everywhere



Songs are mixtures of different instruments



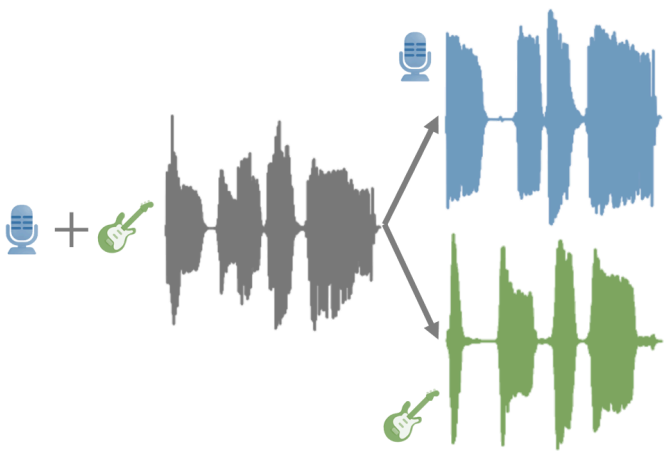
Gene counts are influenced by cell types



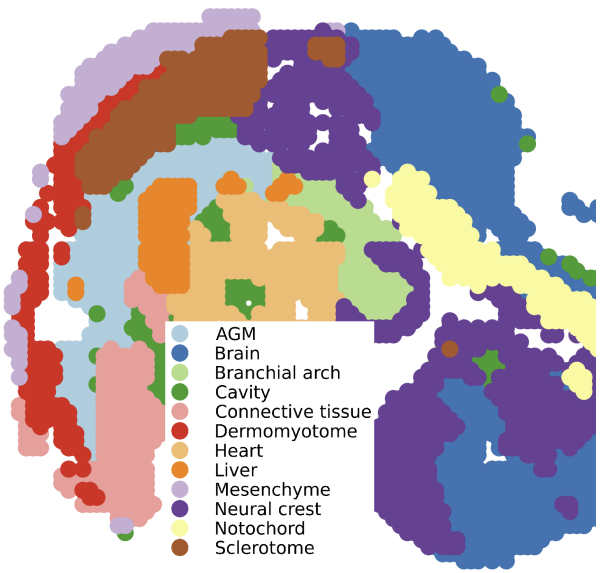
Rocks are mixtures of mineral sources

Determine the underlying sources in data

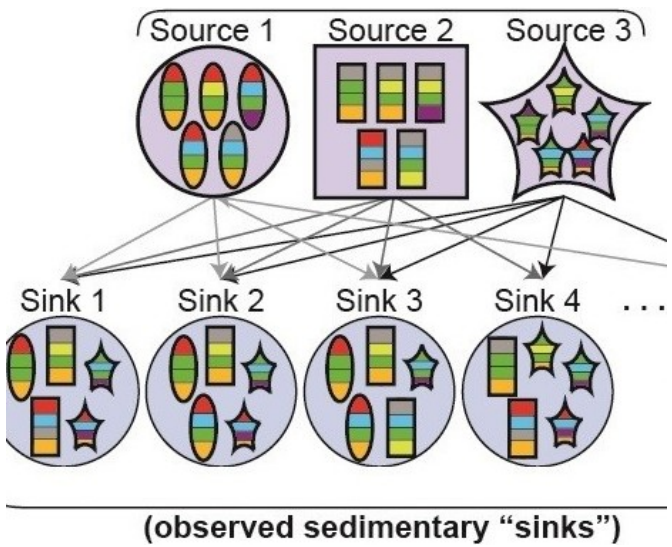
Why? For identifying sources, pre-processing, removing noise, etc.



Identify sound sources within a song



Segment embryos by cell types



Find most prominent minerals across rocks

Complicated data
= Mixtures of simple data

Understand the pieces

$\Longrightarrow$ Understand the whole

State of Data Processing and Analysis

- ✓ State-of-the-art methods use supervised machine learning methods
- ✓ Have amazing performance when trained on lots of data

State of Data Processing and Analysis

- ✓ State-of-the-art methods use supervised machine learning methods
- ✓ Have amazing performance when trained on lots of data
- ✗ Lack of tools to solve problems where collecting data is costly
- ✗ Modern machine learning becoming expensive as deep neural networks use increasingly more resources (energy, water, etc.)

State of Data Processing and Analysis

- ✓ State-of-the-art methods use supervised machine learning methods
- ✓ Have amazing performance when trained on lots of data
- ✗ Lack of tools to solve problems where collecting data is costly
- ✗ Modern machine learning becoming expensive as deep neural networks use increasingly more resources (energy, water, etc.)
- **Q** Can we use unsupervised machine learning techniques to analyze data without labelled examples?
- **Q** How can we analyze complicated data efficiently?

Big Picture Goal:

Develop, improve, and apply tensor factorization tools to un-mix data

Overview of Thesis Contributions

- **Model** decomposition problems as tensor factorizations in new ways
- **Re-express** first-order algorithms using tensors natively
- **Develop** fast and flexible factorization packages in the Julia language
- **Innovate** factorization algorithms to improve convergence
- **Generalize** techniques beyond signal decomposition
- **Apply** these tools to applications in Geology, Biology, and Music

Modelling Signal Decomposition Problems

Single Signal Separation

Unmix a signal $y \in \mathbb{R}^n$ into R sources $\{s_1, ..., s_R\}$:

$$y = s_1 + s_2 + \cdots + s_R.$$

Classical Methods (IMF, GMMs, etc.)	**Supervised Deep Learning (CNNs, RNNs, etc.)**
✓ Little or no training	✗ Need lots of training data
✗ Need to know your data (sparse, low rank, etc.)	✓ Few assumptions
✗ Specialized to a specific task	✗ Specialized to a specific task

Single Signal Separation

Unmix a signal $y \in \mathbb{R}^n$ into R sources $\{s_1, ..., s_R\}$:

$$y = s_1 + s_2 + \cdots + s_R.$$

Classical Methods (IMF, GMMs, etc.)	**Supervised Deep Learning (CNNs, RNNs, etc.)**
✓ Little or no training	✗ Need lots of training data
✗ Need to know your data (sparse, low rank, etc.)	✓ Few assumptions
✗ Specialized to a specific task	✗ Specialized to a specific task

Q Can we have it all?

Multiple Mixtures Model

Assume you have multiple mixtures $\{y_m\}$ of the same sources $\{b_r\}$,

$$\begin{cases} y_1 = a_{11}b_1 + a_{12}b_2 + \cdots + a_{1R}b_R \\ \quad\vdots \\ y_M = a_{M1}b_1 + a_{M2}b_2 + \cdots + a_{MR}b_R. \end{cases}$$

As a matrix equation,

$$\begin{bmatrix} \leftarrow y_1^\top \rightarrow \\ \vdots \\ \leftarrow y_M^\top \rightarrow \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1R} \\ \vdots & \ddots & \vdots \\ a_{M1} & \cdots & a_{MR} \end{bmatrix} \begin{bmatrix} \leftarrow b_1^\top \rightarrow \\ \vdots \\ \leftarrow b_R^\top \rightarrow \end{bmatrix}$$

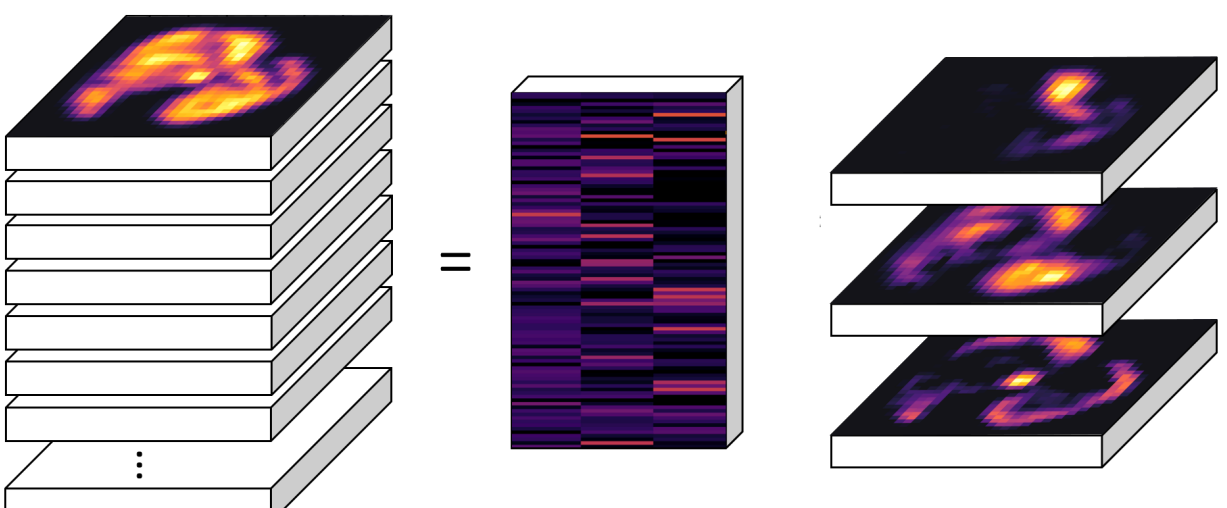
$$Y = AB.$$

What if mixtures y_m are high dimensional?

The Tucker-1 factorization generalizes the matrix equation to tensors!

$$Y = B \times_1 A \quad \text{(factors flipped by convention)}$$

$$Y[m, j_1, ..., j_N] = \sum_{r=1}^{R} A[m, r] \cdot B[r, j_1, ..., j_N]$$



Sub-goal:

Given a tensor Y, recover a matrix A and tensor B so that $Y = AB \quad (= B \times_1 A)$.

