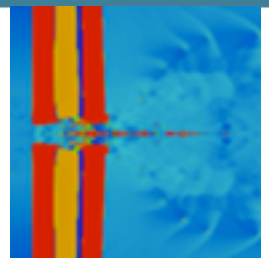
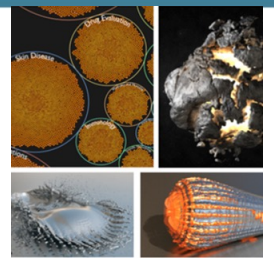
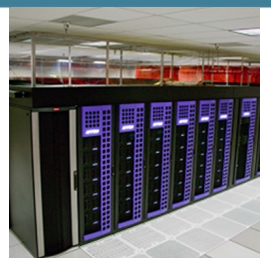
Sandia National Laboratories

Tensor Decomposition of Large-Scale Data with GenTen

Center for Computing Research







Eric Phipps, Scalable Algorithms

With contributions by
Saibal De, Danny Dunlavy, Hemanth Kolla, John Shadid (SNL),
Nick Johnson (BlueSpace.ai) and Tammy Kolda (MathSci.ai)

October 22, 2024

U.S. DEPARTMENT OF ENERGY | NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND2024-13683C

Tensor decompositions facilitate reduction and analysis of high-dimensional data

Terrain identification in hyperspectral image data (lat × lon × channel)

(image credit: NASA)

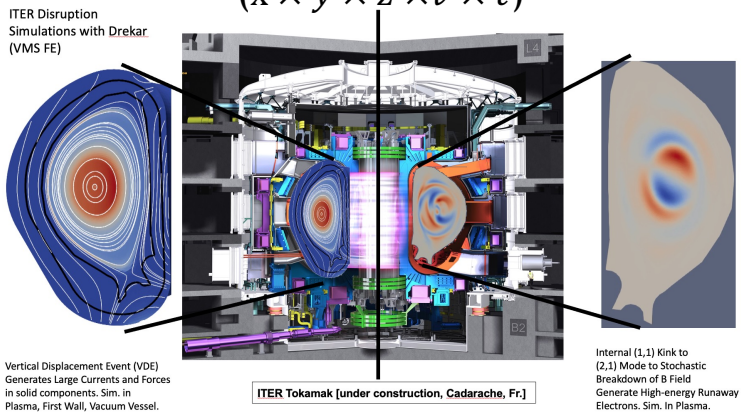
Intrusion detection in cyber data (src IP × src port × dst IP × dst port × time)

(image credit: pixabay)

Neural activity (neuron × time × trial)

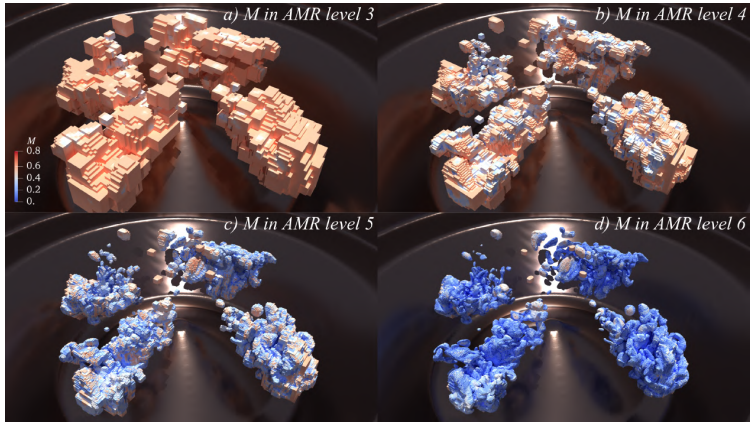
(image credit: pixabay)

Compression of scientific data ($\mathrm{x} \times y \times z \times v \times t$)



(image credit: J. Shadid)

Ignition detection in combustion data ($\mathrm{x} \times y \times z \times v \times t$)



(image credit: Pele/ECP)

Compression of training data in AI (x-pixel × y-pixel × image)

(image credit: pixabay)

Basic Definitions

Terminology:

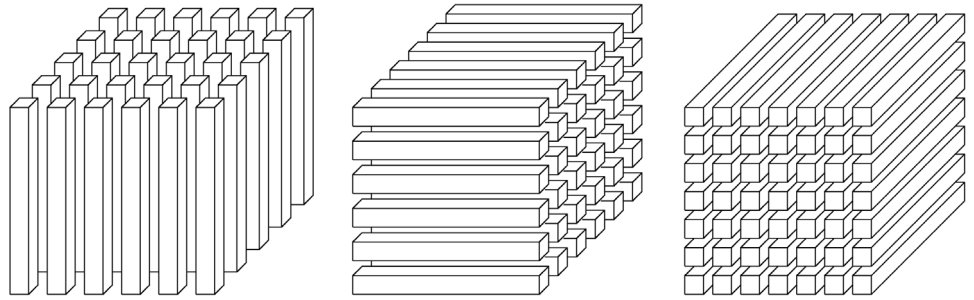
- Mode: a dimension of the tensor, e.g., a 3-way tensor has 3 modes
- Fiber: hold all but one index fixed, e.g., $\boldsymbol{\mathcal{X}}(i,:,k)$
- Slice: Fix a single index, e.g., $\boldsymbol{\mathcal{X}}(i,:,:)$

