Tucker-1 Demixing Model

Under the Hood of Our Density Separation

10 Dec 2025

Nicholas Richardson

Department of Mathematics

THE UNIVERSITY OF BRITISH COLUMBIA

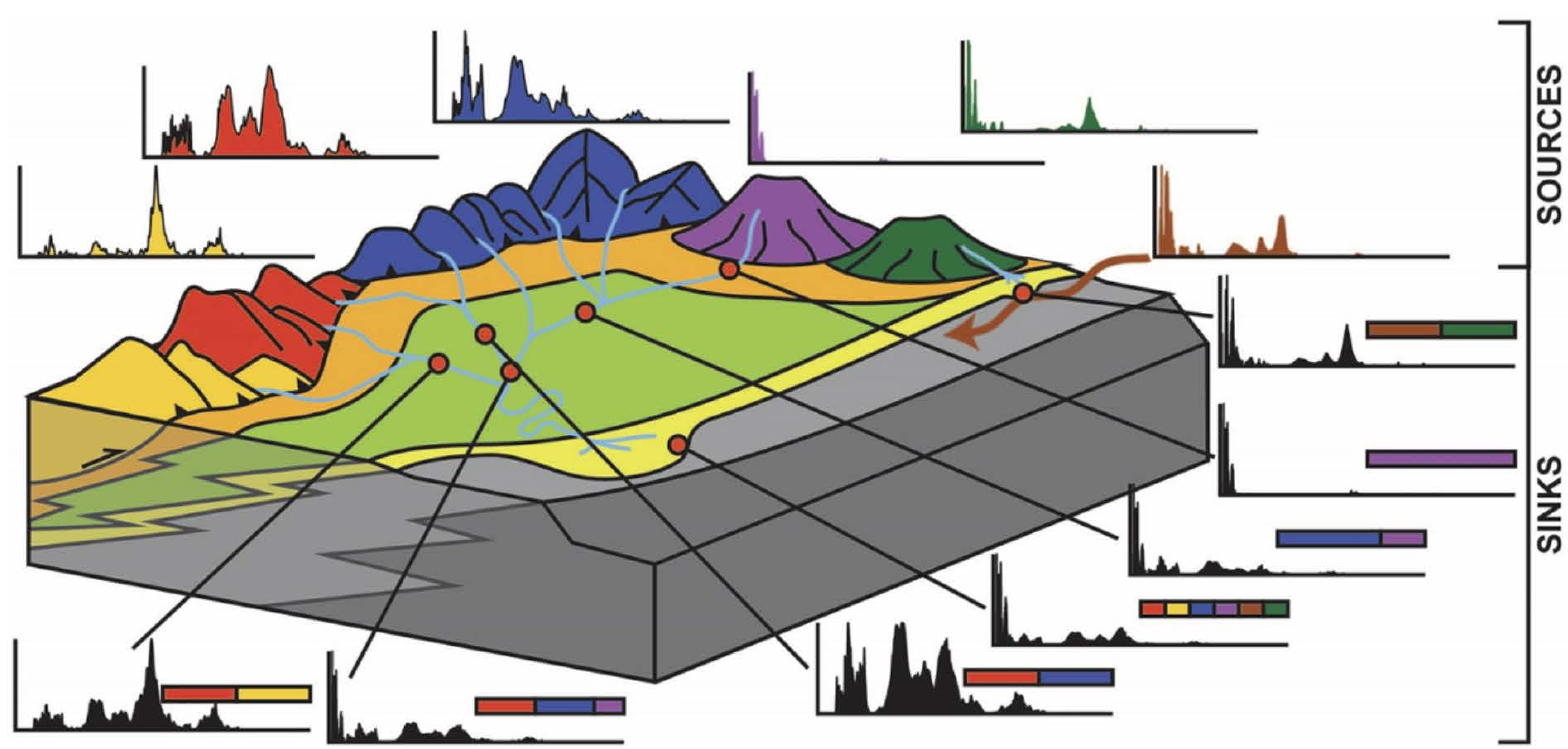
INSTITUTE OF APPLIED MATHEMATICS
University of British Columbia

Overview

- How we frame the problem
- The model and data pipeline
- Solution and implementation
- Most of this is taken from our paper [Graham *et al.* 2025]

Physical Problem

- Source locations have their own distribution of minerals
- Rocks from these sources are mixed & deposited downstream
- Take scoops of rocks at locations downstream (sinks)
- **Goal**: Learn source distributions from the sink measurements



Framing the demixing problem

- Notation: use regular letters for scalars $a, b, \ldots$
- bold lowercase for vectors $\mathbf{a}, \mathbf{b}, \ldots$
- bold upper case for matrices/tensors $\mathbf{A}, \mathbf{B}, \ldots$

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

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

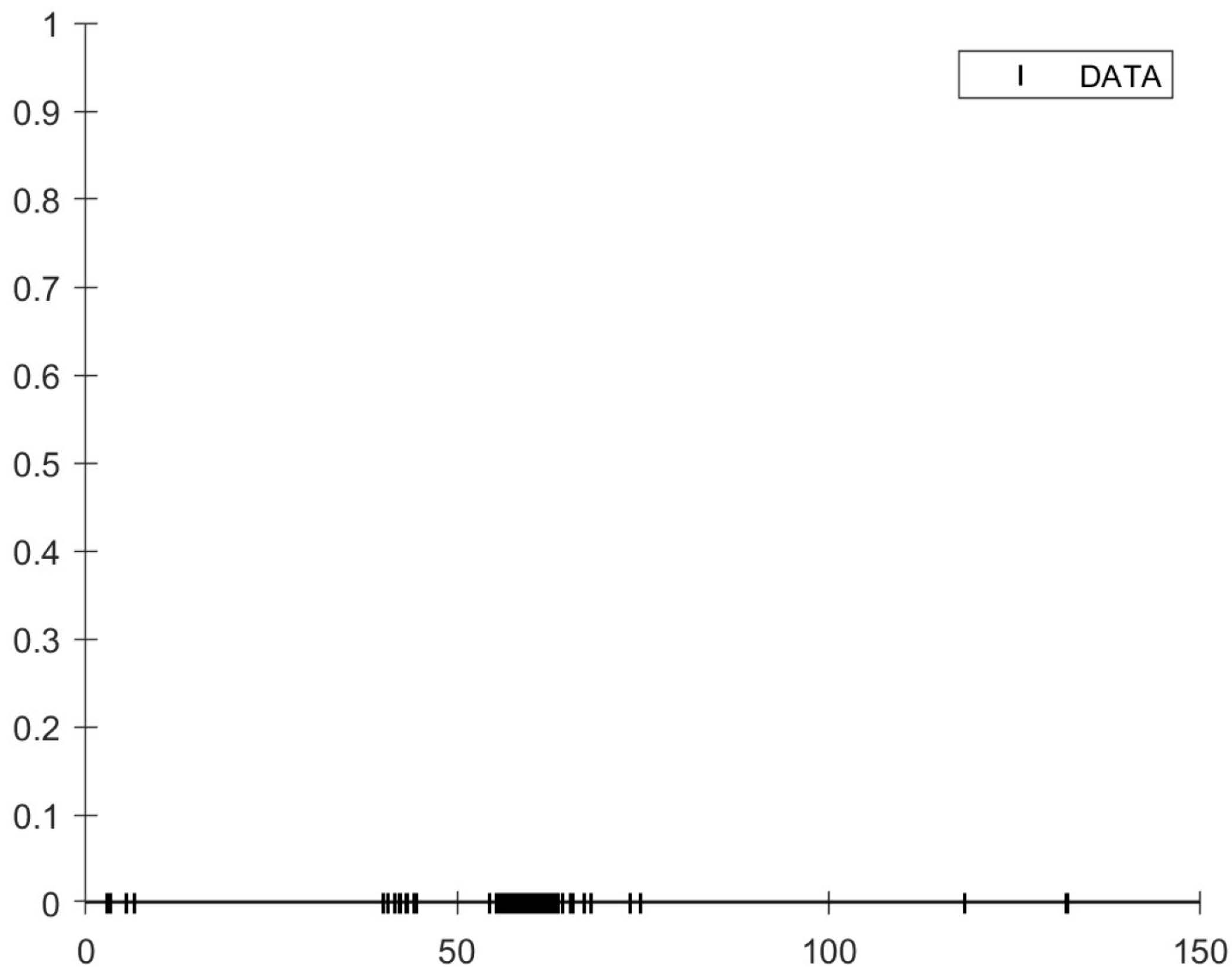
From rocks to densities $\mathbf{Y}_i$

- Sample grains of rock from many sinks
- Each grain belongs to some source
- Want to estimate the sink distributions

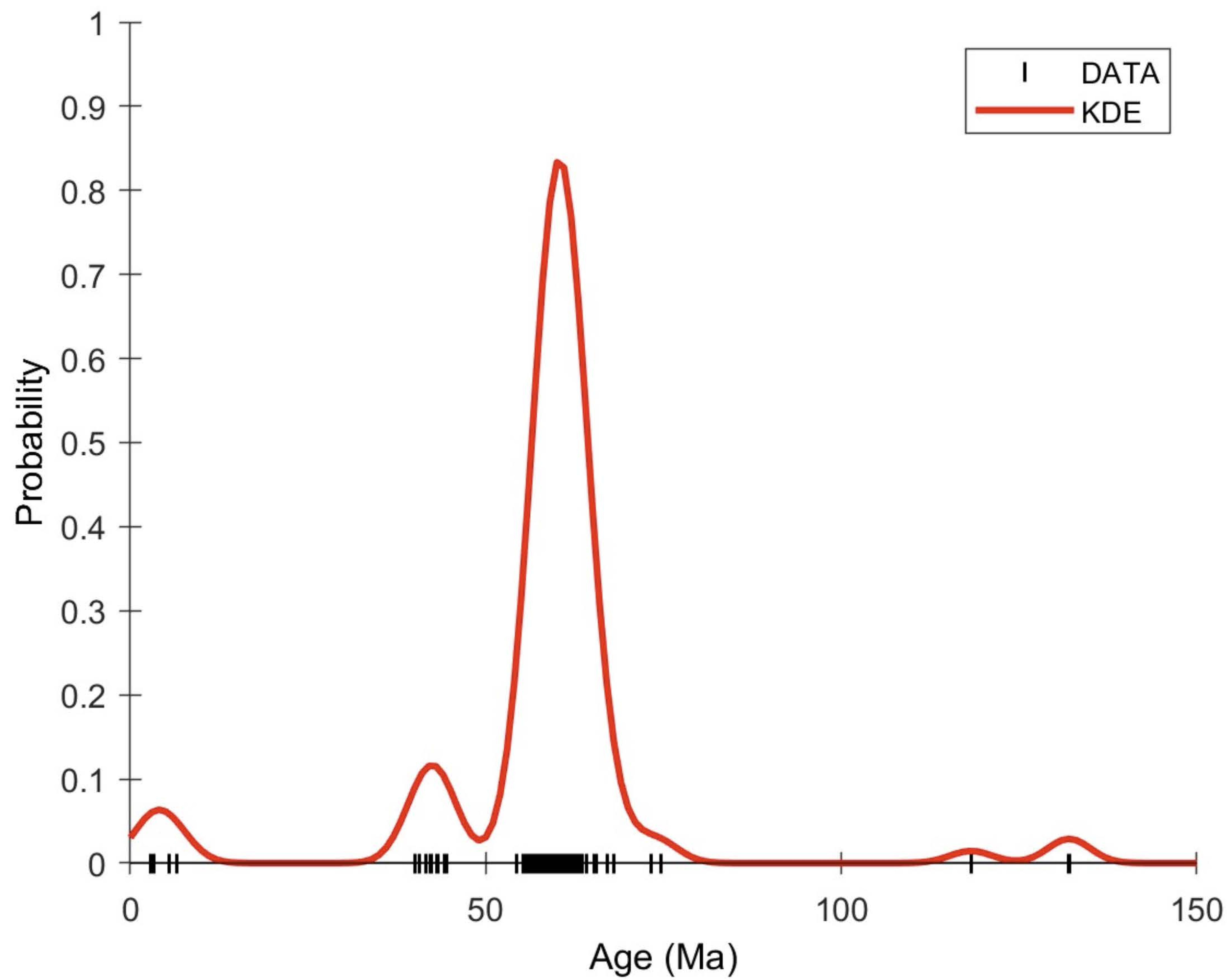
ages (MY)	**2se**	**...**
132.13	4.80	
41.53	2.98	
63.00	8.07	
43.15	2.65	
59.37	4.16	