Computing Tensor Factorizations

Least-squares and Alternating Minimization

Reframe the problem as minimizing the error between Y and AB,

$$\min_{A,B} \|AB - Y\|^2 \quad \text{s.t.} \quad A \in \mathcal{C}_A,\ B \in \mathcal{C}_B.$$

Accounts for possible noise or error in the data or model: $Y = AB + E$

Least-squares and Alternating Minimization

Reframe the problem as minimizing the error between Y and AB,

$$\min_{A,B} \|AB - Y\|^2 \quad \text{s.t.} \quad A \in \mathcal{C}_A,\ B \in \mathcal{C}_B.$$

Accounts for possible noise or error in the data or model: $Y = AB + E$

Solve by alternatingly finding the best A and B, holding the other fixed,

$$A^{k+1} \longleftarrow \arg\min_{A} \left\|AB^k - Y\right\|^2 \quad \text{s.t.} \quad A \in \mathcal{C}_A$$

$$B^{k+1} \longleftarrow \arg\min_{B} \left\|A^{k+1}B - Y\right\|^2 \quad \text{s.t.} \quad B \in \mathcal{C}_B.$$

Least-squares and Alternating Minimization

Reframe the problem as minimizing the error between Y and AB,

$$\min_{A,B} \|AB - Y\|^2 \quad \text{s.t.} \quad A \in \mathcal{C}_A,\ B \in \mathcal{C}_B.$$

Accounts for possible noise or error in the data or model: $Y = AB + E$

Solve by alternatingly finding the best A and B, holding the other fixed,

$$A^{k+1} \longleftarrow \arg\min_{A} \left\|AB^k - Y\right\|^2 \quad \text{s.t.} \quad A \in \mathcal{C}_A$$

$$B^{k+1} \longleftarrow \arg\min_{B} \left\|A^{k+1}B - Y\right\|^2 \quad \text{s.t.} \quad B \in \mathcal{C}_B.$$

These sub-problems have no closed form solution in general $\Longrightarrow$ slow!

What sub problem to use?

Rather than complete minimization,

$$\underset{A \in \mathcal{C}_A}{\arg\min}\ f(A) := \left\| AB^k - Y \right\|^2$$

minimize a quadratic upper bound at current iterate A^k,

$$\underset{A \in \mathcal{C}_A}{\arg\min}\ f\left(A^k\right) + \left\langle \nabla f\left(A^k\right), A - A^k \right\rangle + \frac{L}{2} \left\| A - A^k \right\|^2$$

where L is the Lipschitz constant: $\|\nabla f(X) - \nabla f(Y)\| \leq L \|X - Y\|$.

What sub problem to use?

Rather than complete minimization,

$$\underset{A \in \mathcal{C}_A}{\arg\min}\ f(A) := \left\| AB^k - Y \right\|^2$$

minimize a quadratic upper bound at current iterate A^k,

$$\underset{A \in \mathcal{C}_A}{\arg\min}\ f\left(A^k\right) + \left\langle \nabla f\left(A^k\right), A - A^k \right\rangle + \frac{L}{2} \left\| A - A^k \right\|^2$$

where L is the Lipschitz constant: $\|\nabla f(X) - \nabla f(Y)\| \leq L \|X - Y\|$.

This has the closed form solution: $P_A\left(A^k - \frac{1}{L} \nabla f\left(A^k\right)\right)$,
where P_A is the projection operator onto $\mathcal{C}_A$.

Least-squares and Alternating Minimization

Reframe the problem as minimizing the error between Y and AB,

$$\min_{A,B} \|AB - Y\|^2 \quad \text{s.t.} \quad A \in \mathcal{C}_A,\ B \in \mathcal{C}_B.$$

Accounts for possible noise or error in the data or model: $Y = AB + E$

Solve by alternatingly finding the best A and B, holding the other fixed,

$$A^{k+1} \longleftarrow \arg\min_{A} \left\|AB^k - Y\right\|^2 \quad \text{s.t.} \quad A \in \mathcal{C}_A$$

$$B^{k+1} \longleftarrow \arg\min_{B} \left\|A^{k+1}B - Y\right\|^2 \quad \text{s.t.} \quad B \in \mathcal{C}_B.$$

Least-squares and *Block Gradient Descent*

Reframe the problem as minimizing the error between Y and AB,

$$\min_{A,B} \|AB - Y\|^2 \quad \text{s.t.} \quad A \in \mathcal{C}_A,\ B \in \mathcal{C}_B.$$

Accounts for possible noise or error in the data or model: $Y = AB + E$

Solve by alternatingly *decreasing the objective* $f(A, B) = \|AB - Y\|^2$,

$$A^{k+1} \longleftarrow P_A\left(A^k - \alpha^k \nabla_A f\left(A^k, B^k\right)\right)$$

$$B^{k+1} \longleftarrow P_B\left(B^k - \beta^k \nabla_B f\left(A^{k+1}, B^k\right)\right).$$

Least-squares and *Block Gradient Descent*

Reframe the problem as minimizing the error between Y and AB,

$$\min_{A,B} \|AB - Y\|^2 \quad \text{s.t.} \quad A \in \mathcal{C}_A,\ B \in \mathcal{C}_B.$$

Accounts for possible noise or error in the data or model: $Y = AB + E$

Solve by alternatingly *decreasing the objective* $f(A,B) = \|AB - Y\|^2$,

$$A^{k+1} \longleftarrow P_A\left(A^k - \alpha^k \nabla_A f\left(A^k, B^k\right)\right)$$

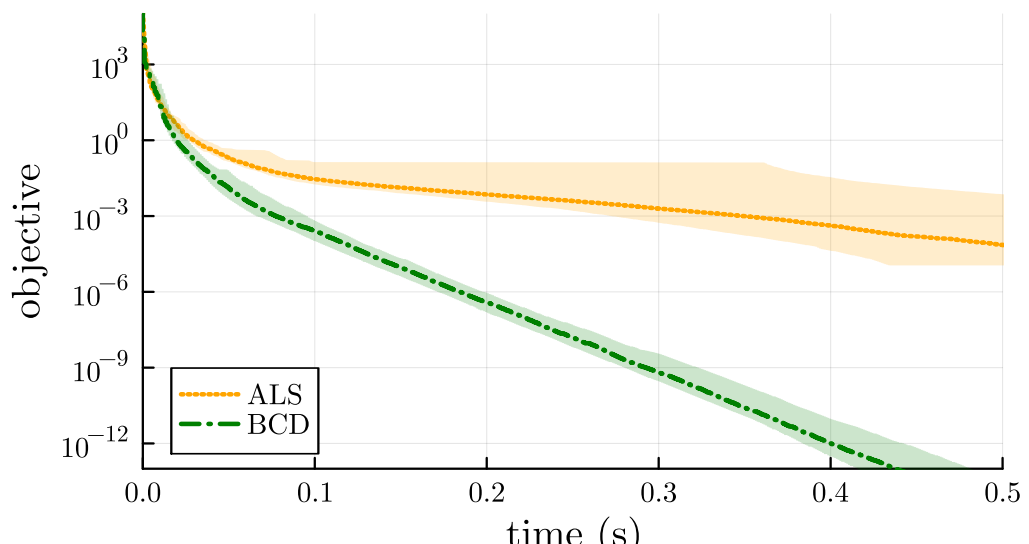
$$B^{k+1} \longleftarrow P_B\left(B^k - \beta^k \nabla_B f\left(A^{k+1}, B^k\right)\right).$$

with stepsizes

$$\alpha^k = 1/L_A^k = 1/\left\|B^k B^{k\top}\right\|_{\text{op}} \quad \& \quad \beta^k = 1/L_B^k = 1/\left\|\left(A^{k+1}\right)^\top A^{k+1}\right\|_{\text{op}}.$$

New Evidence in Favour of Block Gradient Descent

Algorithm	Update Rule for A (update for B is similar)
ALS	$A \leftarrow \arg\min_A \|AB - Y\|^2 \quad \text{s.t.} \quad A \geq 0$
BCD	$A \leftarrow P_{\geq 0}(A - \alpha(AB - Y)B^\top), \quad \alpha = \frac{1}{\|BB^\top\|_{\text{op}}}$



BlockTensorFactorization.jl [Richardson *et al.* 2025]

Fast & Flexible Tensor Factorizations in Julia

There's lots of existing packages...that cannot handle most constraints.

So I made one! Sample code:

```
using BlockTensorFactorization

X, stats, kwargs = factorize(Y;
  model=Tucker1, constraints=simplex_rows!, rank=5)

B, A = factors(X) # product of factors B ×₁ A = X
@assert X ≈ Y
@assert sum.(eachrow(A)) .== 1
@assert all(A .≥ 0)
```

Suite of algorithm options are available!

Example arguments

```
converged = [GradientNorm, RelativeError]
random_order = true # update factors randomly
```

Calculate stepsize and update factors

```
steps = LipschitzStep(X -> norm(factor(X, 1))^2)
update! = GradientDescent(n, gradient, steps)
```

Example info you can request

```
stats[end, :ObjectiveValue] # final objective value
kwargs[:tolerance] # convergence tolerance used
```

NEW: Custom Objectives with Automatic Differentiation

- No longer restricted to the Euclidean 2-norm!
- Implementation for KL-Divergence, Cross Entropy, and all p-norms
- Can apply these objectives "slicewise"

```
objective = RowWiseObjective(KLDivergence)
```

- Uses `ReverseDiff` under the hood
- Alternate stepsize methods available

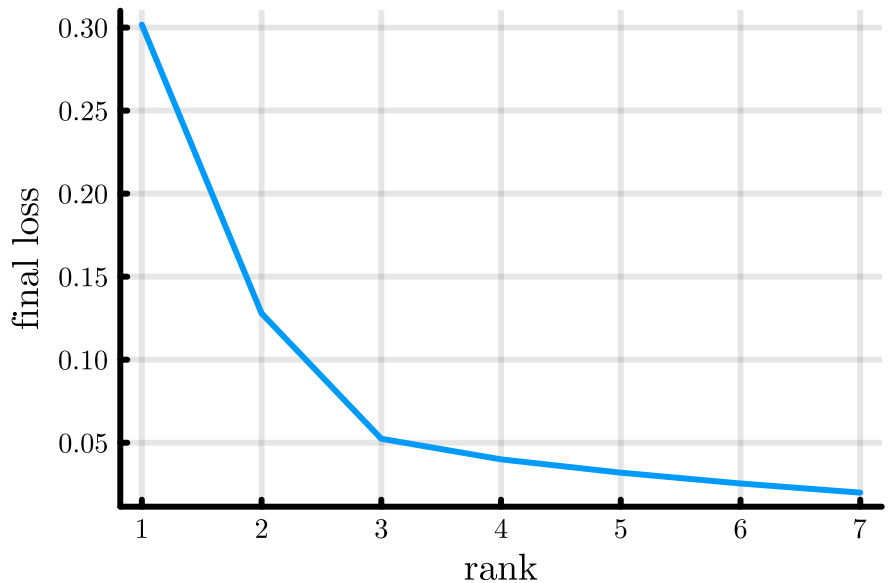
```
steps = [ArmijoStep, SecantStep, SPGStep]
```

Algorithm Innovations:

- Rank Estimation
- Accelerated Convergence
- Rescaling Constraints
- Multiscaled Factorization

Tenor Rank Estimation

- Usually the rank R needs to be known in advance to factorize.
- Can we try many ranks and pick the right one?



