BlockTensorDecomposition.jl

Optimization Framework for Constrained Tensor Factorization

30 Jul 2025 | SIAM/CAIMS AN25 | Montréal, QC

Nicholas Richardson[1]

[1]Department of Mathematics

Michael P. Friedlander[12]

[2]Department of Computer Science

THE UNIVERSITY OF BRITISH COLUMBIA

siam 2025 | The Third Joint SIAM/CAIMS Annual Meetings

INSTITUTE OF APPLIED MATHEMATICS
University of British Columbia

Overview

- half new developments in tensor factorization / half advertisement
- propose a signal demixing framework and implementation in Julia using constrained tensor factorization
- use this tool to separate real signal mixtures in applications like geology, biology, and music
- easily extendable to a broader class of tensor decompositions

Setting: Multiple Unlabeled Mixtures

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

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

Setting: Multiple Unlabeled Mixtures

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

- Mixtures can be multivariable functions $\mathbf{y}_i : \mathbb{R}^N \to \mathbb{R}$
- or directly measured vectors/matrices/tensors $\mathbf{y}_i[j_1, \ldots, j_N]$

Either way, package data into a tensor Y by sampling the mixtures $\{\mathbf{y}_i\}$ or stacking the observations:

$$Y[i, j_1, \ldots, j_N] = \mathbf{y}_i(x_1[j_1], \ldots, x_N[j_N]) = \mathbf{y}_i[j_1, \ldots, j_N].$$

Model: Tucker-1 Tensor Factorization

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

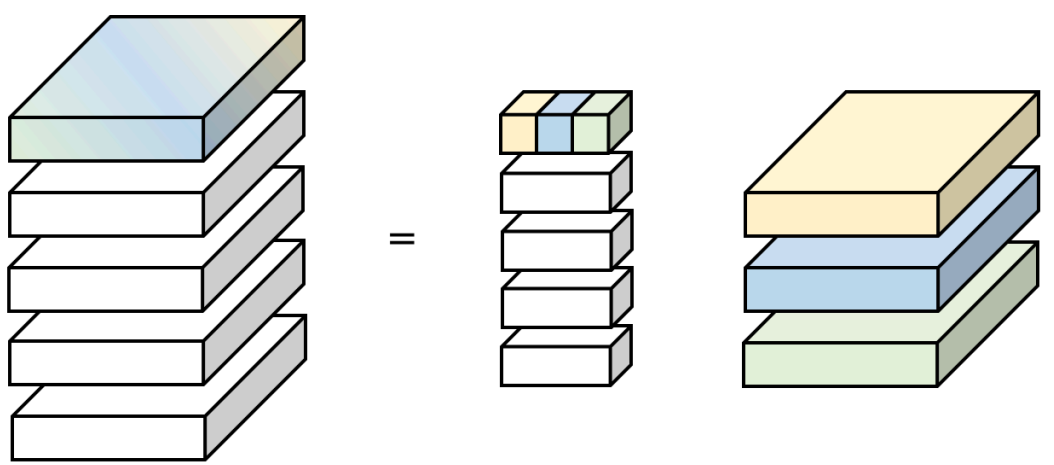


Figure 1: Example Tucker-1 decomposition for a 3rd-order tensor.