$\vdots$

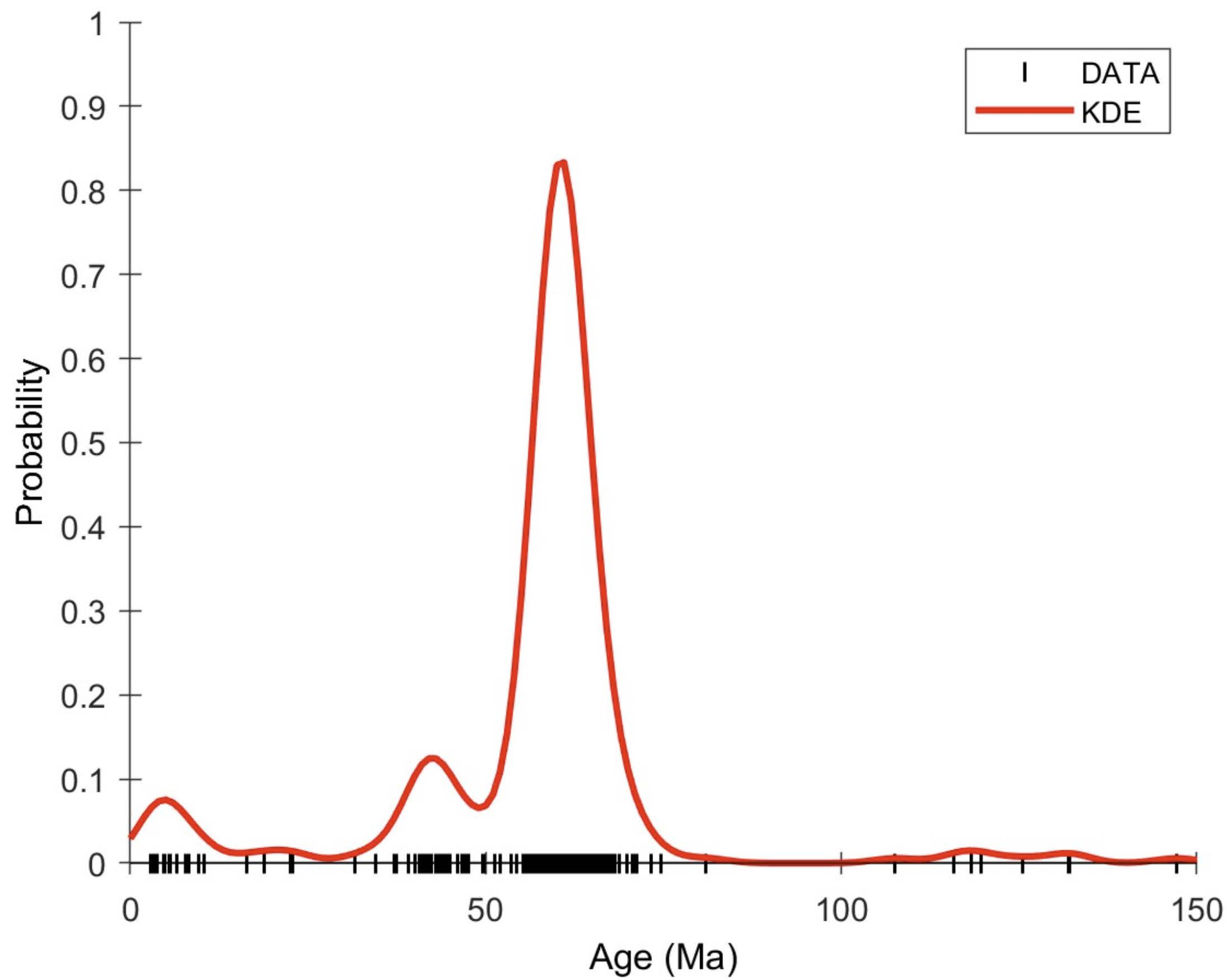
The Raw Data “Rug Plot”