- Larger ranks ($\longrightarrow$) gives smaller error ($\downarrow$) but eventually fit noise.
- **Goal**: Find the simplest model that can explain most of the data.

Tensor Rank Estimation

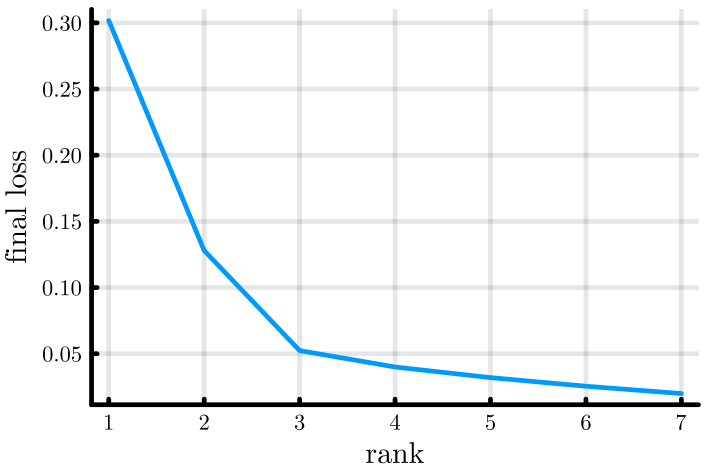
Estimate rank using the maximum curvature:

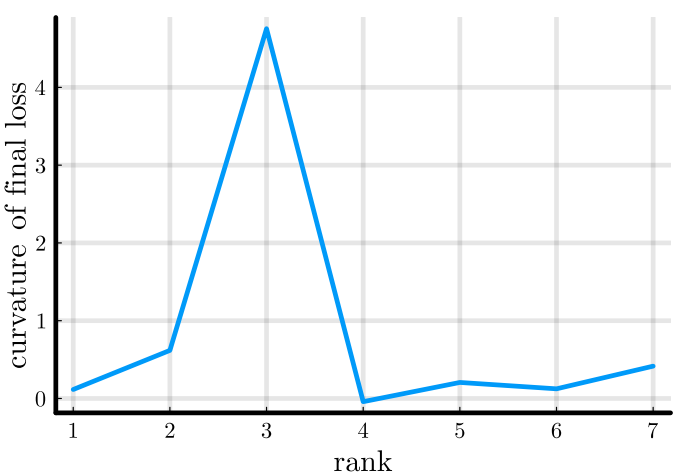
- Let X_r^* be the rank r factorization of Y
- Define the rank-r relative error:

$$\ell(r) = \frac{\|X_r^* - Y\|_F}{\|Y\|_F}$$

- Pick the r that maximizes curvature:

$$\hat{R} = \arg\max_r \kappa_\ell(r) := \frac{\ell''(r)}{\left(1 + (\ell'(r))^2\right)^{\frac{3}{2}}}$$





Accelerated Descent: Subblock Updates

Idea: Update smaller-sized blocks (columns a_r vs. the whole matrix A)

$$(1)\ A^{k+1} \longleftarrow A^k - \frac{1}{L^k} \nabla_A f\left(A^k, B^k\right)$$

Accelerated Descent: Subblock Updates

Idea: Update smaller-sized blocks (columns a_r vs. the whole matrix A)

$$(1)\ A^{k+1} \longleftarrow A^k - \frac{1}{L^k}\nabla_A f\left(A^k, B^k\right)$$

$$(2)\ a_r^{k+1} \longleftarrow a_r^k - \frac{1}{L_r^k}\nabla_{a_r} f\left(\left[a_1^{k+1}, ..., a_r^k, ..., a_R^k\right], B^k\right) \quad \forall r = 1, ..., R$$

Accelerated Descent: Subblock Updates

Idea: Update smaller-sized blocks (columns a_r vs. the whole matrix A)

$$(1)\ A^{k+1} \longleftarrow A^k - \frac{1}{L^k}\nabla_A f\left(A^k, B^k\right)$$

$$(2)\ a_r^{k+1} \longleftarrow a_r^k - \frac{1}{L_r^k}\nabla_{a_r} f\left(\left[a_1^{k+1}, ..., a_r^k, ..., a_R^k\right], B^k\right) \quad \forall r = 1, ..., R$$

$$(3)\ a_r^{k+1} \longleftarrow a_r^k - \frac{1}{L_r^k}\nabla_{a_r} f\left(\left[a_1^k, ..., a_r^k, ..., a_R^k\right], B^k\right) \quad \forall r = 1, ..., R$$

Accelerated Descent: Subblock Updates

Idea: Update smaller-sized blocks (columns a_r vs. the whole matrix A)

$$(1)\ A^{k+1} \longleftarrow A^k - \tfrac{1}{L^k}\nabla_A f\left(A^k, B^k\right)$$

$$(2)\ a_r^{k+1} \longleftarrow a_r^k - \tfrac{1}{L_r^k}\nabla_{a_r} f\left(\left[a_1^{k+1}, ..., a_r^k, ..., a_R^k\right], B^k\right) \quad \forall r = 1, ..., R$$

$$(3)\ a_r^{k+1} \longleftarrow a_r^k - \tfrac{1}{L_r^k}\nabla_{a_r} f\left(\left[a_1^k\ , ..., a_r^k, ..., a_R^k\right], B^k\right) \quad \forall r = 1, ..., R$$

$$\equiv A^{k+1} \longleftarrow A^k - \nabla_A f\left(A^k, B^k\right)\left(\hat{L}^k\right)^{-1}$$

where $\hat{L}^k = \begin{bmatrix} L_r^k & & \\ & \ddots & \\ & & L_R^k \end{bmatrix}$

Accelerated Descent: Subblock Updates

Idea: Update smaller-sized blocks (columns a_r vs. the whole matrix A)

$$(1)\ A^{k+1} \longleftarrow A^k - \tfrac{1}{L^k}\nabla_A f\left(A^k, B^k\right)$$

$$(2)\ a_r^{k+1} \longleftarrow a_r^k - \tfrac{1}{L_r^k}\nabla_{a_r} f\left(\left[a_1^{k+1}, ..., a_r^k, ..., a_R^k\right], B^k\right) \quad \forall r = 1, ..., R$$

$$(3)\ a_r^{k+1} \longleftarrow a_r^k - \tfrac{1}{L_r^k}\nabla_{a_r} f\left(\left[a_1^k\ , ..., a_r^k, ..., a_R^k\right], B^k\right) \quad \forall r = 1, ..., R$$

$$\equiv A^{k+1} \longleftarrow A^k - \nabla_A f\left(A^k, B^k\right)\left(\hat{L}^k\right)^{-1}$$

where $\hat{L}^k = \begin{bmatrix} L_r^k & & \\ & \ddots & \\ & & L_R^k \end{bmatrix}$ approximates the Hessian $\nabla_A^2 f\left(A^k, B^k\right)$.

Accelerated Descent: Subblock With Momentum

Subblock updates compatible with the classical first-order acceleration,

$$A^{k+\frac{1}{2}} \leftarrow A^k + \delta\sqrt{L^{k-1}/L^k}\left(A^k - A^{k-1}\right).$$

Accelerated Descent: Subblock With Momentum

Subblock updates compatible with the classical first-order acceleration,

$$A^{k+\frac{1}{2}} \leftarrow A^k + \delta\sqrt{L^{k-1}/L^k}\left(A^k - A^{k-1}\right).$$

Modify to $$A^{k+\frac{1}{2}} \leftarrow A^k + \delta\sqrt{\hat{L}^{k-1}\left(\hat{L}^k\right)^{-1}}\left(A^k - A^{k-1}\right).$$

Accelerated Descent: Subblock With Momentum

Subblock updates compatible with the classical first-order acceleration,

$$A^{k+\frac{1}{2}} \leftarrow A^k + \delta\sqrt{L^{k-1}/L^k}\left(A^k - A^{k-1}\right).$$

Modify to $$A^{k+\frac{1}{2}} \leftarrow A^k + \delta\sqrt{\hat{L}^{k-1}\left(\hat{L}^k\right)^{-1}}\left(A^k - A^{k-1}\right).$$

	No Momentum	**Yes Momentum**
No Subblock	75.5 iterations	61.5 iterations (22.8% faster)
Yes Subblock	9.5 iterations (695% faster)	**9 iterations (739% faster)**

Rescaling Constraints

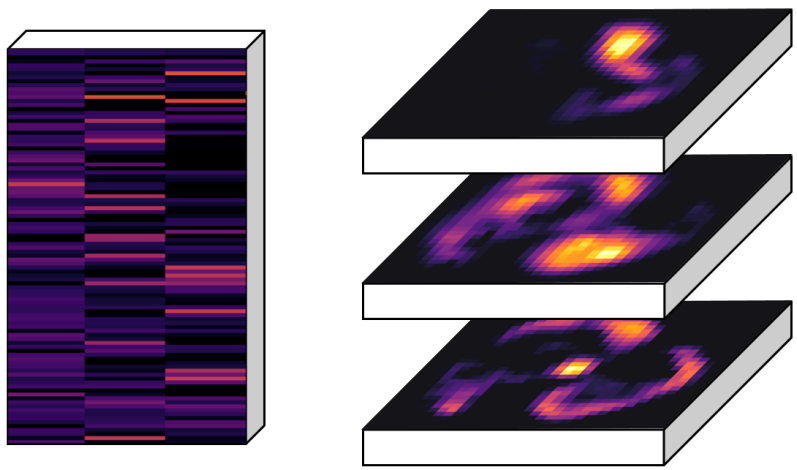
When demixing probabilities, you have the following constraints