Application: Sediment Analysis

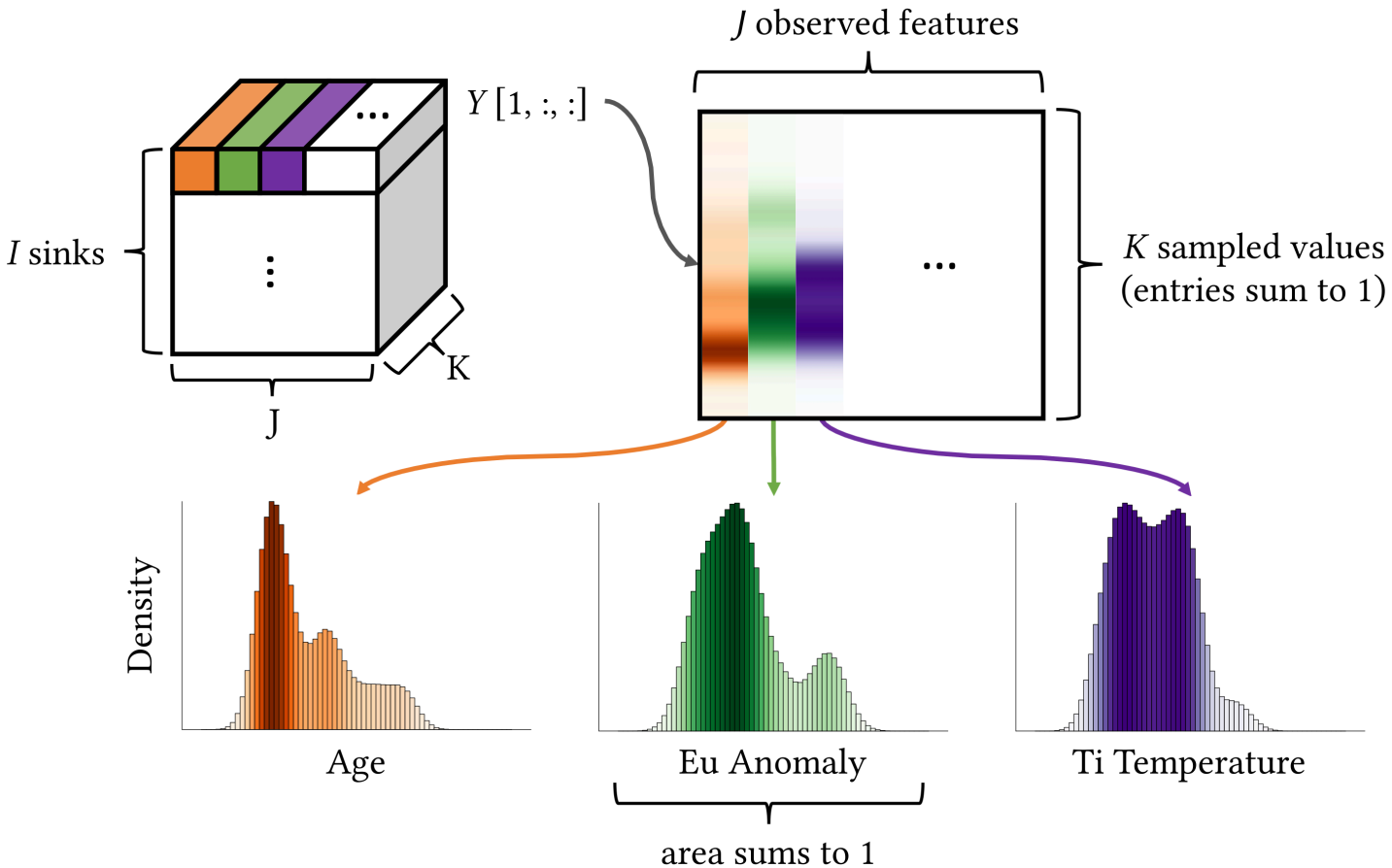


Figure 2: Input data tensor Y. Each depth fibre $Y[i, j, :]$ is a discretized probability density for a different geological feature. Decomposed source distributions can be used to classify grains. See our paper *Tracing Sedimentary Origins in Multivariate Geochronology via Constrained Tensor Factorization* [2].