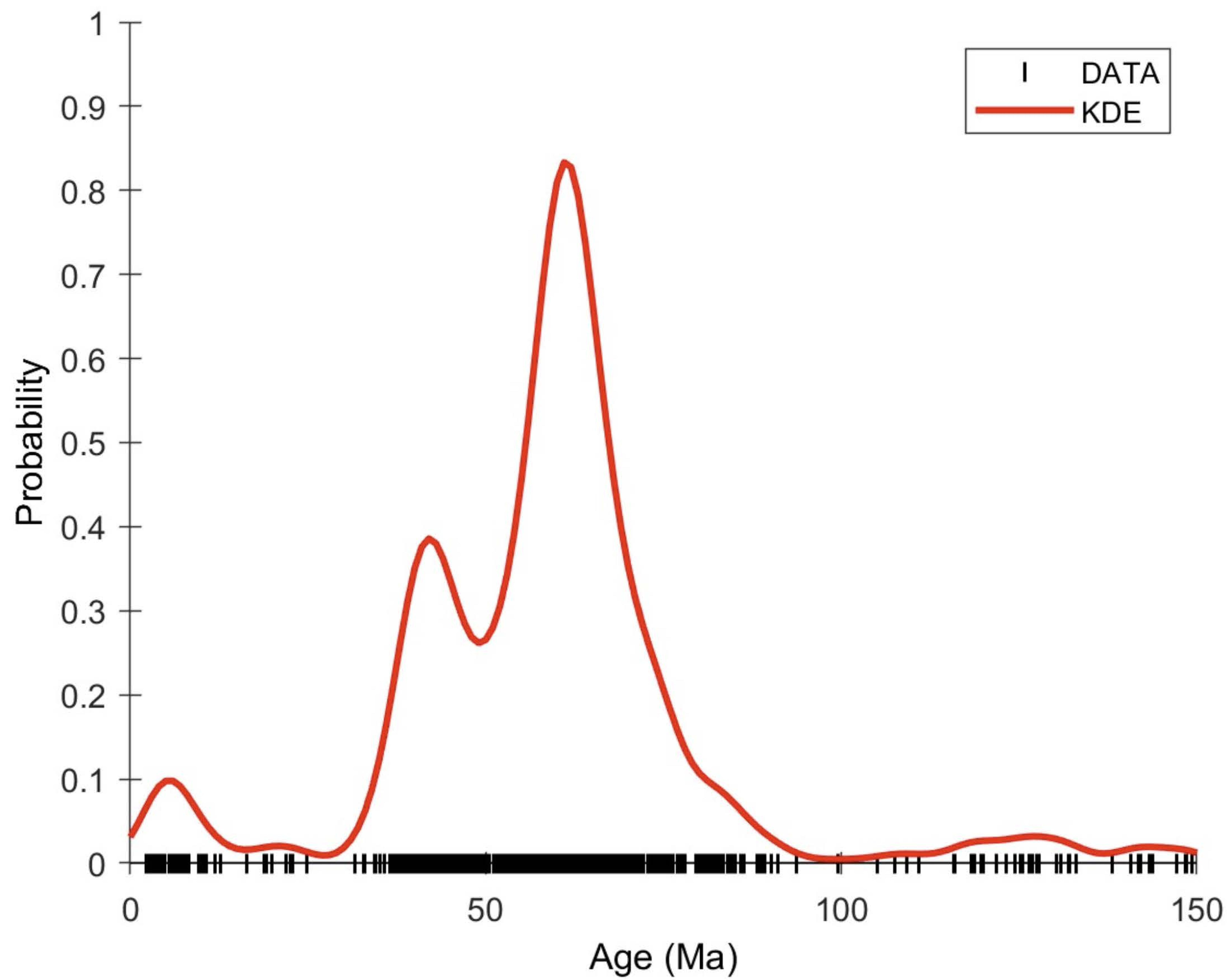
Transform Into a Smooth Density



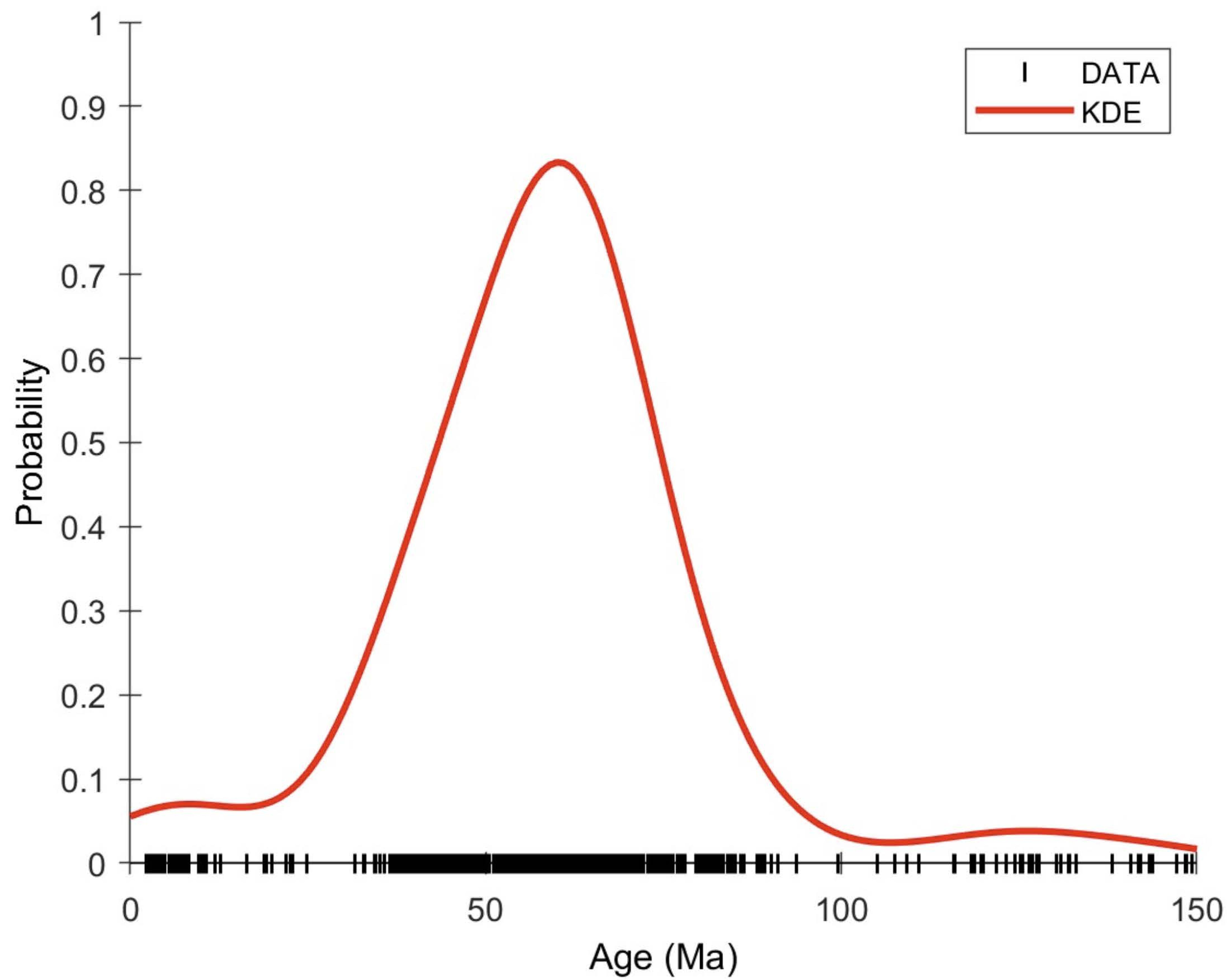
More Samples $\implies$ More Accurate



More Samples $\implies$ More Accurate

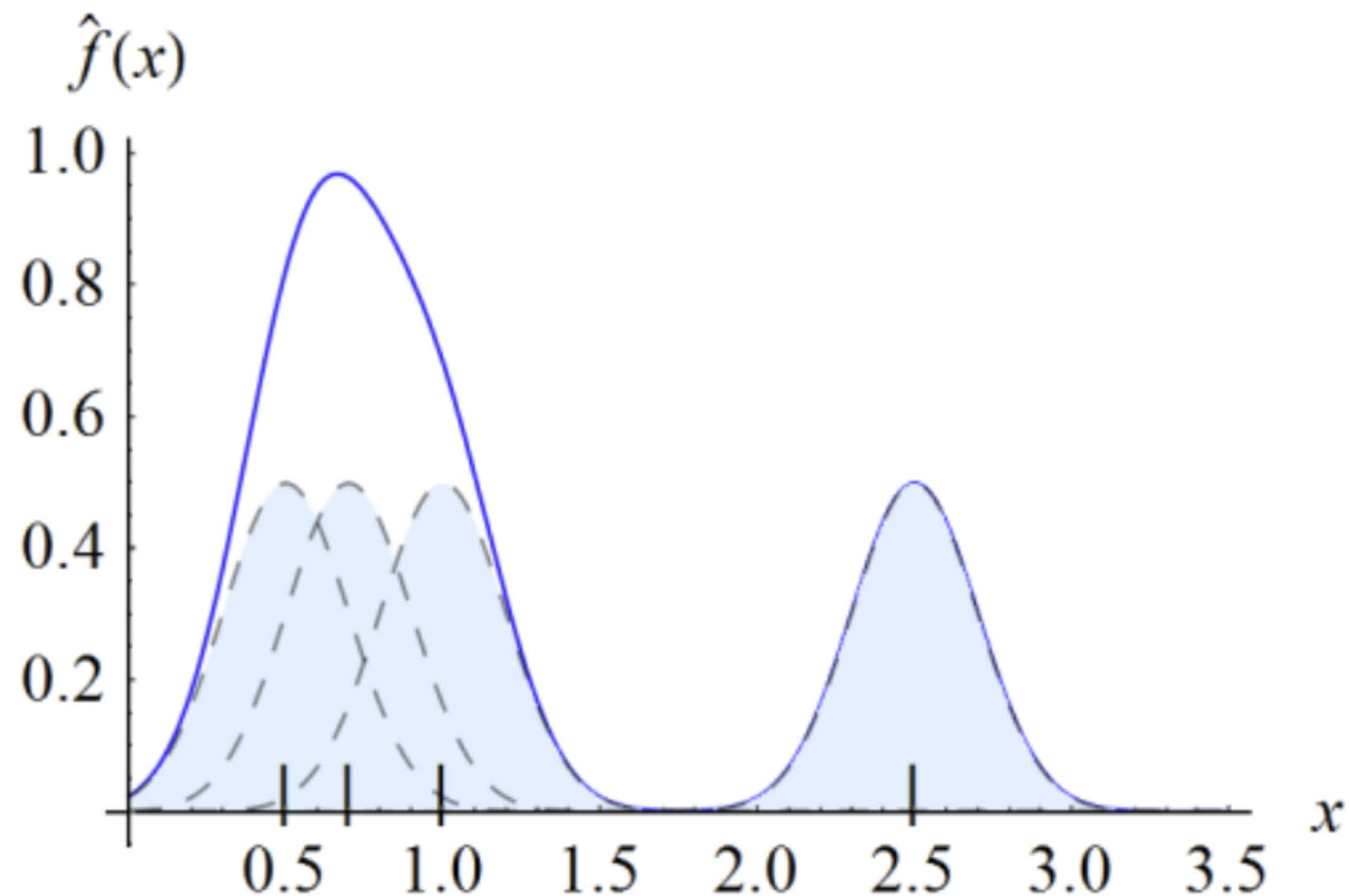


Larger bandwidth $\implies$ smoother



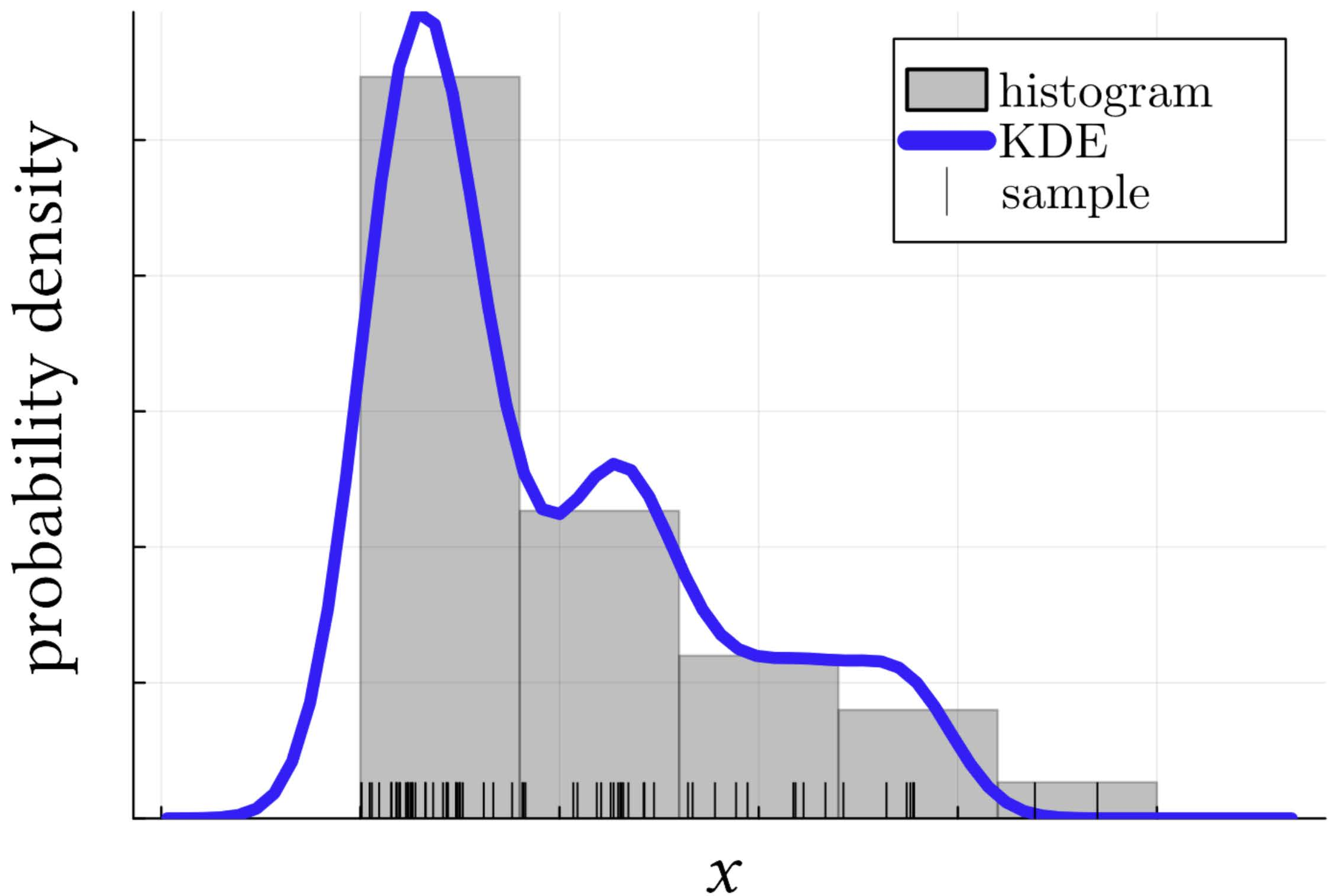
Kernel Density Estimation (KDE)

- Turns samples into distributions [Węglarczyk 2018]:



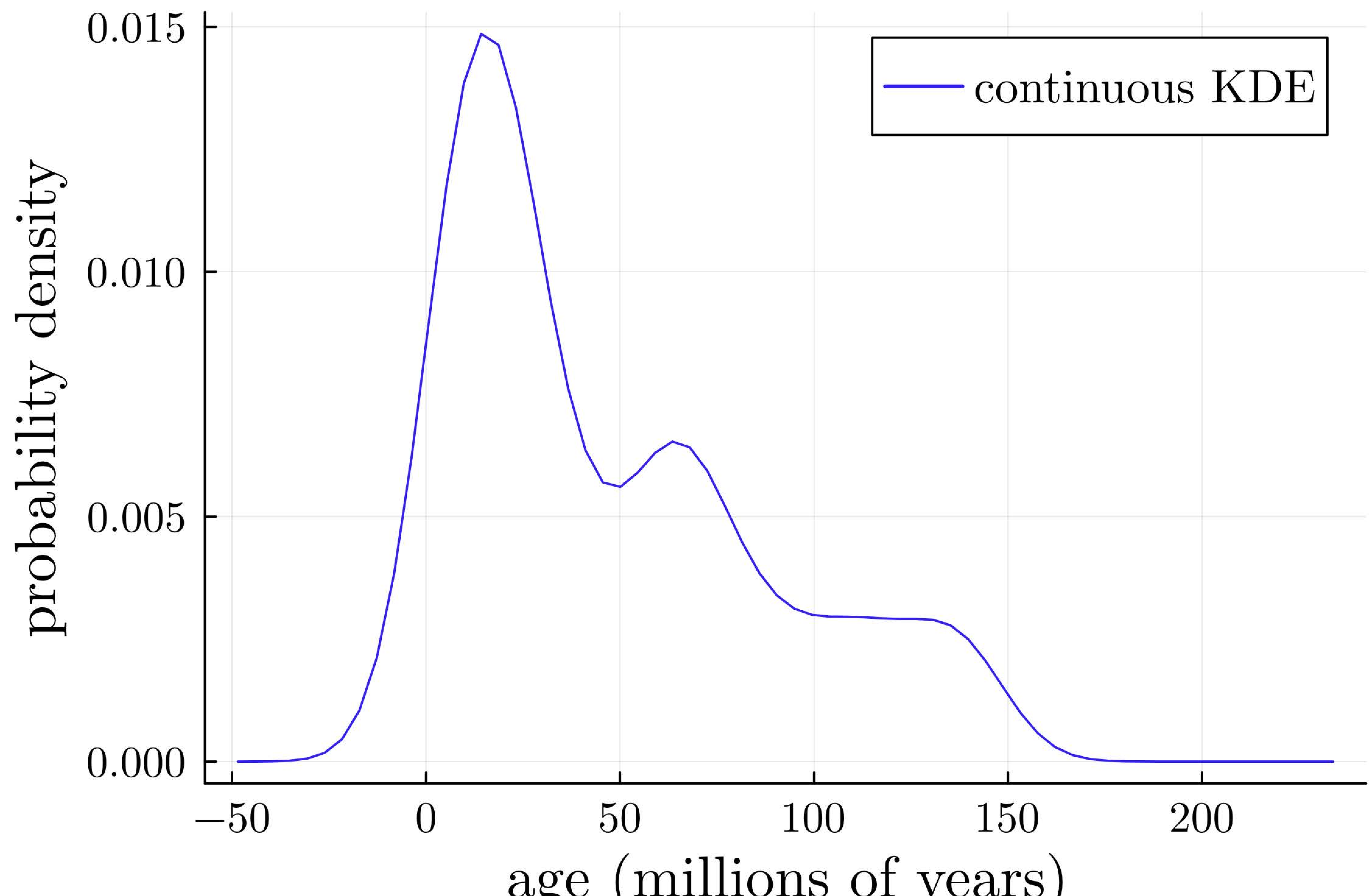
Kernel Density Estimation (KDE)