- Convex mixtures: $A \geq 0, \ \sum_r A[i, r] = 1$
- Density Sources: $B \geq 0, \ \sum_{j_1,\ldots,j_N} B[r, j_1, \ldots, j_N] = 1$

This gives you the updates:

$$A^{k+1} \longleftarrow P_A\left(A^k - \alpha^k \nabla_A f\left(A^k, B^k\right)\right)$$

$$B^{k+1} \longleftarrow P_B\left(B^k - \beta^k \nabla_B f\left(A^{k+1}, B^k\right)\right)$$



Use KL-projection instead of Euclidean

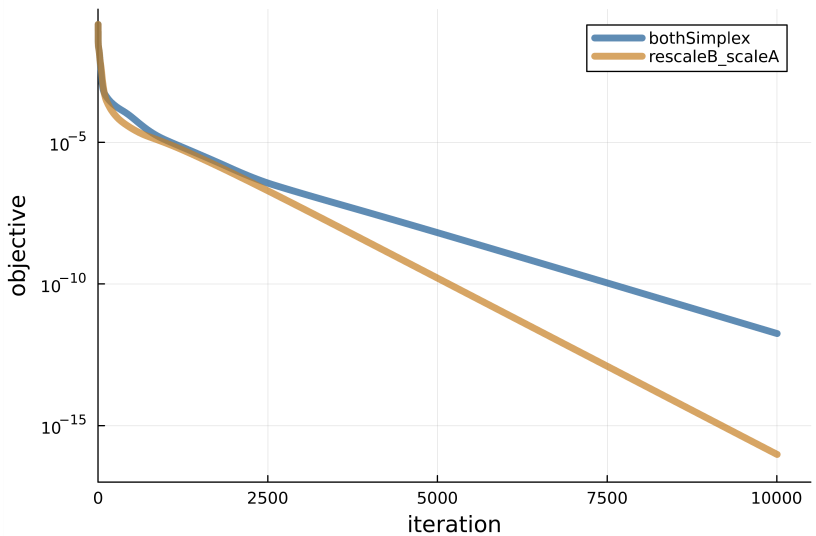
- Project nonnegatively, divide by the sum, and move the weight to A:

$$A^{k+\frac{1}{2}} \longleftarrow P_{\geq 0}\left(A^k - \alpha^k \nabla_A f\left(A^k, B^k\right)\right)$$

$$B^{k+\frac{1}{2}} \longleftarrow P_{\geq 0}\left(B^k - \beta^k \nabla_B f\left(A^{k+\frac{1}{2}}, B^k\right)\right)$$

$$C[r,r] \longleftarrow \sum_{j_1,\ldots,j_N} B[r, j_1, \ldots, j_N]$$

$$\left(A^{k+1}, B^{k+1}\right) \longleftarrow \left(A^{t+\frac{1}{2}} C,\ C^{-1} B^{k+\frac{1}{2}}\right)$$



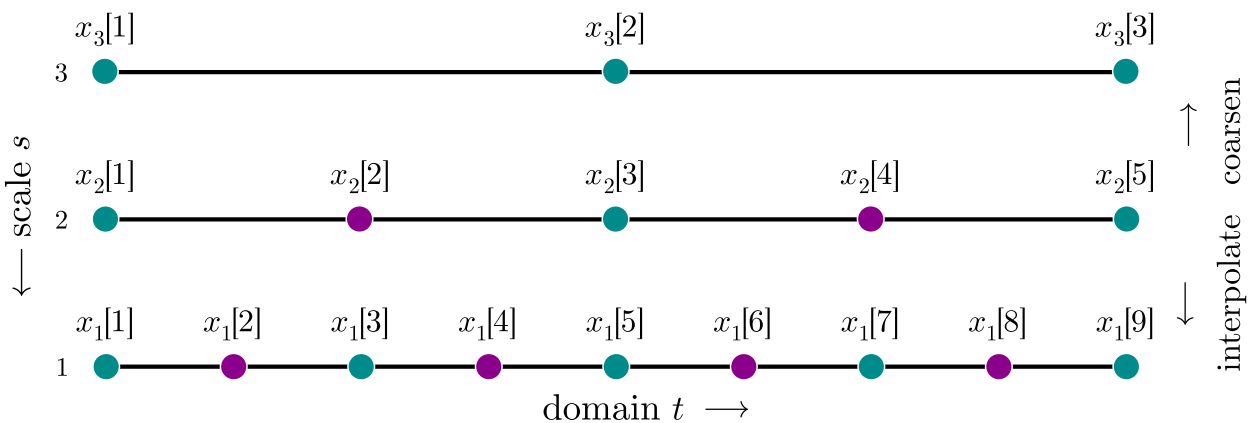
- We do not explicitly enforce the linear constraint on A
- Constraint is implied since Y and B normalized $\Longrightarrow$ A normalized
- Product $AB = (AC)(C^{-1}B) = \tilde{A}\tilde{B}$ is preserved

Multiscaled Optimization [Richardson *et al.* 2025]

- Consider demixing 1D continuous probability densities $Y[i,:]$,

$$\min_{A,B\geq 0} \|Y - AB\|^2 \quad \text{s.t.} \quad \|A[i,:]\|_1 = 1, \ \|B[r,:]\|_1 = 1.$$

- If underlying data is continuous $\Longrightarrow$ factorize at multiple scales!
- How? Approximately solve a series of progressively finer problems:



Multiscale Optimization [Richardson *et al.* 2025]

Series of problems P_s for scale $s = S, \ldots, 1$:

$$\min_{A, B_s \geq 0} \|Y_s - AB_s\|^2 \quad \text{s.t.} \quad \|A[i,:]\|_1 = 1, \ \|B_s[r,:]\|_1 = \frac{1}{2^{S-s+1}}$$

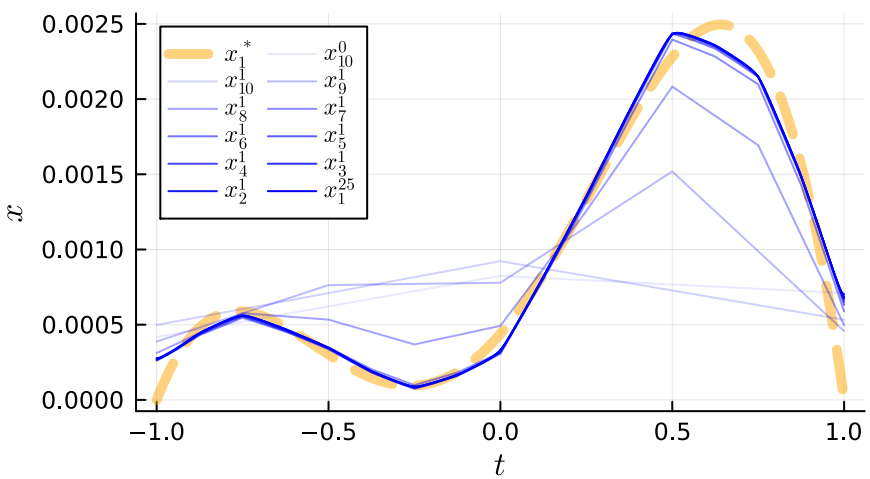
Trick: Interpolate solution at one scale to to warm start the next scale

Multiscale Optimization [Richardson *et al.* 2025]

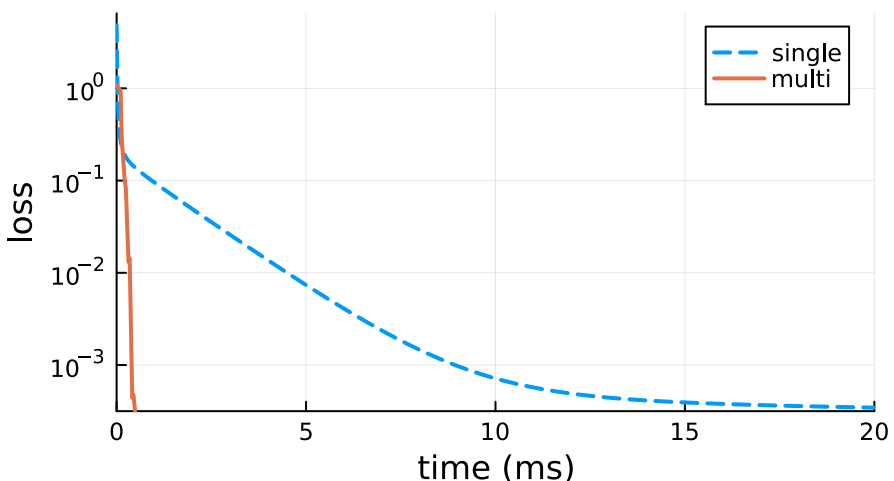
Series of problems P_s for scale $s = S, \ldots, 1$:

$$\min_{A, B_s \geq 0} \|Y_s - AB_s\|^2 \quad \text{s.t.} \quad \|A[i,:]\|_1 = 1, \ \|B_s[r,:]\|_1 = \frac{1}{2^{S-s+1}}$$