Application: Spatial Transcriptomics

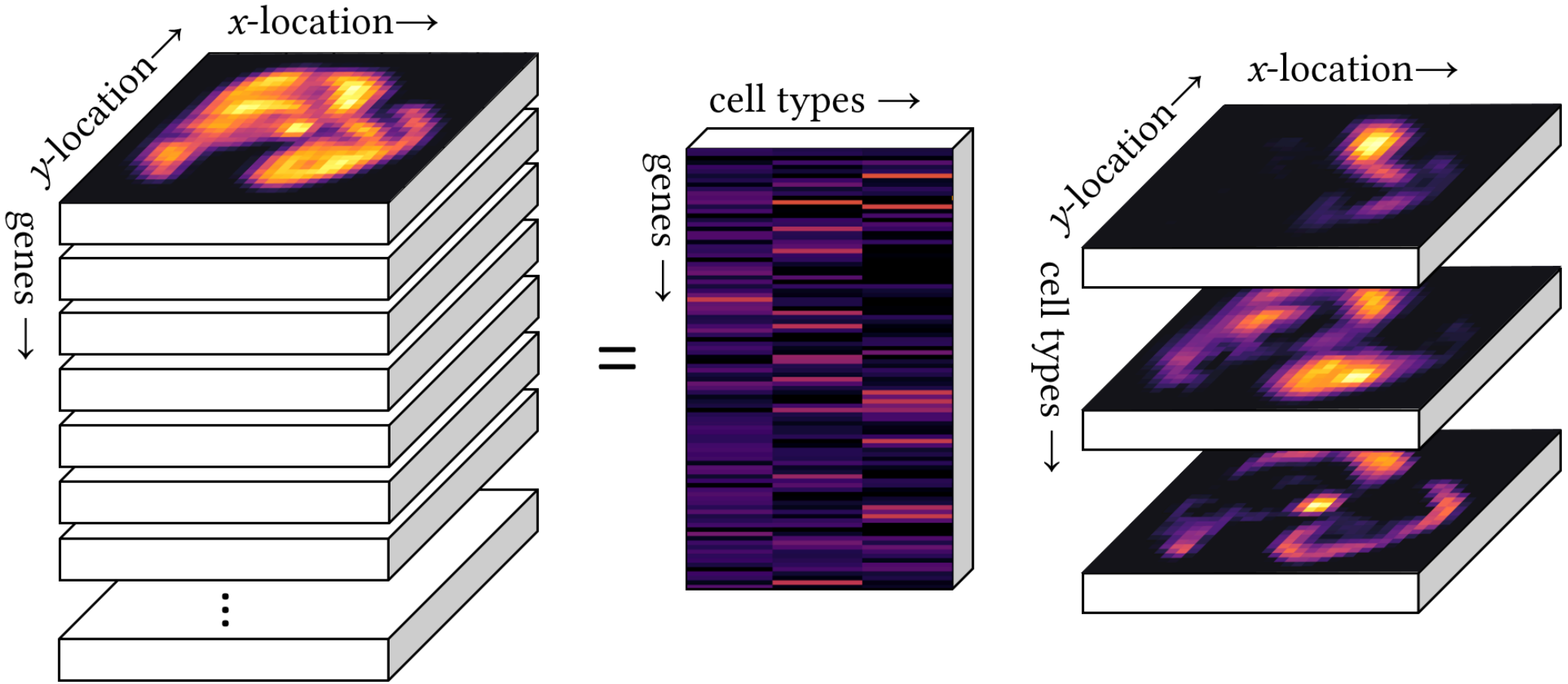


Figure 3: Spatial transcriptomics factorization model. Spatial distribution of many genes can be decomposed into few cell types. We uncover the gene expression and spatial distribution of these cell types, and can label distinct regions accordingly.

Application: Musical Instrument Separation

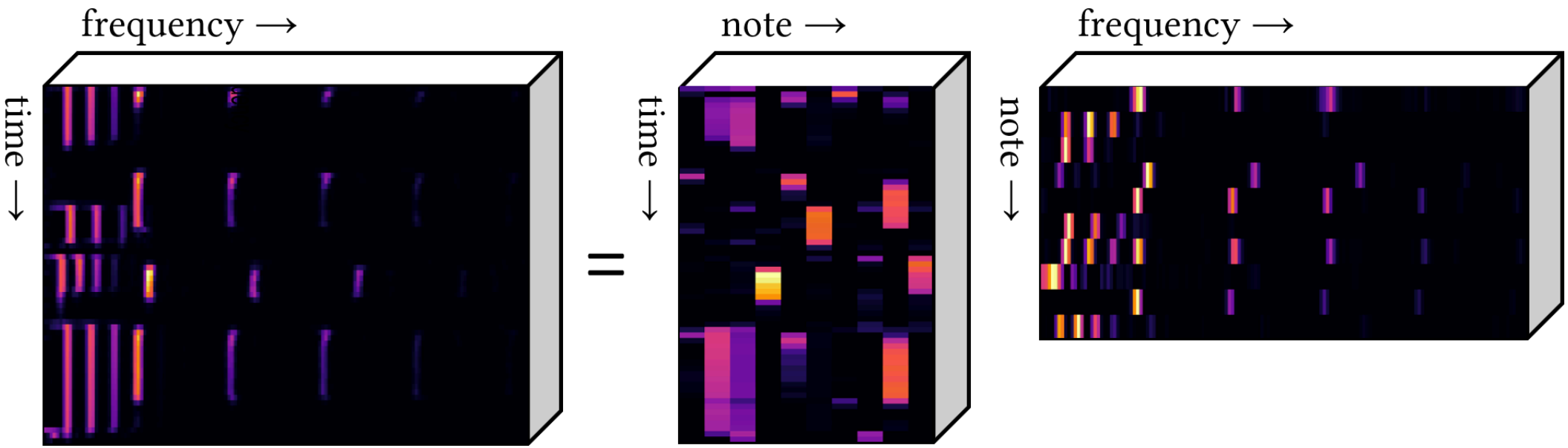


Figure 4: Audio source separation model. The short-time Fourier transform of a mixture can be separated into harmonically distinct notes. These can be grouped by their spectral similarity to recover instrument sources.