- Smooth version of a histogram



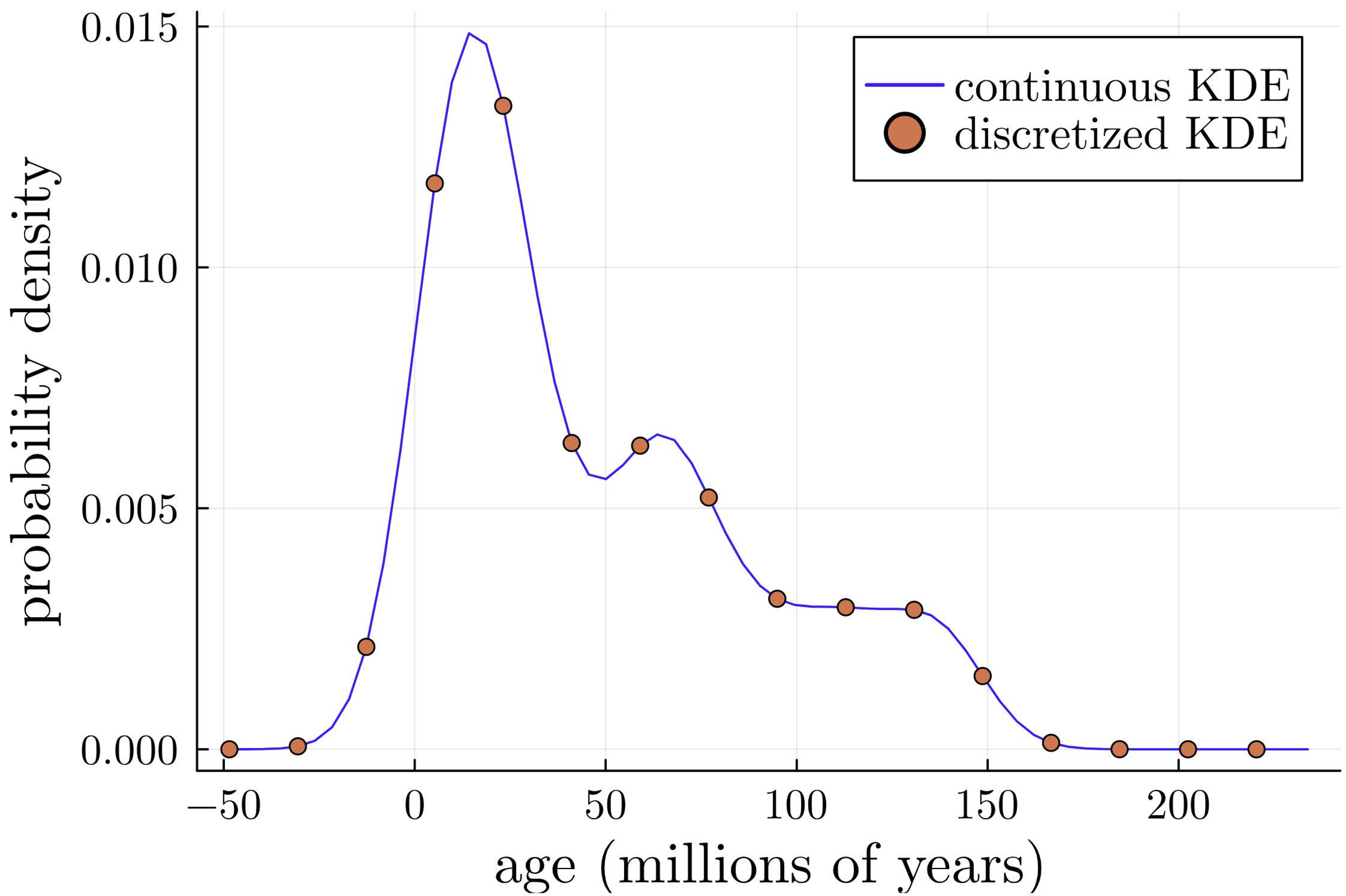
Discretizing the KDE

- Start with smooth KDE



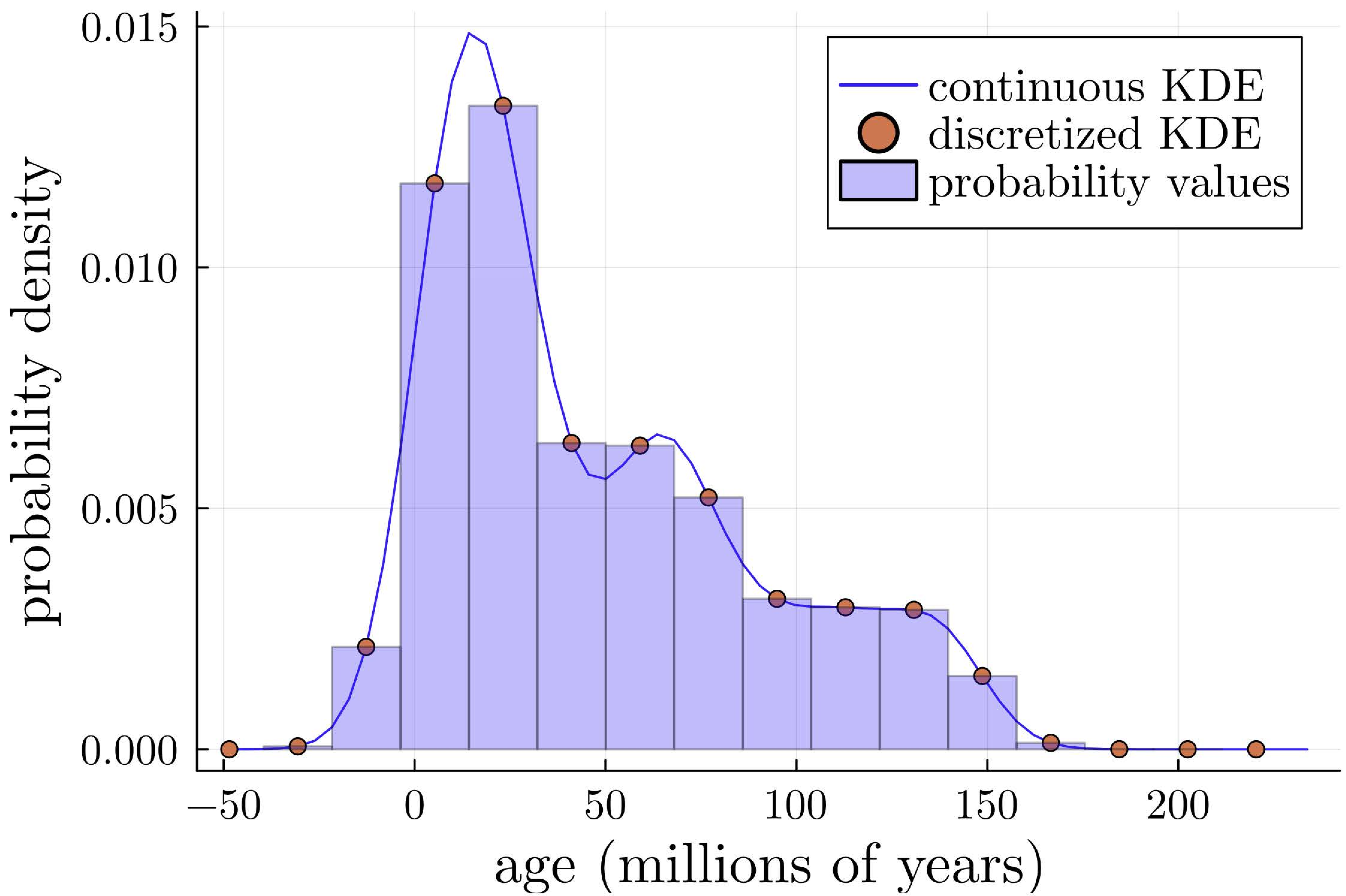
Discretizing the KDE

- Sample on a uniform grid



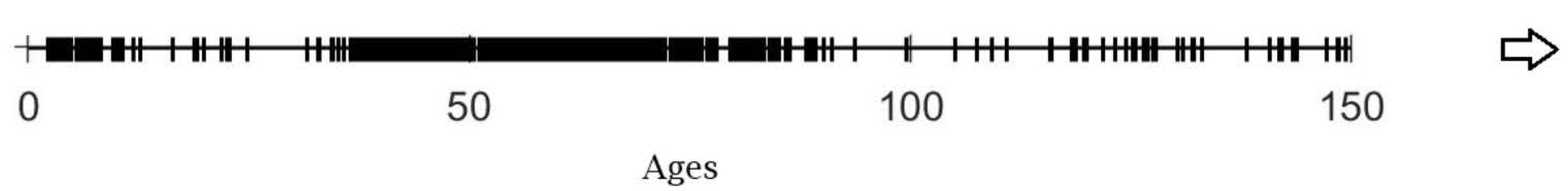
Discretizing the KDE

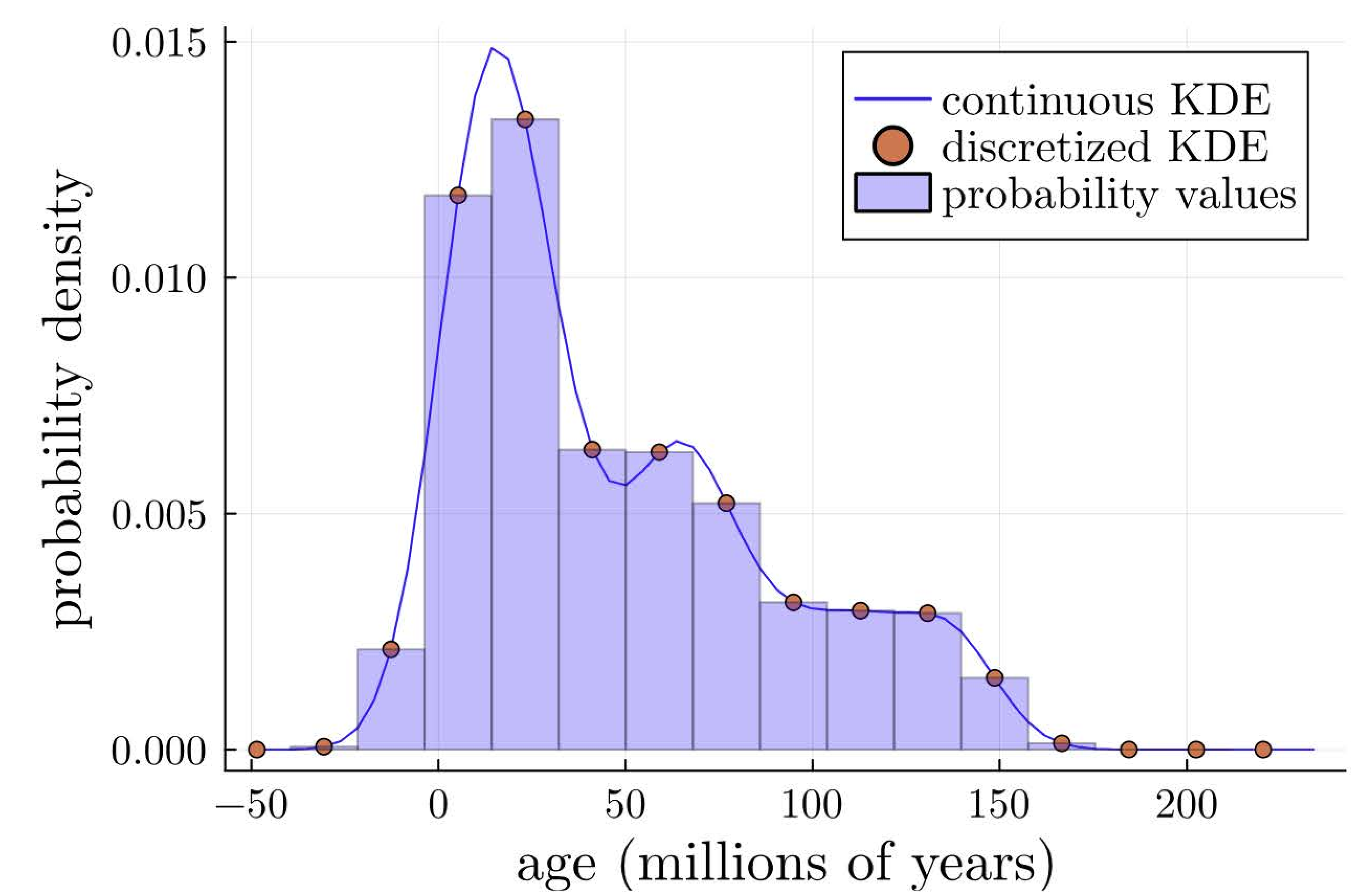
- Record the area of each box



Why do all this work?

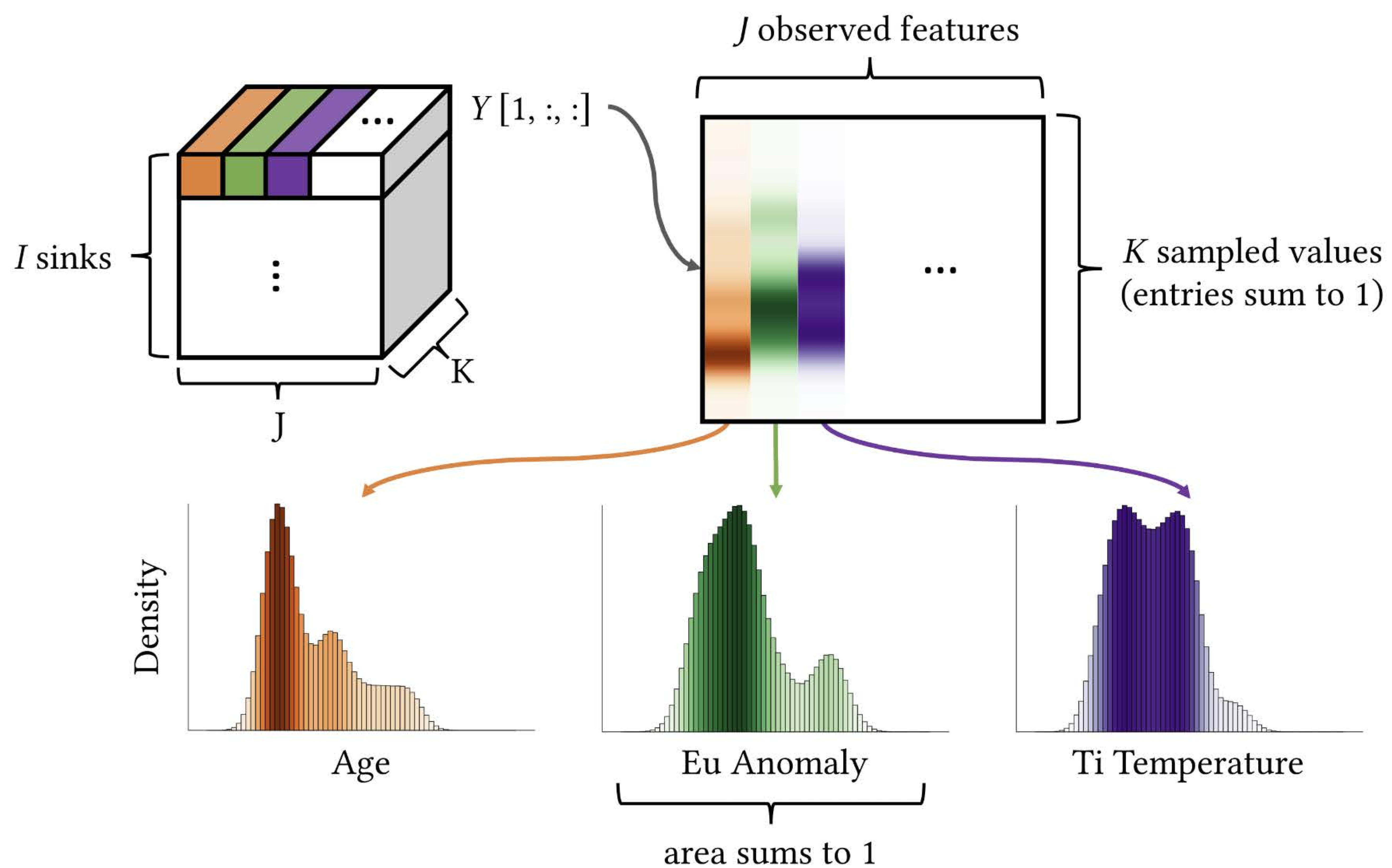
- This constructs density estimate for each feature in each sink
- Why use KDEs over histogram?
  - The size and smoothness of each KDE is the same
  - # of samples can be different across sinks and features





Data Tensor Y

- Lay KDEs along each fibre
  - i is the sink (different locations)
  - j is the feature (different elements)



Framing the demixing problem

Now we have $\mathbf{Y}$! Onto finding $\mathbf{A}$ and $\mathbf{B}$…

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