Trick: Interpolate solution at one scale to to warm start the next scale



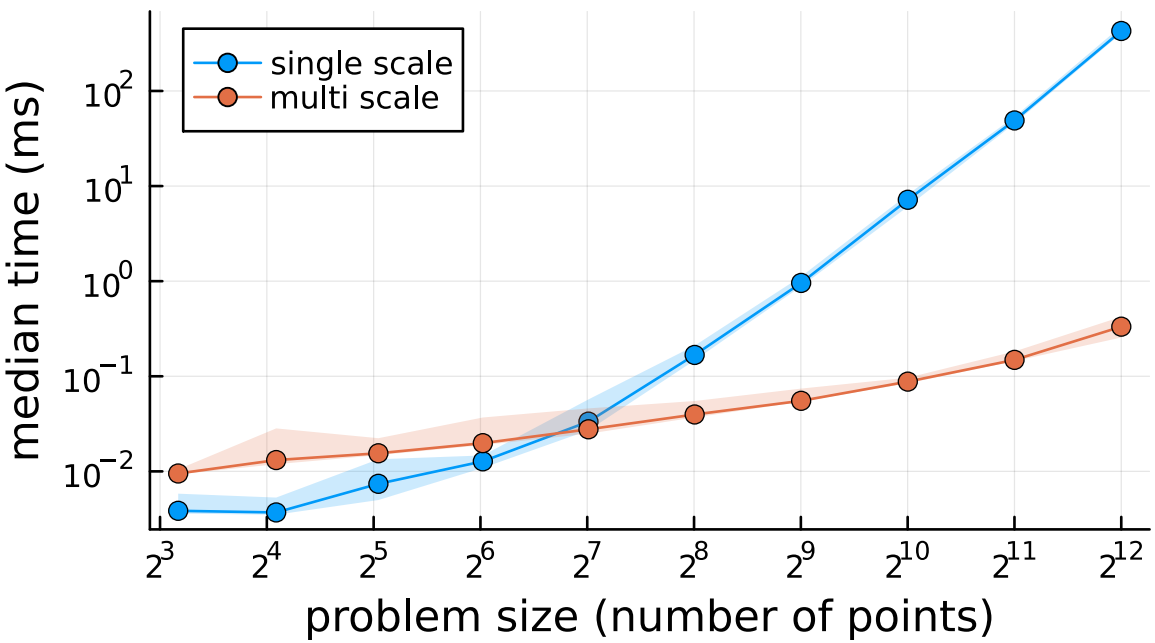
Example iterate convergence



Objective function convergence comparison

Multiscale Optimization [Richardson *et al.* 2025]

- Multiscale advantage grows as the number of points increases

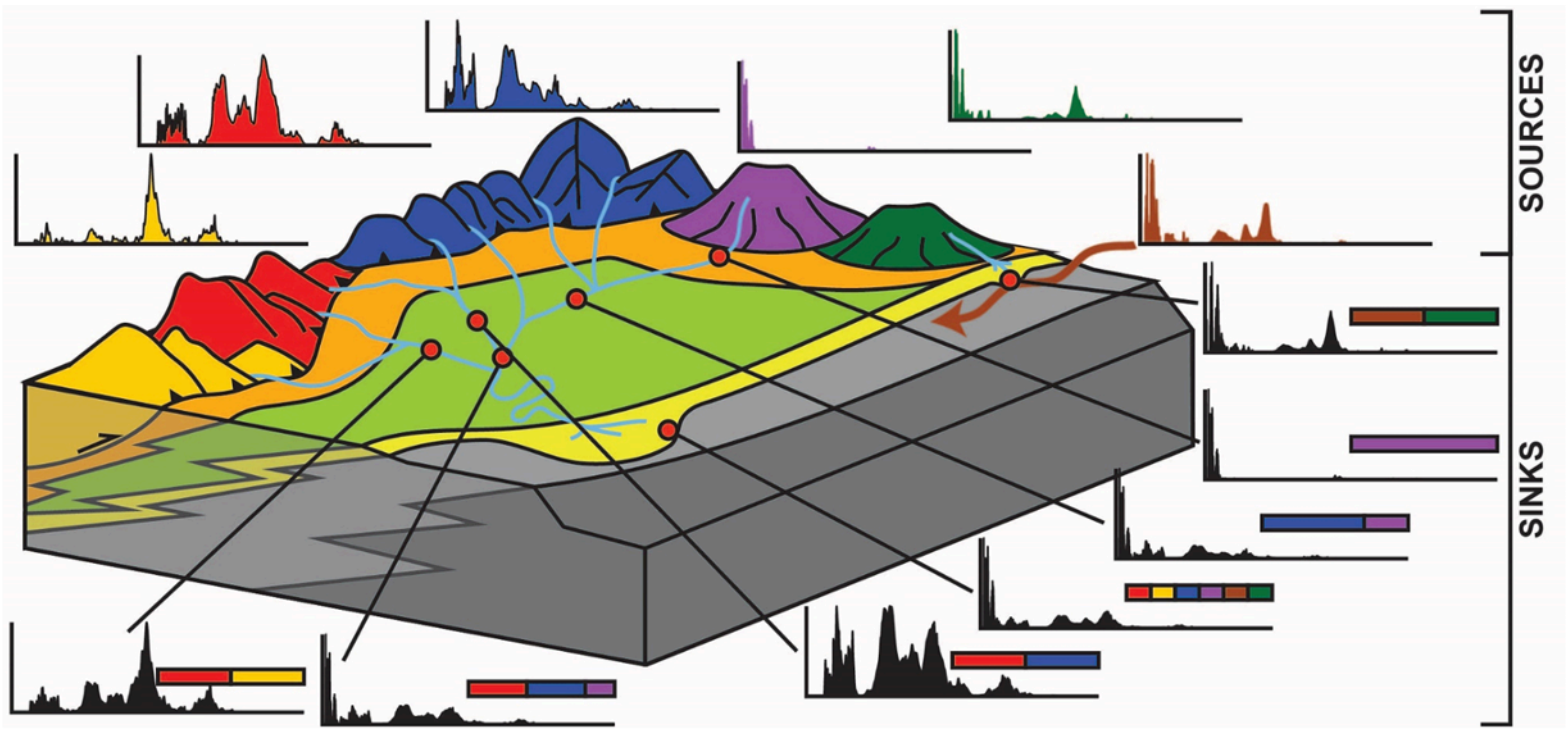


- It's only better when your problem gets harder with more points

Applications

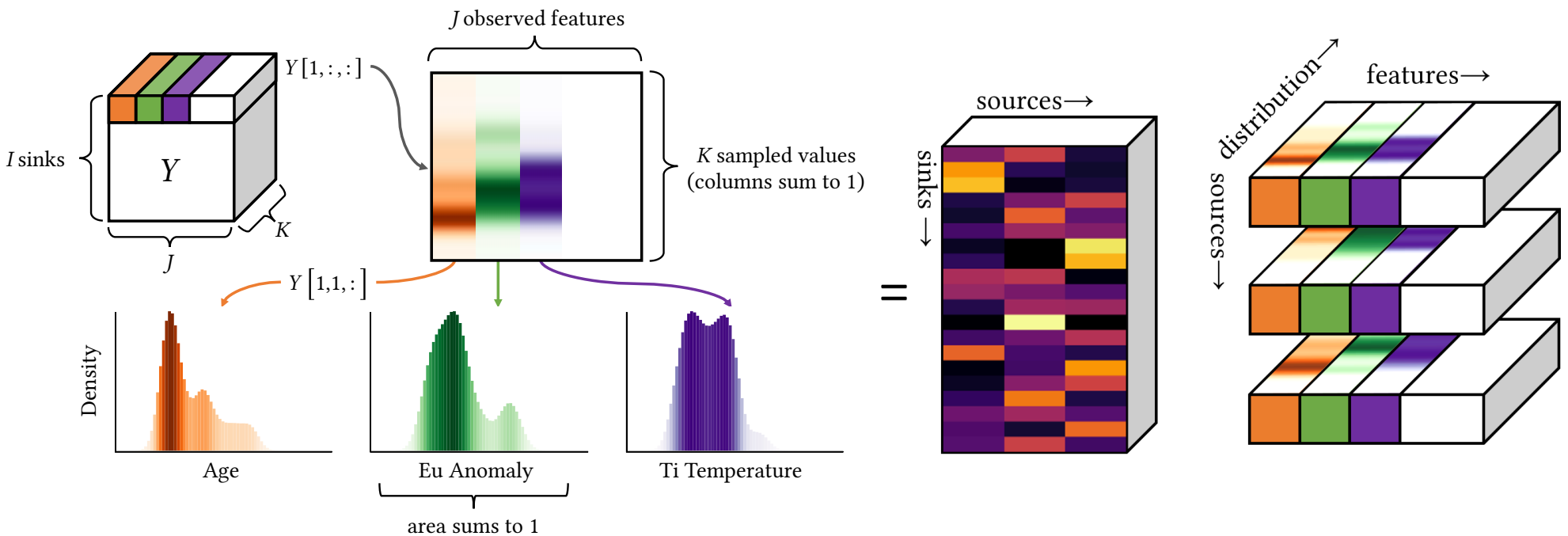
Geology [Graham *et al.* 2025, Saylor *et al.* 2025]

- Measure mineral distributions of rock samples at varying locations
- Determine upstream sources and how much they were mixed



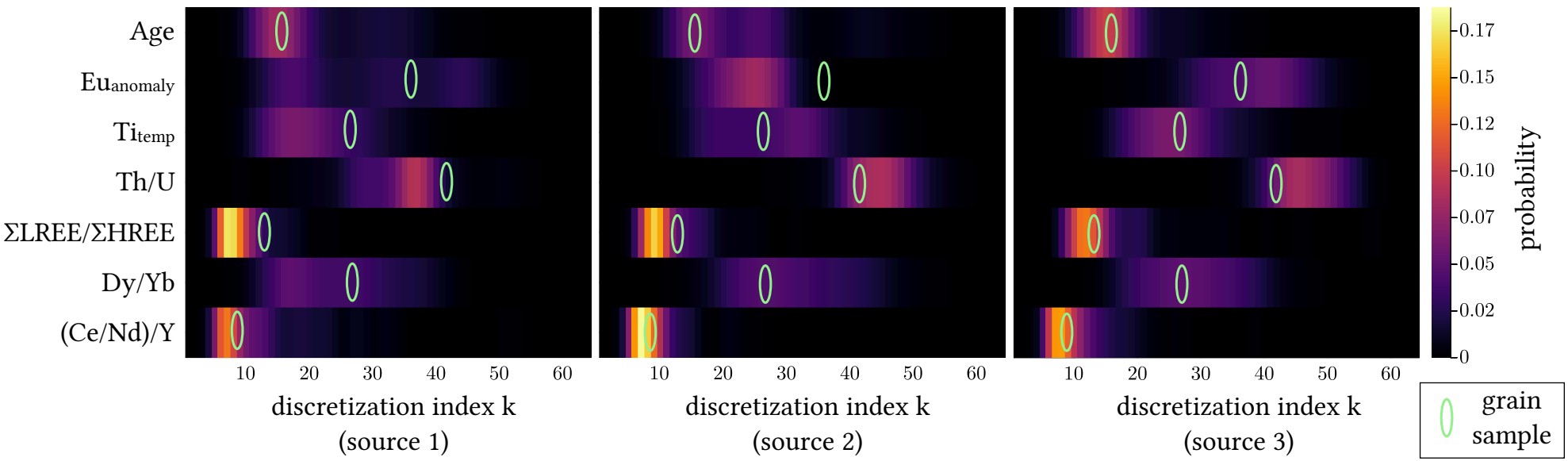
Geology [Graham *et al.* 2025, Saylor *et al.* 2025]