BlockTensorDecomposition.jl

Least-Squares Optimization

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

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

Basic use:

```
A = abs.(randn(5, 3))
B = abs.(randn(3, 6, 7))
Y = B ×₁ A

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

Algorithm

Block Projected Gradient Descent

Cyclically update factors with descent updates [3]:

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

```
julia> kwargs[:update]
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))
)
```

Guarantees

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

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

Set `tolerance=0.01` to ensure `RelativeError` is less that 1 %.

```
julia> display(stats_data)
 Row │ Iteration  ObjectiveValue  GradientNNCone  RelativeError
     │ Int64      Float64          Float64         Float64
─────┼──────────────────────────────────────────────────────────
   1 │         0     0.618283         1.78212        0.505625
   2 │         1     0.25426          1.23183        0.324246
   3 │         2     0.104661         0.605968       0.20803
   ⋮ │      ⋮           ⋮               ⋮              ⋮
  45 │        44     0.000297298      0.00446732     0.0110874
  46 │        45     0.000250057      0.00410569     0.0101684
  47 │        46     0.000210449      0.003773       0.00932844
```

The Bells and Whistles

Multi-Scaled Decomposition

Use `continuous_dims` to optimize over multiple scales for faster convergence.

```
julia> @time decomposition, stats_data, kwargs = factorize(Y; options...);
11.828295 seconds (214.97 k allocations: 15.197 GiB, 32.42% gc time)
```