Framing the demixing problem

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

The system of equations becomes the equation $\mathbf{Y} = \mathbf{A}\mathbf{B}$

$$
\begin{bmatrix} \leftarrow & \mathbf{Y}_1 & \rightarrow \\ \leftarrow & \mathbf{Y}_2 & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{Y}_I & \rightarrow \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1R} \\ a_{21} & a_{22} & \dots & a_{2R} \\ \vdots & & & \vdots \\ a_{I1} & a_{I2} & \dots & a_{IR} \end{bmatrix} \begin{bmatrix} \leftarrow & \mathbf{B}_1 & \rightarrow \\ \leftarrow & \mathbf{B}_2 & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{B}_R & \rightarrow \end{bmatrix}
$$

Finding A and B with Tucker-1

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

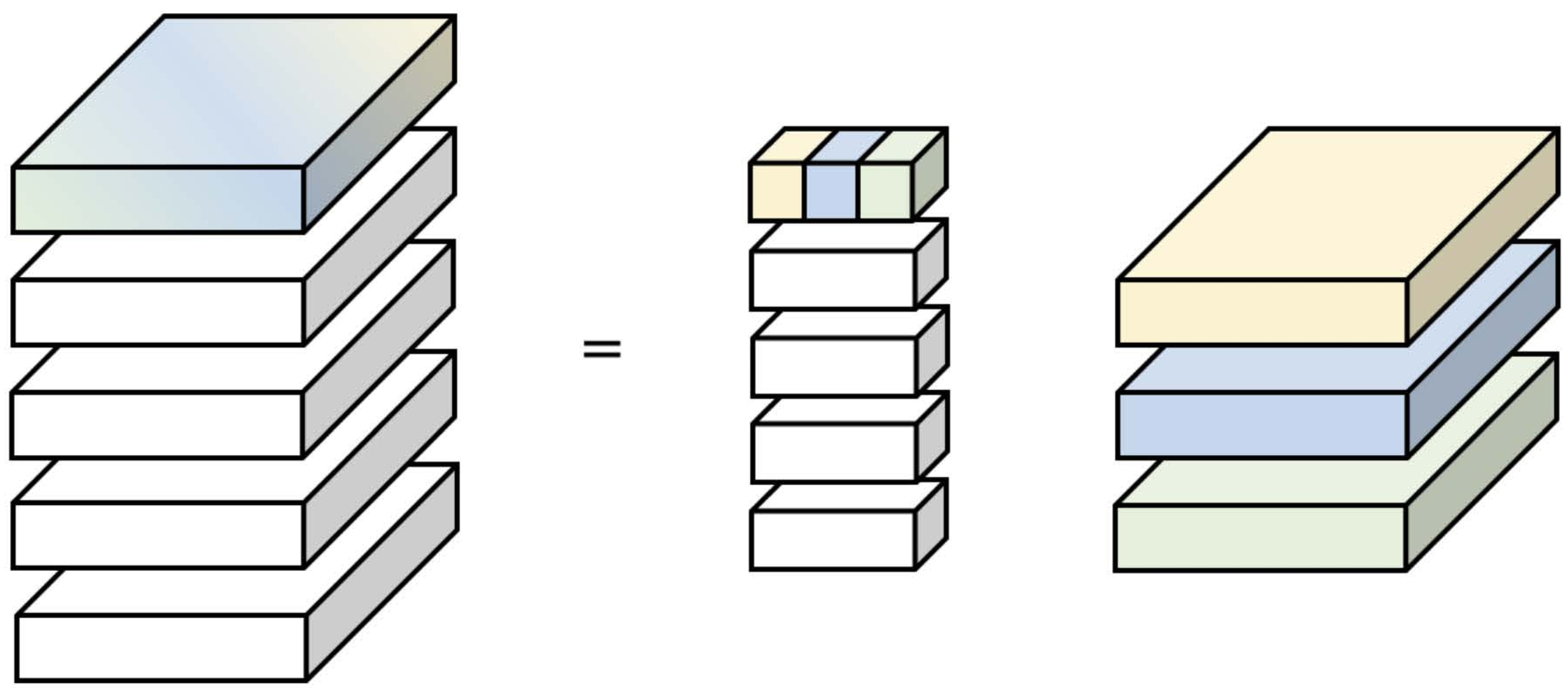


Figure 1: Example Tucker-1 decomposition for a 3rd-order tensor.