- Collect data into a single 3D tensor
- Each fibre is the distribution of a particular feature at one location



Geology [Graham *et al.* 2025, Saylor *et al.* 2025]

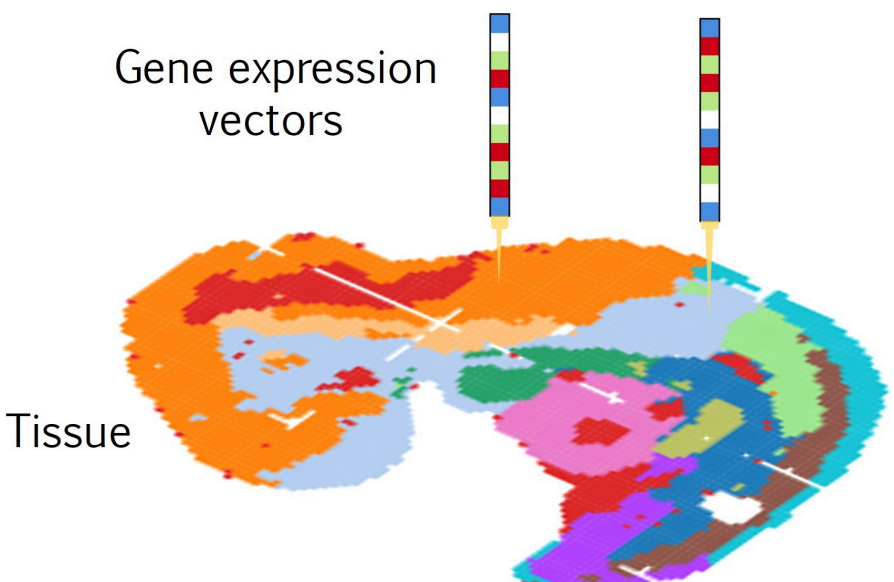
- Use the learned mixtures to estimate where each grain came from
- Assign probabilities $p_r = \mathbb{P}\left(g \in \mathcal{V}_g \mid g \sim B_r\right) \approx \prod_{j=1}^{J} B\left[r, j, \hat{k}_j\right]$



- Estimate how confidence in the prediction with $\log_{10}\left(p_{(1)}/p_{(2)}\right)$

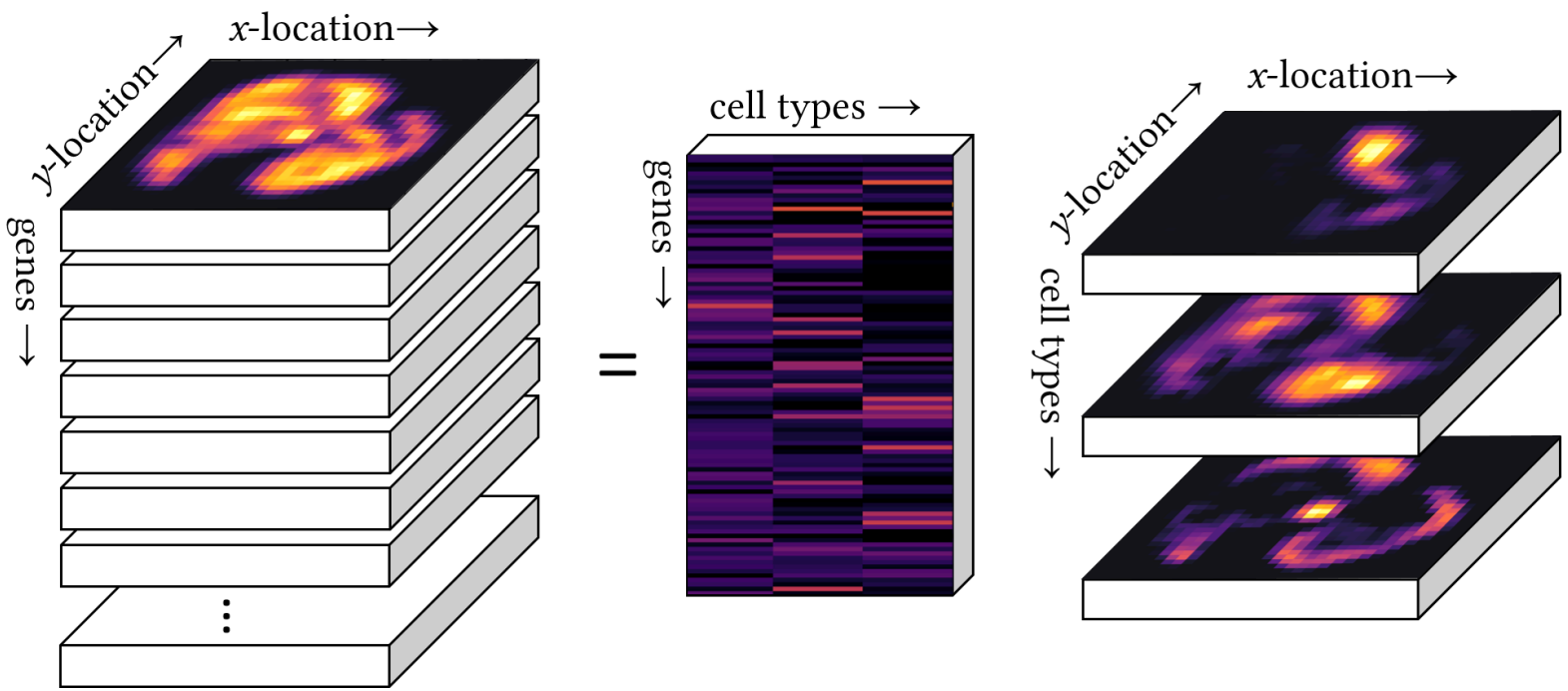
Spatial Transcriptomics

- Count the presence of thousands of genes on a 2D grid
- **Goal:** Label regions of an embryo according to its types of cells
- Lets us track development across multiple embryos



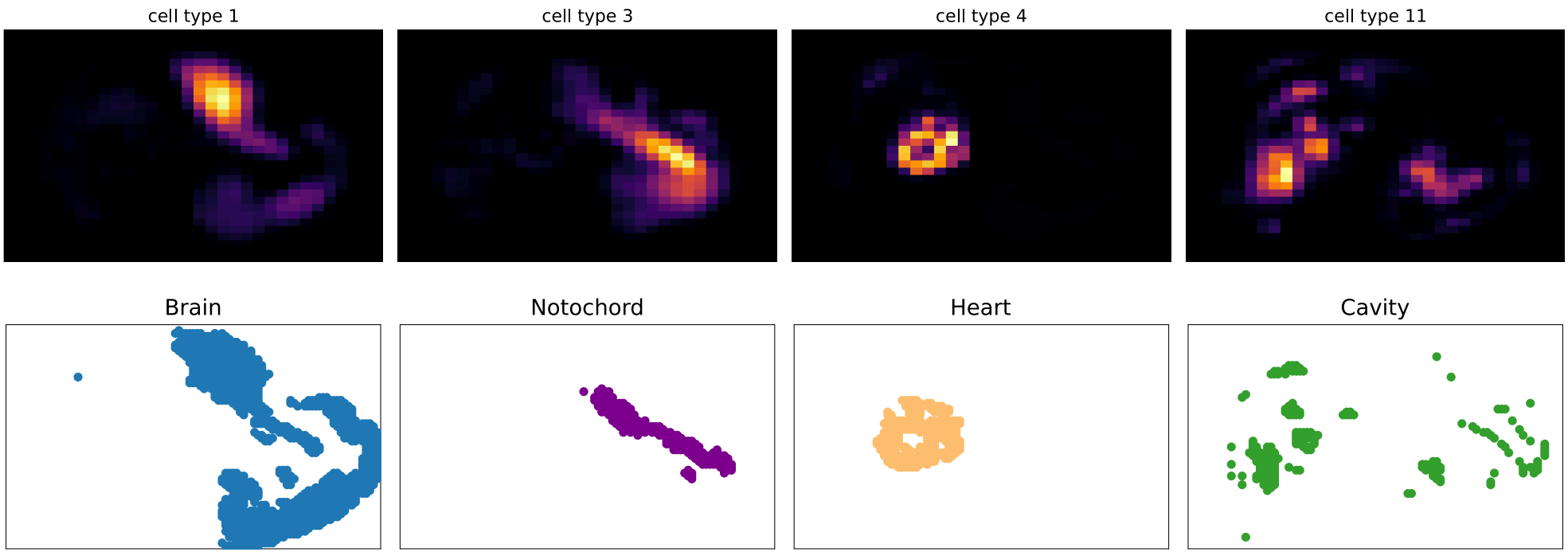
Spatial Transcriptomics

- Stack heatmaps of each gene into a 3D tensor
- Uncover gene expressions and spatial distributions of these cell types