```
julia> @time decomposition, stats_data, kwargs = multiscale_factorize(Y;
      continuous_dims=[2, 3, 4], options...);
2.335374 seconds (201.33 k allocations: 2.905 GiB, 25.26% gc time)
```

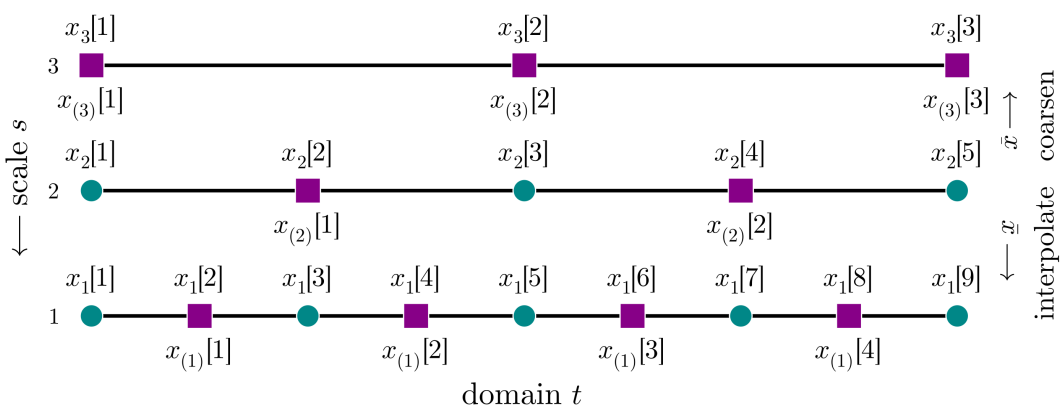


Figure 5: Rather than discretizing the mixtures $\{\mathbf{y}_i\}$ on a fine grid from the start, optimize over a cheaper, coarse discretization with fewer points and gradually refine.

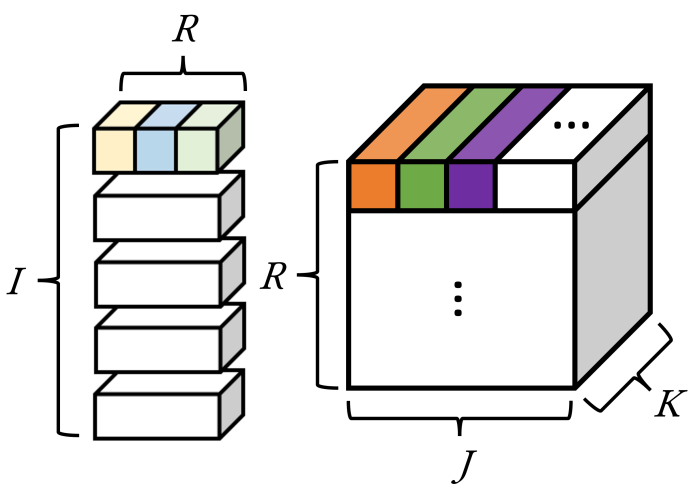
Rescaling trick

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

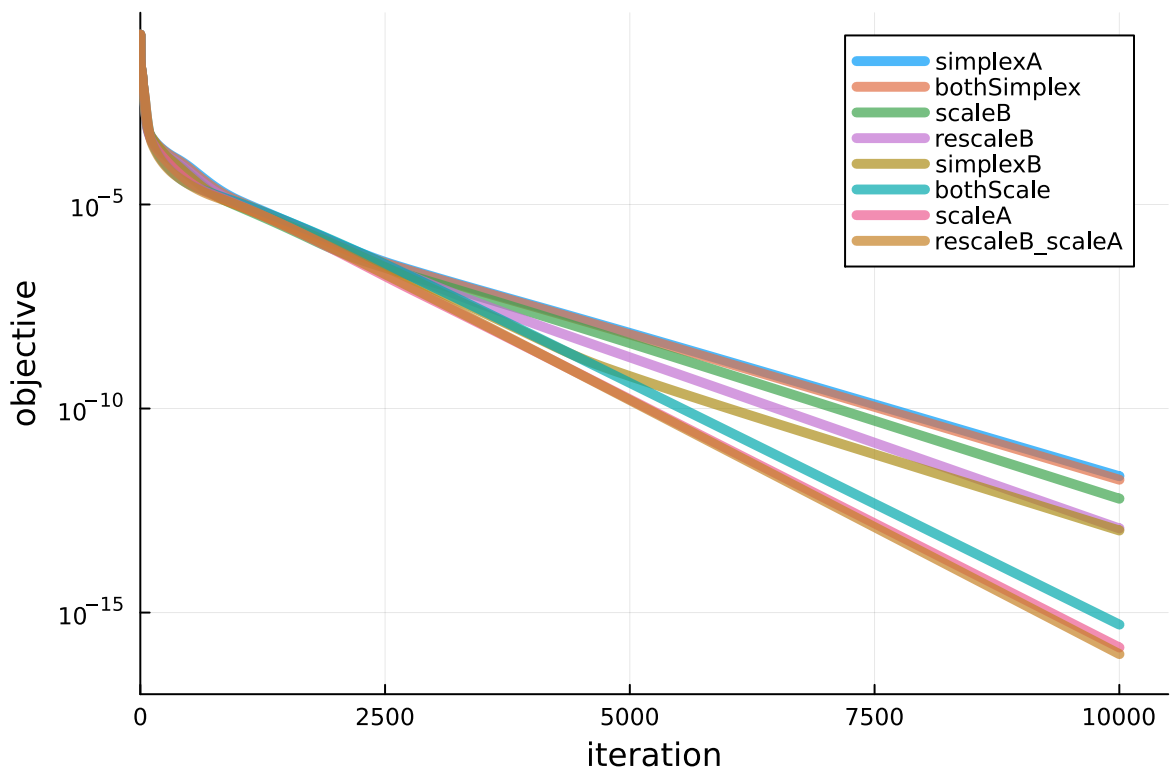
- Relax simplex constraints to $\mathbf{A} \geq 0, \mathbf{B} \geq 0$
- and $\frac{1}{J} \sum_{jk} B_{rjk} = 1$ for all r (vs $\sum_{k} B_{rjk} = 1$ for all r, j)

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: dist $(0, \partial(\ell + \delta_{\geq 0})(\mathbf{A}, \mathbf{B}))$ at every iteration for different constraint methods



Many more…

- 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
- Coming soon: Other objectives like 1-norm, entropy, KL-divergence

References

[1] T. G. Kolda and B. W. Bader, “Tensor Decompositions and Applications,” *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, Aug. 2009, doi: 10.1137/07070111X.

[2] N. Graham, N. Richardson, M. P. Friedlander, and J. Saylor, “Tracing Sedimentary Origins in Multivariate Geochronology via Constrained Tensor Factorization,” *Mathematical Geosciences*, Feb. 2025, doi: 10.1007/s11004-024-10175-0.

[3] Y. Xu and W. Yin, “A Block Coordinate Descent Method for Regularized Multiconvex Optimization with Applications to Nonnegative Tensor Factorization and Completion,” *SIAM J. Imaging Sci.*, vol. 6, no. 3, pp. 1758–1789, Jan. 2013, doi: 10.1137/120887795.

[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, pp. 166–171. doi: 10.1109/ICDCSW.2011.20.

Please try out our code! Change branch to latest version to test full suite of features.
https://github.com/MPF-Optimization-Laboratory/MatrixTensorFactor.jl

Estimating Rank

- Usually we need to know R in advance
- For 1-parameter factorizations like Tucker-1 and CP-Decomposition…
- Let $f(r) = \|X_r^* - Y\|_F / \|Y\|_F$ be final relative error with a rank r factorization
- Pick the r at the maximum curvature [4]

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

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)