BlockTensorFactorization.jl

Least-Squares Optimization

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

$$\min_{\mathbf{A},\mathbf{B}} \ell(\mathbf{A}, \mathbf{B}) := \frac{1}{2}\|\mathbf{B} \times_1 \mathbf{A} - \mathbf{Y}\|_F^2 \quad \text{s.t} \quad \mathbf{A} \text{ in } \mathcal{C}_{\mathbf{A}},\ \mathbf{B} \text{ in } \mathcal{C}_{\mathbf{B}}.$$

Basic use:

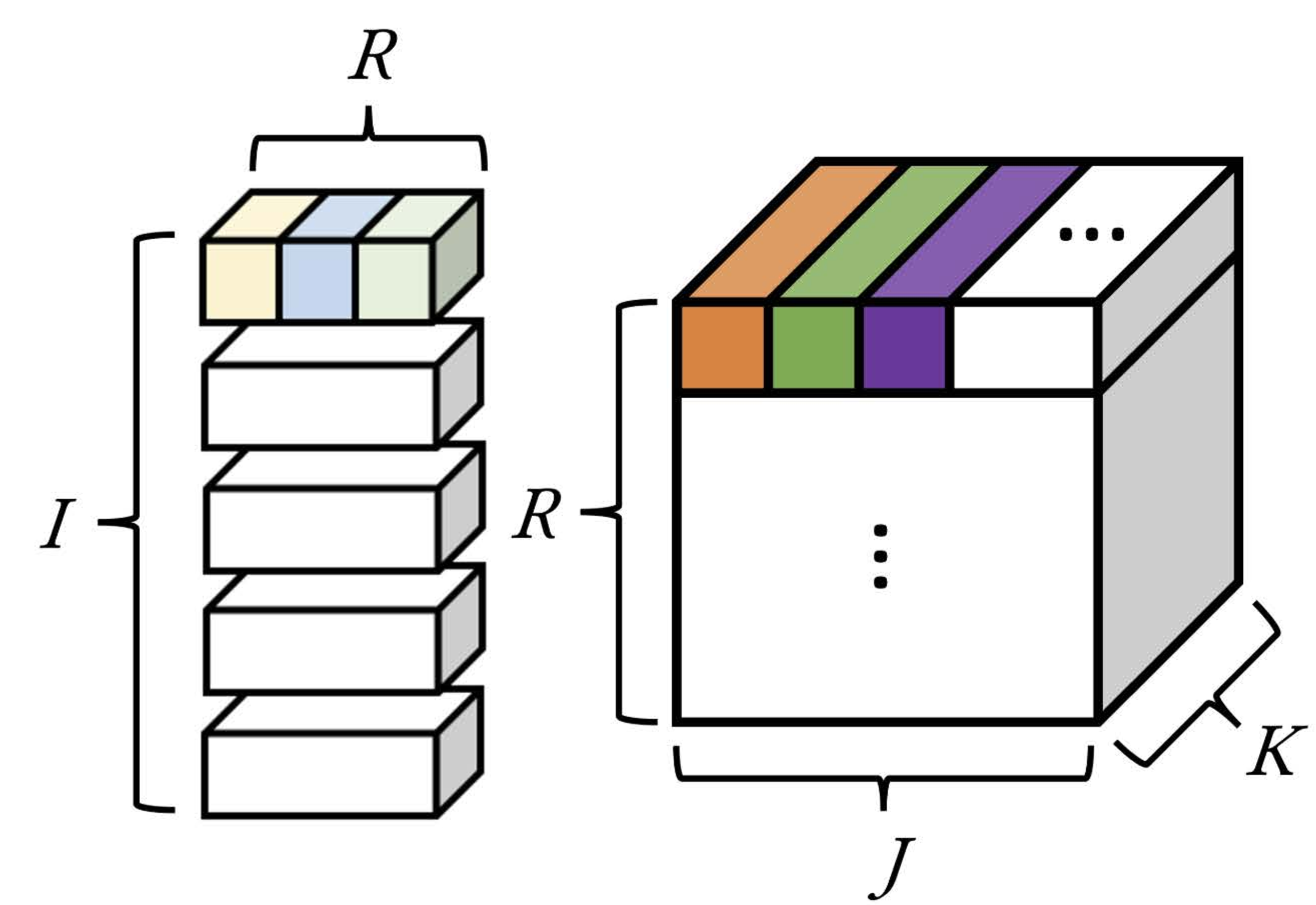
```
options = (rank=3, model=Tucker1, constraints=[B_constraint, A_constraint])

decomposition, stats, kwargs = factorize(Y; options...)

(B_out, A_out) = factors(decomposition)
```

Algorithm: Constraints

$$\mathcal{C}_{\mathbf{A}} = \Delta^I_R = \left\{ \mathbf{A} \in \mathbb{R}_+^{I\times R} \middle| \sum_{r=1}^R \mathbf{A}[i,r] = 1, \text{ for all } i \right\}$$

$$\mathcal{C}_{\mathbf{B}} = \Delta^{RJ}_K = \left\{ \mathbf{A} \in \mathbb{R}_+^{R\times J\times K} \middle| \sum_{k=1}^K \mathbf{B}[r,j,k] = 1, \text{ for all } r,j \right\}$$

- Rows of $\mathbf{A}$ and fibres of $\mathbf{B}$ need to be nonnegative and sum to one
- Ensures their product is a probability distribution




```
constraint_A = simplex_rows!
constraint_B = simplex_12slices!
```

Algorithm: Alternating Descent

- Start with a guess for $\mathbf{A}^0$ and $\mathbf{B}^0$. Then for $t = 1, 2, \ldots$

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

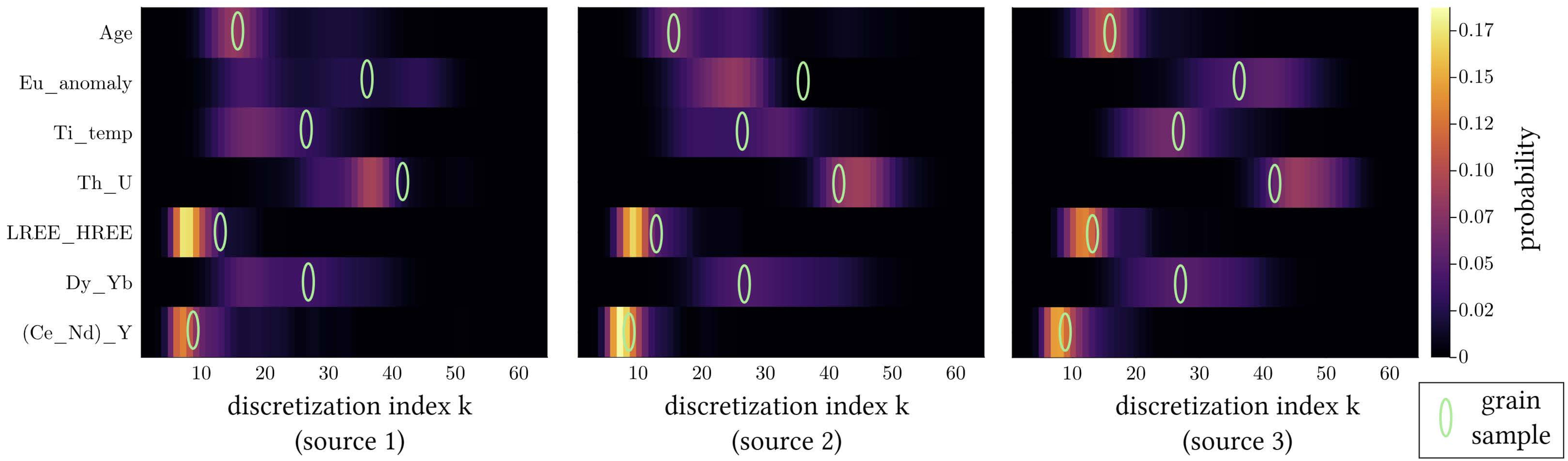
- Converges to a *block-wise* minimum and stationary point

$$\ell(\mathbf{A}^*, \mathbf{B}^*) \leq \min_{\mathbf{A} \in \mathcal{C}_{\mathbf{A}}, \mathbf{B} \in \mathcal{C}_{\mathbf{B}}} \{\ell(\mathbf{A}^*, \mathbf{B}), \ell(\mathbf{A}, \mathbf{B}^*)\}$$

The Bells and Whistles

Labelling Grains

- Categorize each grain **g** according to its most likely source
- Learned distribution sources $\mathbf{B}_r$:



Labelling Grains

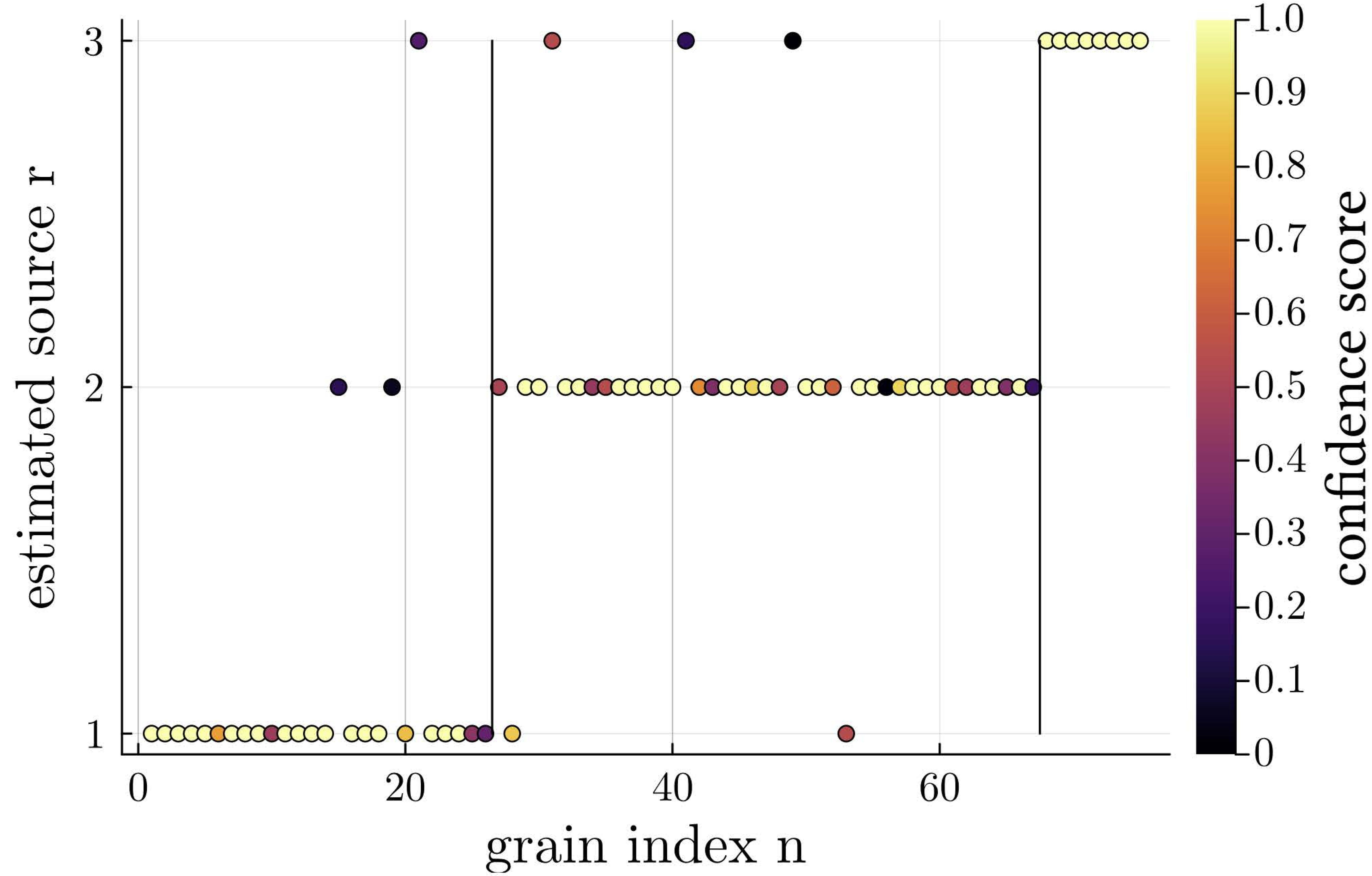
- Approximate the probability that a grain came from source r