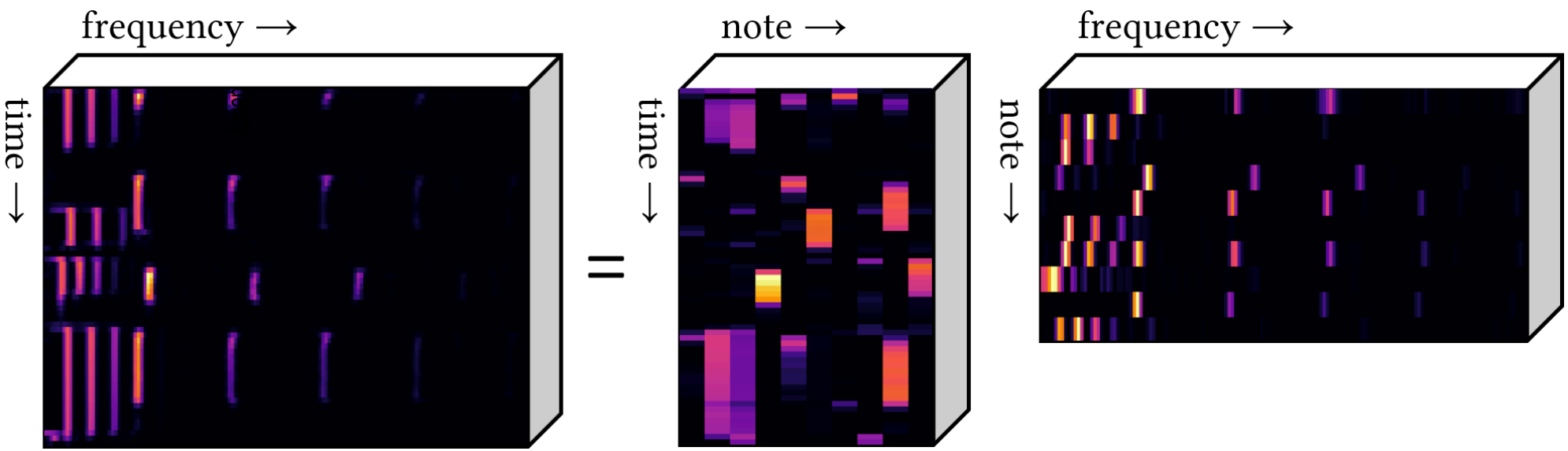
Spatial Transcriptomics

- Learned cell types spatial presence compared to true cell types.



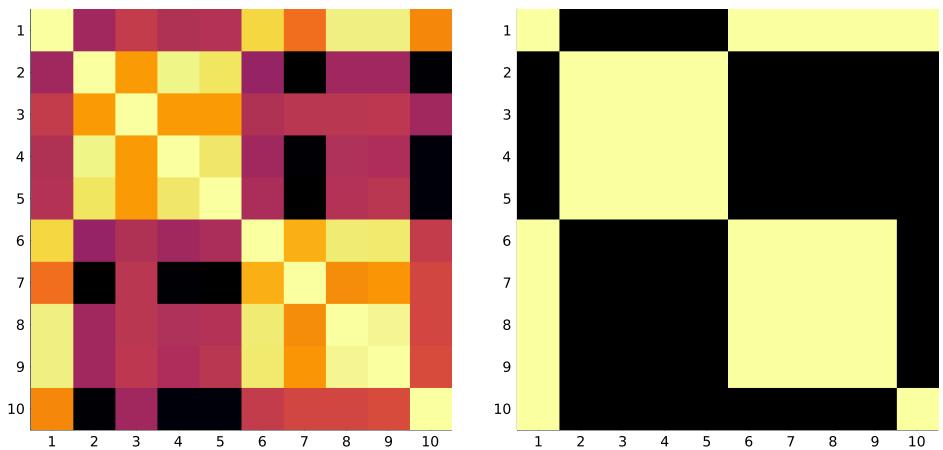
Music

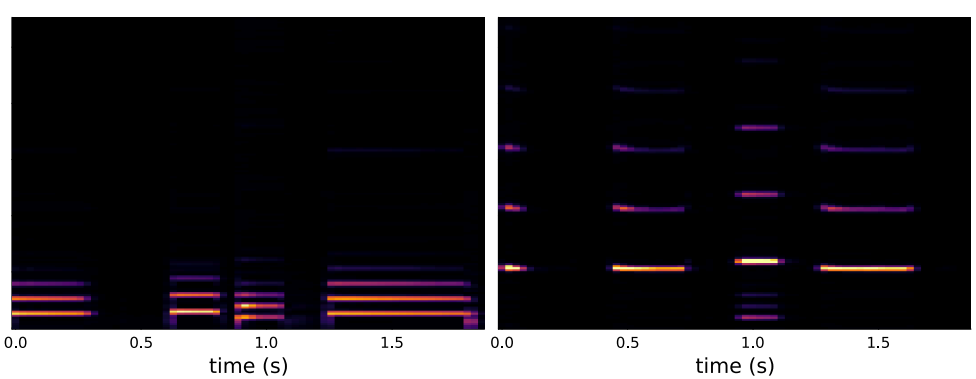
- Given a song, extract distinct instruments (e.g. guitars, vocals, drums)
- Use the short-time Fourier transform as input
- Factorize into individual notes' amplitudes A and frequencies B



Music

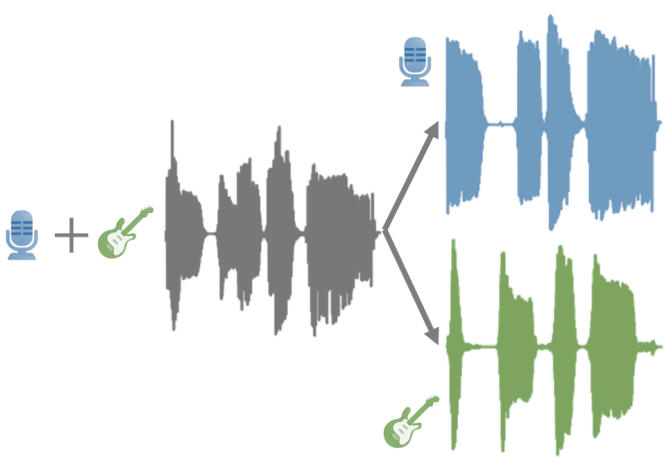
- Use cross-correlation to find notes with similar frequency spectrums
- (Top-right) Maximum cross-correlation matrix before and after thresholding
- (Bottom-right) Reconstruct a spectrogram for each grouping of notes
- (if time) Hear the results!

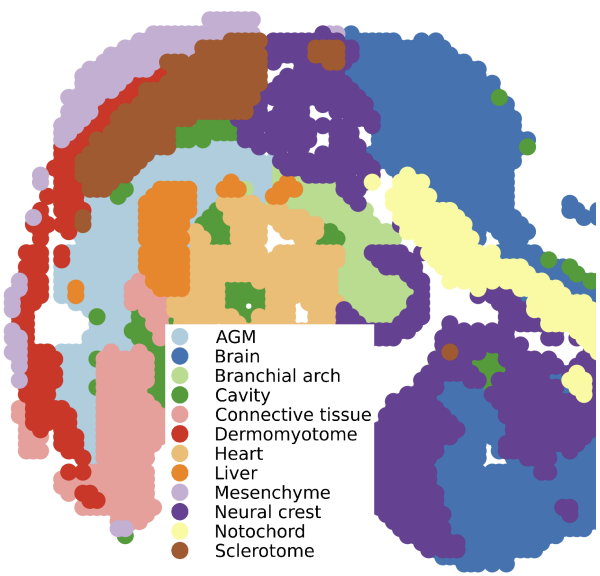


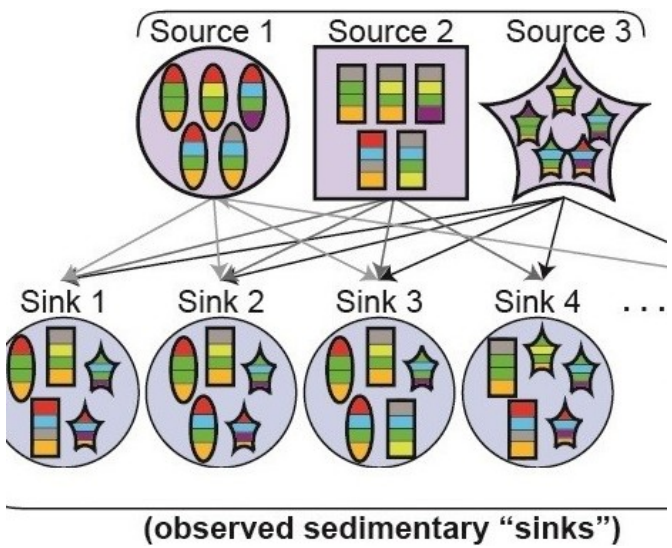


Summary

- Understand complex data by de-mixing into simpler sources
- Unsupervised learning: model these problems as tensor factorizations
- Advance practical methods (techniques & code) for computing them







References

[1] **N. Richardson**, N. Marusenko, and M. P. Friedlander, "BlockTensorFactorization.jl v0.4.0," 2025. https://github.com/MPF-Optimization-Laboratory/BlockTensorFactorization.jl

[2] **N. J. E. Richardson**, N. Marusenko, and M. P. Friedlander, "Multiple Scale Methods For Optimization Of Discretized Continuous Functions," Dec. 2025. http://arxiv.org/abs/2512.13993

[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] J. E. Saylor, **N. Richardson**, N. Graham, R. G. Lee, and M. P. Friedlander, "Endmember modelling of detrital zircon petrochronology data via multivariate Tucker-1 tensor decomposition," technical report EGU25–4081, Mar. 2025.