$$p_r = \mathbb{P}(\mathbf{g} \in \mathcal{V}_{\mathbf{g}} \mid \mathbf{g} \sim \mathbf{B}_r) \approx \prod_{j=1}^{J} \mathbf{B}[r, j, \hat{k}_j]$$

- $\mathcal{V}_{\mathbf{g}} = \left[x_{1\hat{k}_1}, x_{1(\hat{k}_1+1)}\right] \times \cdots \times \left[x_{J\hat{k}_J}, x_{J(\hat{k}_J+1)}\right]$ is the box that contains the measured grain
- Label based on the most likely probability $\hat{r} = \text{argmax}_r p_r$

Labelling Grains

- Estimated source for many grains:
- Grains are ordered by their true source in this example



Label Confidence Score

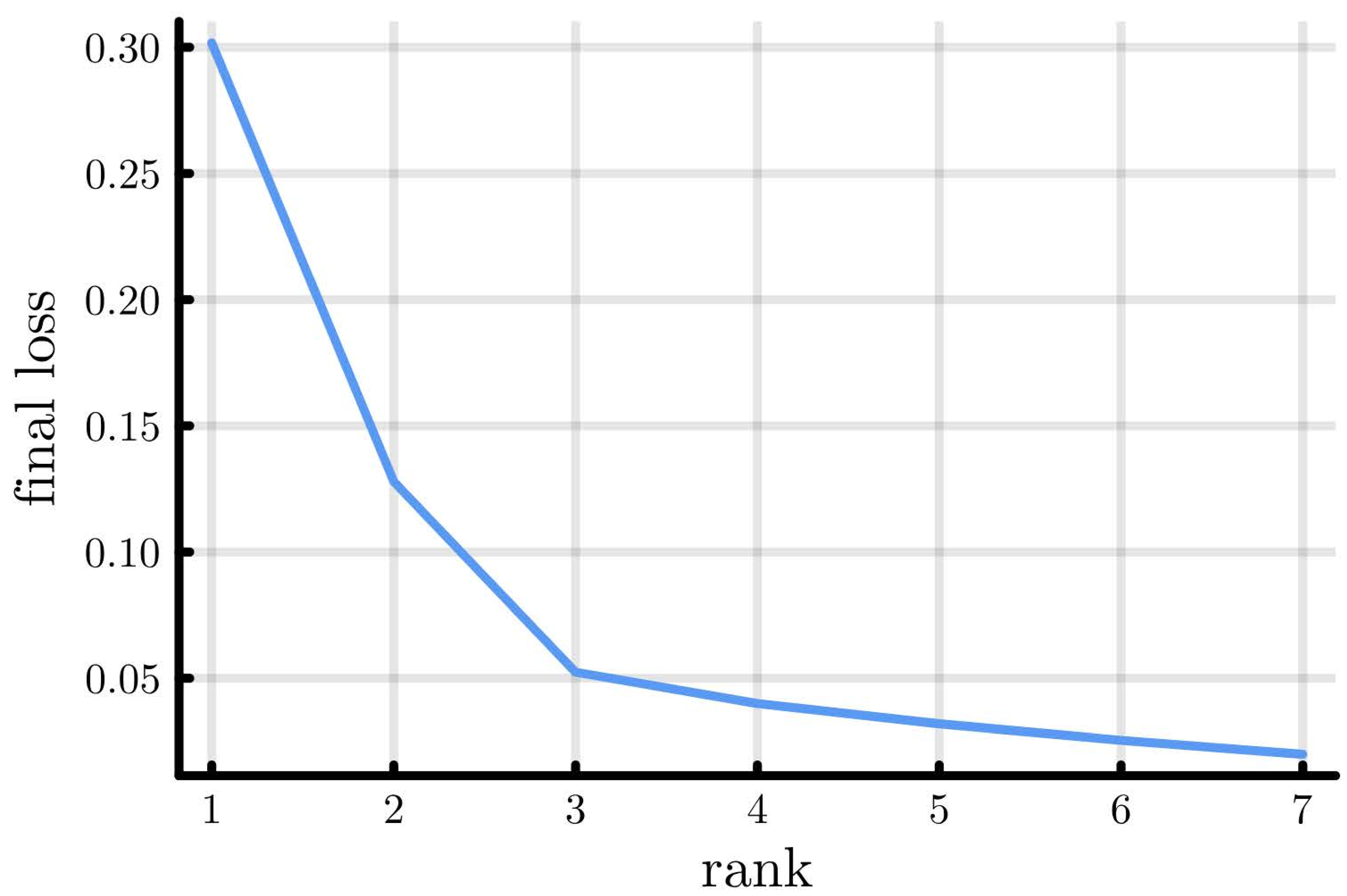
- Give a confidence score for the grain labels using the log-ratio of the top two estimated probabilities

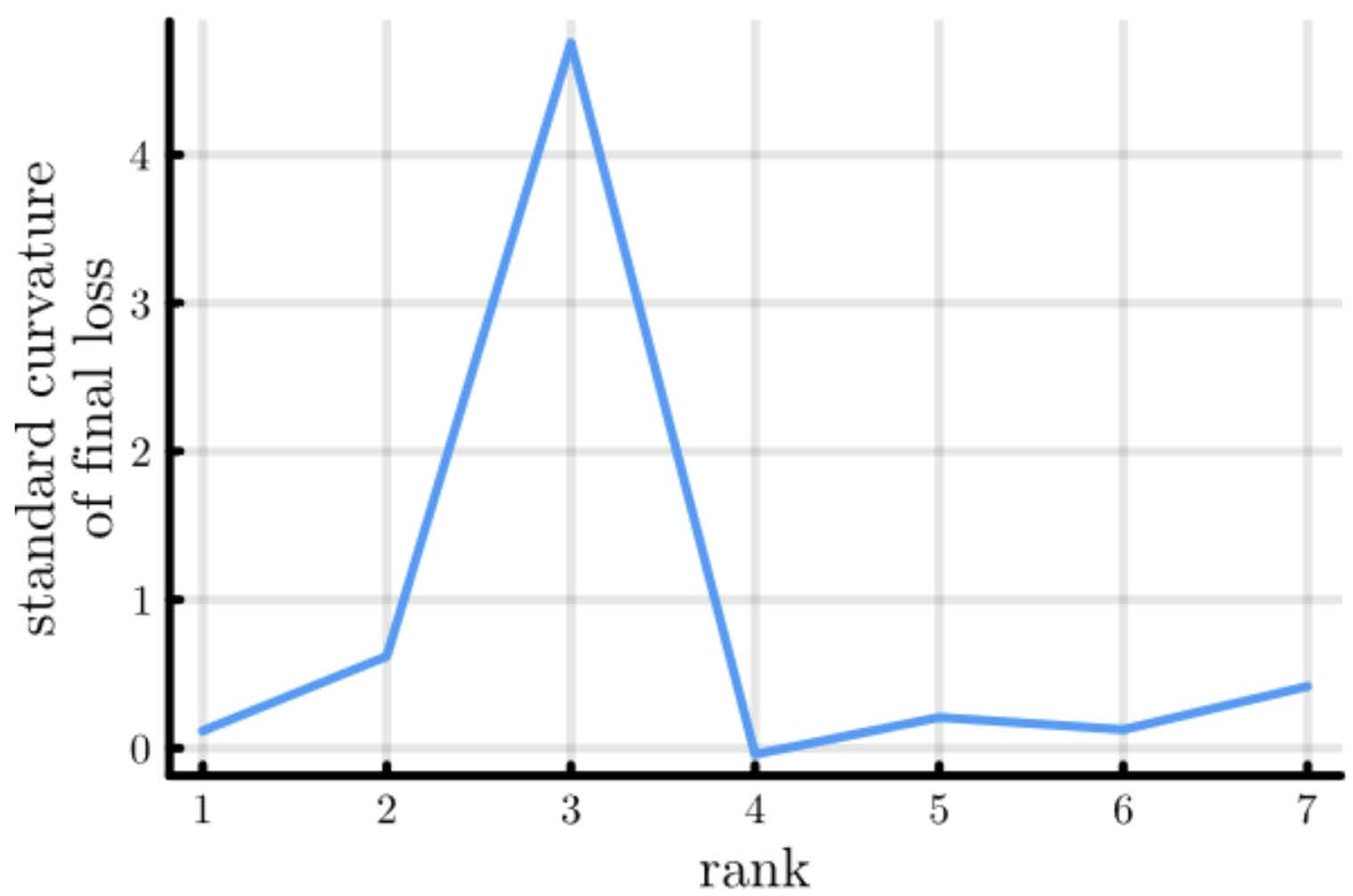
$$\text{confidence score} = \log_{10}\left(\frac{p_{(1)}}{p_{(2)}}\right)$$

- score ≥ 1 means it's at least 10 times as likely the grain came from source $\hat{r}$ than any other source
- score ≈ 0 means there's a similar probability the grain came from a difference source

Estimating Rank

- Usually we need to know R in advance
- Can estimate it based on an accuracy and simplicity trade-off





Estimating Rank

- Let $f(r) = \|\mathbf{X}_r^* - \mathbf{Y}\|_F / \|\mathbf{Y}\|_F$ be final relative error with a rank r factorization
- Pick the r at the maximum curvature [Satopaa *et al.* 2011]

$$\hat{R} = \arg\max_r \kappa_f(r) := \frac{f''(r)}{(1 + (f'(r))^2)^{3/2}}$$

```
options = (model=Tucker1, constraints=[B_constraint, A_constraint])

# Will try to factorize at many ranks and pick the "best" one
decomposition, stats, kwargs = rank_detect_factorize(Y; options...)

estimated_rank = kwargs[:rank]
```

Multi-Scaled Decomposition

- Use `continuous_dims` to optimize over multiple scales
- Often faster, and leads to smoother results

```
factorize(Y; continuous_dims=[3], options...)
# Third dimension K is continuous in our model
```

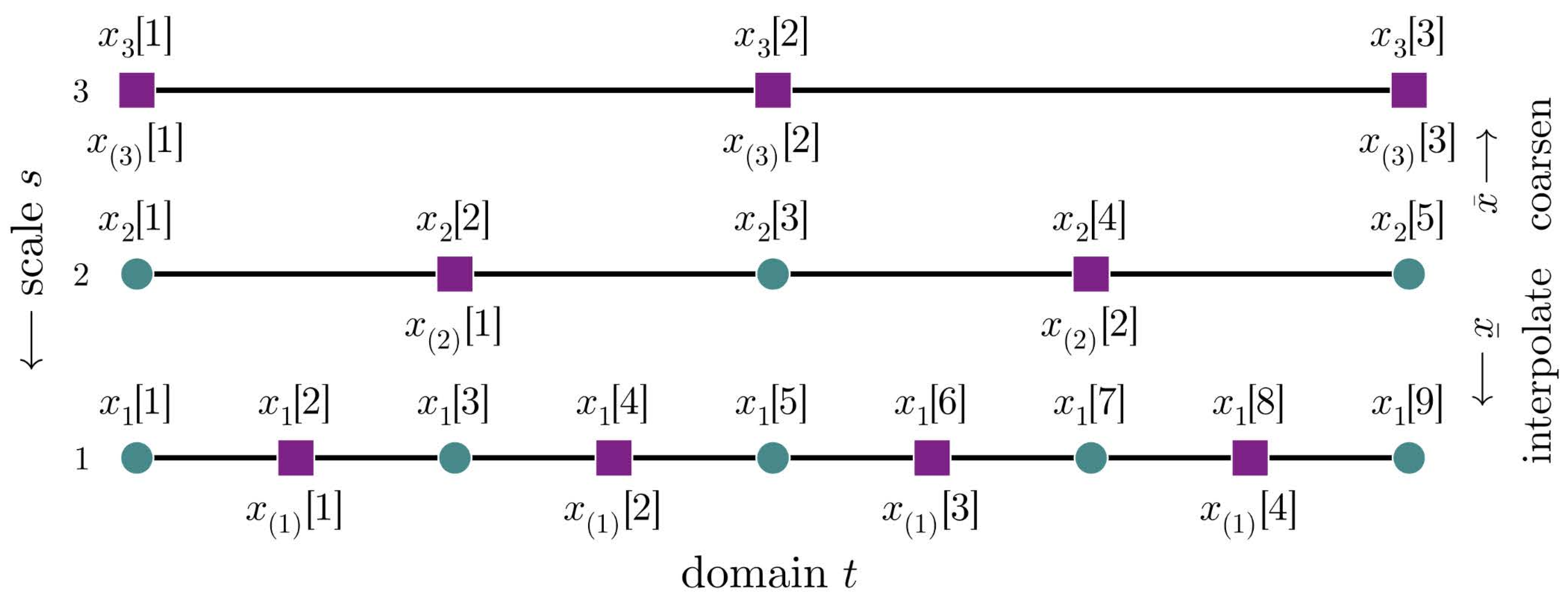


Figure 2: Rather than discretizing the mixtures $\mathbf{Y}_i$ on a fine grid from the start, optimize over a cheaper, coarse discretization with fewer points and gradually refine.

References

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

[2] S. Węglarczyk, "Kernel density estimation and its application," *ITM Web of Conferences*, vol. 23, 2018.

[3] T. G. Kolda and B. W. Bader, "Tensor Decompositions and Applications," *SIAM Rev.*, vol. 51, no. 3, Aug. 2009.

[4] V. Satopaa, J. Albrecht, D. Irwin, and B. Raghavan, "Finding a "Kneedle" in a Haystack: Detecting Knee Points in System Behavior," in *2011 31st International Conference on Distributed Computing Systems Workshops*, Jun. 2011.

Try our code here!

https://github.com/MPF-Optimization-Laboratory/BlockTensorFactorization.jl

Extra Slides

Many more extra features…

- Momentum and second-order acceleration like Block-lipschitz constants (Hessian approximations)
- Rank detection
- Other factorizations like full Tucker, CP-Decomposition, custom
- Other constraints like p-norms, interval, linear, custom
- Other block update order: fully random, partial random

Rescaling trick

- Relax simplex constraints to $\mathbf{A} \geq 0, \mathbf{B} \geq 0$
- and $\frac{1}{J}\sum_{jk} \mathbf{B}[r, j, k] = 1$ for all r. Old constraints were $\sum_k \mathbf{B}[r, j, k] = 1$ for all r, j

Updates now look like:

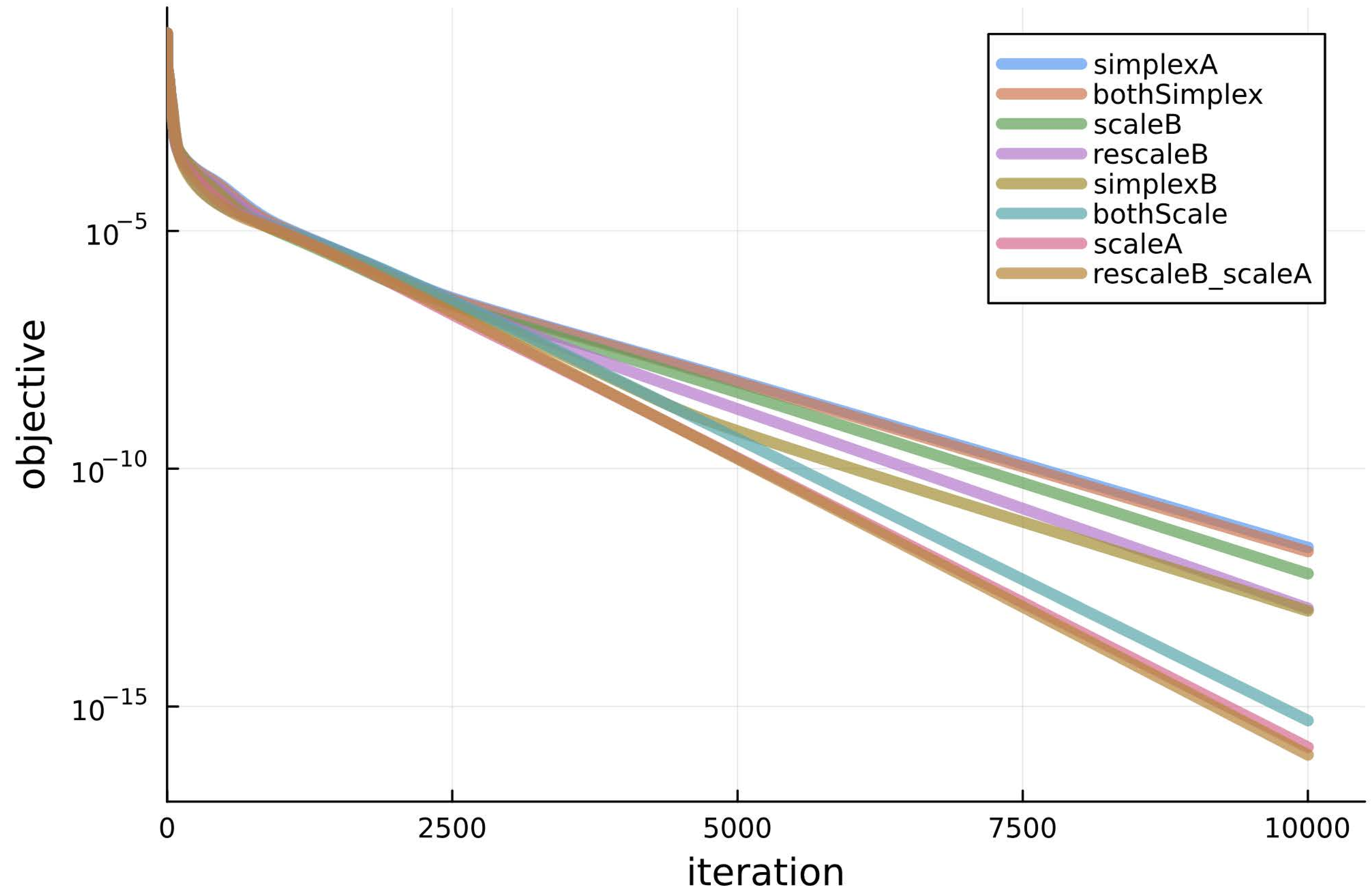
- $\mathbf{A}^{t+1/2} = (\mathbf{A}^t - \frac{1}{L_{\mathbf{A}}}\nabla_{\mathbf{A}}\ell(\mathbf{A}^t, \mathbf{B}^t))_+$
- $\mathbf{B}^{t+1/2} = (\mathbf{B}^t - \frac{1}{L_{\mathbf{B}}}\nabla_{\mathbf{B}}\ell(\mathbf{A}^{t+1/2}, \mathbf{B}^t))_+$
- $\mathbf{B}^{t+1} = \mathbf{C}^{-1}\mathbf{B}^{t+1/2}$ and $\mathbf{A}^{t+1} = \mathbf{A}^{t+1/2}\mathbf{C}$
- where $\mathbf{C}[r, r] = \frac{1}{J}\sum_{jk} \mathbf{B}[r, j, k]$

Examples of Constraints

```
l1scale_average12slices! = ScaledNormalization(l1norm;
    whats_normalized=each1slice, scale=size2)

nonnegative! = Entrywise(ReLU, isnonnegative)

scaleB_rescaleA! = ConstraintUpdate(
    0, # the zeroth factor
    l1scale_average12slices! ∘ nonnegative!;
    whats_rescaled = x -> eachcol(factor(x, 1))
)

constraint_A = nonnegative!
constraint_B = scaleB_rescaleA!
```

Rescaling vs simplex projection

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



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

$f_{n,r}^{t}(a) = \frac{1}{2}\left\|[\mathbf{B}; \mathbf{A}_1^{t+1}, \ldots, \mathbf{A}_{n-1}^{t+1}, A_{n,r}^{t}(a), \mathbf{A}_{n+1}^{t}, \ldots, \mathbf{A}_N^{t}] - \mathbf{Y}\right\|_F^2$

and

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

Block-Lipschitz Constant

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

where

$$f_n^t(a) = \frac{1}{2}\left\|[\mathbf{B}; \mathbf{A}_1^{t+1}, \dots, \mathbf{A}_{n-1}^{t+1}, \mathbf{A}_n^t, \mathbf{A}_{n+1}^t, \dots, \mathbf{A}_N^t] - \mathbf{Y}\right\|_F^2$$

and

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

Momentum

Before a gradient step, we move $\mathbf{A}$ further in the direction of travel

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

where the amount of momentum is determined by

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

Methods for estimating rank

- Can be one of…
- `:spline`, default and often accurate
- `:finite_differences`, faster but less accurate
- `:circles`, fastest and smallest memory footprint, but more sensitive to results from factorize
- `:breakpoints`, picks the rank R that fits the model

$$f(r) = a + b(\min(r, R) - R) + c(\max(r, R) - R)$$

to the final errors