Website:
https://njericha.github.io

Additional Slides

Projected Gradient Descent as a Quadratic Minimization

- Rather than minimize a function directly $\min_x f(x)$
- Approx. $f(x) \approx f(a) + \nabla f(a) \cdot (x - a) + \frac{1}{2}(x - a)^\top \nabla^2 f(a)(x - a)$
- Use L-smoothness to upper bound the Hessian $\nabla^2 f(a) \preceq LI$
- Find the minimum of the quadratic upper bound:

$$\begin{aligned}
&\arg\min_{x \in C} \; f(x) \\
&\leq \arg\min_{x \in C} \; f(a) + \nabla f(a) \cdot (x - a) + \frac{1}{2}(x - a)^\top LI(x - a) \\
&= P_C\left(a - \frac{1}{L}\nabla f(a)\right)
\end{aligned}$$

- Start at $x^k = a$, can get a closer minimum $x^{k+1} = x^k - \frac{1}{L}\nabla f\left(x^k\right)$

PGD as Quadratic Minimization Derivation

$$\arg\min_{x \in C} \; f(a) + \langle \nabla f(a), x - a \rangle + \frac{L}{2}\|x - a\|^2$$

$$= \arg\min_{x \in C} \langle \nabla f(a), x - a \rangle + \frac{L}{2}\|x - a\|^2$$

$$= \arg\min_{x \in C} \langle \nabla f(a), x - a \rangle + \frac{L}{2}\|x - a\|^2 + \frac{1}{2L}\|\nabla f(a)\|^2$$

$$= \arg\min_{x \in C} \; \frac{L}{2}\left\|(x - a) + \frac{1}{L}\nabla f(a)\right\|^2$$

$$= \arg\min_{x \in C} \; \frac{L}{2}\left\|x - \left(a - \frac{1}{L}\nabla f(a)\right)\right\|^2 = P_C\left(a - \frac{1}{L}\nabla f(a)\right)$$

Convergence of Block Gradient Descent

Theorem 1 (Convergence of Block Descent): *Let f be smooth and block-convex. Then the block projected gradient descent algorithm with step sizes $\alpha_n^t = 1/\mathcal{S}_{f_n^t}$ converges to a* block-minimizer $(A_1^*, ..., A_N^*)$ *which satisfies, for all blocks $n \in [N]$,*

$$f(A_1^*, ..., A_n^*, ..., A_N^*) \leq f(A_1^*, ..., A_n, ..., A_N^*) \quad \textit{for all} \quad A_n \in \mathcal{C}_n.$$

- Proved in Xu & Yin 2013 for standard block PGD
- We extended to rescaling [Graham *et al.* 2025]
- Solution cannot be improved if you only change one block
- *may* be able to improve if you change multiple blocks at once

Can we allow any A and B?

- Most applications have **constraints** on the factors $AB = Y$.
- Constraining A only allows certain types of mixtures

$$A[m,r] \in \mathbb{R} \implies y_m \in \text{span}\{b_r\}$$

$$A[m,r] \geq 0 \implies y_m \in \text{cone}\{b_r\}$$

$$\sum_r A[m,r] = 1 \ \forall m \implies y_m \in \text{aff}\{b_r\}$$

$$A[m,r] \geq 0 \ \text{ and } \ \sum_r A[m,r] = 1 \ \forall m \implies y_m \in \text{conv}\{b_r\}$$

- Sources $\{b_r\}$ should have similar constraints as the mixtures $\{y_m\}$

Auto Diff Details

```
function make_gradient(X, n, Y, objective)
  decomp_type = typeof(X)

  f(factors...)= objective(decomp_type(factors), Y)
  compiled_f_tape = compile(GradientTape(f, factors(X)))

  function ∇f_n(X)
      G = gradient!(compiled_f_tape, factors(X))
      return G[n]
  end

  return ∇f_n
end
```

Numerical (Approximate) Rank

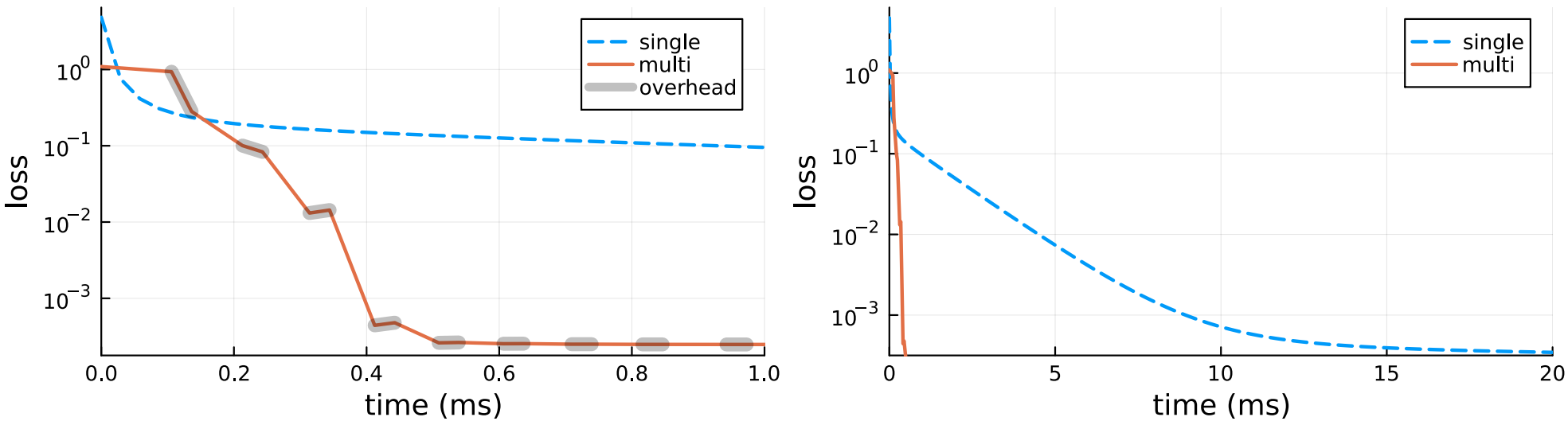
Definition 1 (ϵ-rank): $\operatorname{rank}_\epsilon(A) = \min\{\operatorname{rank}(A+E) \mid \|E\|_2 \le \epsilon\}$

Theorem 2: *Let* $Y = X + E \in \mathbb{R}^{m\times n}$ *where* $\operatorname{rank}(X) = r$. *Then,*

$$\|E\|_2 \le \epsilon < \sigma_r/2 \quad \Longrightarrow \quad \operatorname{rank}_\epsilon(Y) = \operatorname{rank}(X).$$

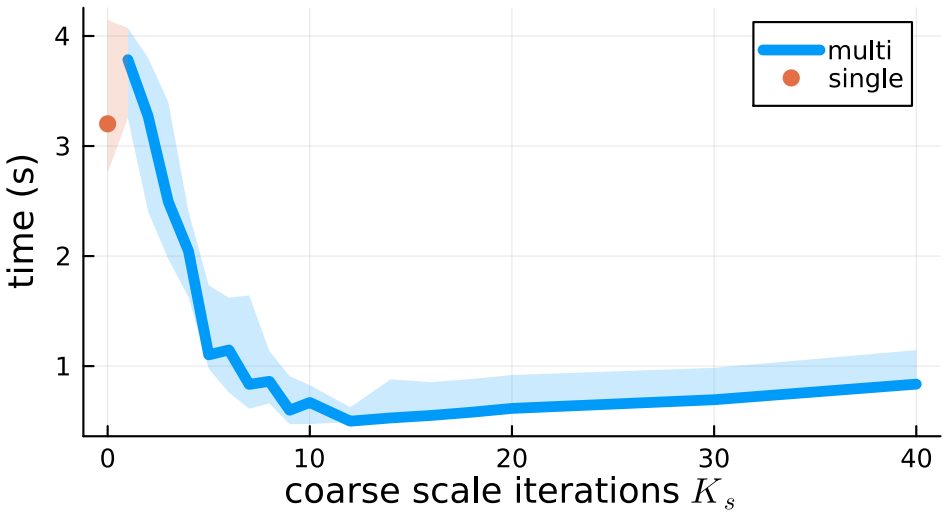
- The noisy data Y is approximately low rank $r = \operatorname{rank}(X)$
- The numerical rank is "blind" to small amounts of noise
- Unlike the standard rank which can jump when noise is added

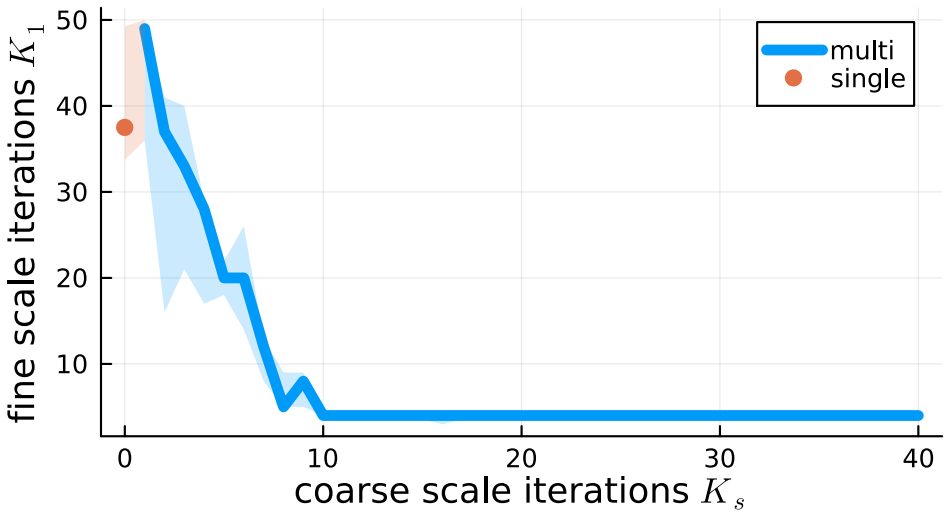
Multiscale Optimization Advantage Experiment 1



Left plot highlights overhead (interpolation, reallocation, etc.) in gray.

Multiscale Optimization Advantage Experiment 2





Left: How long does multiscale take if you do K_s iterations at each coarse scale.

Right: How many fine scale iterations K_1 are needed when you do K_s iterations at coarse scales.

Multiscale Optimization Advantage Theorem

- Assume solution functions are L-Lipschitz on the interval $[0, 1]$
- Use a q-linear update: $\left\|U\left(x^k\right) - x^*\right\|_2 \leq q\left\|x^k - x^*\right\|_2, 0 \leq q < 1$
- Cost of U scales polynomially in the # of points: $C \sim I^p$

Theorem 3: *If you perform multiscale with S scales where $2^S \geq L \,/\, \mathrm{poly}(q)$, then multiscale simultaneously achieves lower total cost (i.e. faster) and a tighter final error bound (i.e. more accurate) than single scale.*

For larger q (harder problems!), need **fewer scales** for it to be worth it.