Matricization:

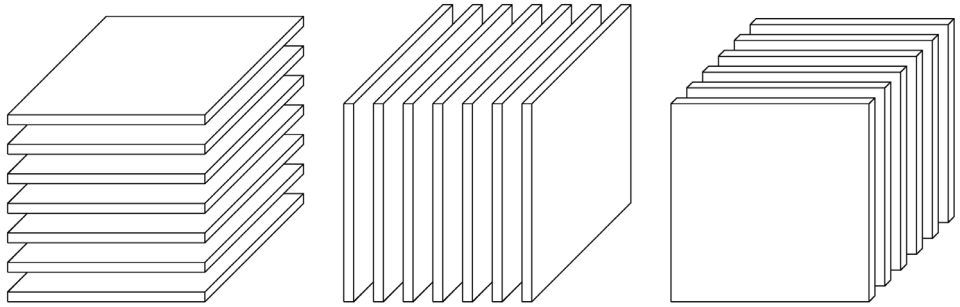
- Reshaping a tensor into a matrix
- Mode-n matricization $\boldsymbol{X}_{(n)}$: mode-n fibers become columns

Khatri-Rao Product:

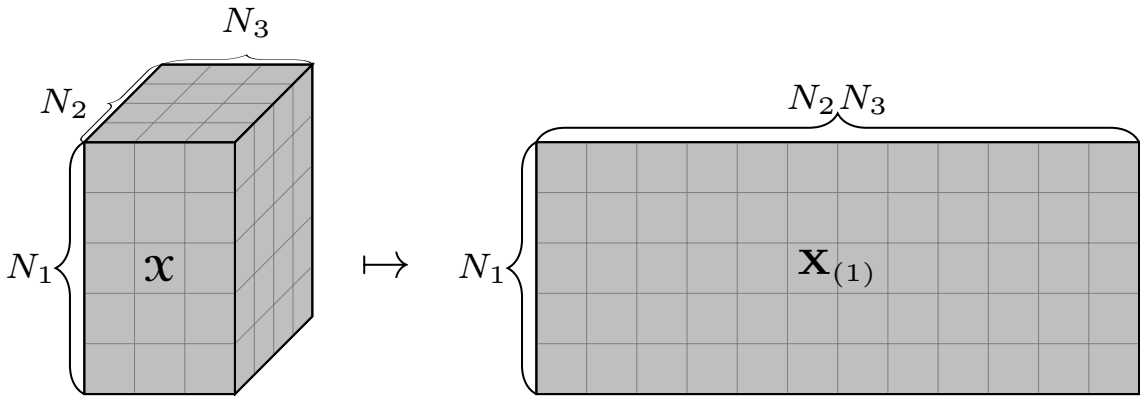
- Column-wise Kronecker product of matrices
- $\boldsymbol{A} = [\boldsymbol{A}_1 \cdots \boldsymbol{A}_R] \in \mathbb{R}^{N_1 \times R}, \boldsymbol{B} = [\boldsymbol{B}_1 \cdots \boldsymbol{B}_R] \in \mathbb{R}^{N_2 \times R},$
- $\boldsymbol{C} = \boldsymbol{A} \odot \boldsymbol{B} = [\boldsymbol{A}_1 \otimes \boldsymbol{B}_1 \cdots \boldsymbol{A}_R \otimes \boldsymbol{B}_R] \in \mathbb{R}^{N_1 N_2 \times R}$



Mode 1, 2, and 3 fibers*



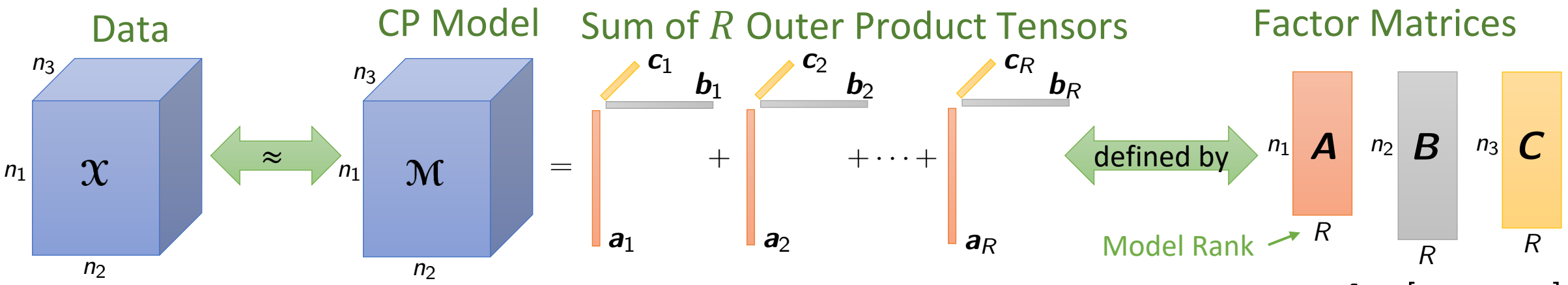
Mode 1, 2, and 3 slices*



Mode 1 matricization

*From T. Kolda and B. Bader, Tensor Decompositions and Applications, SIREV, 2009.

Canonical Polyadic (CP) Tensor Decomposition



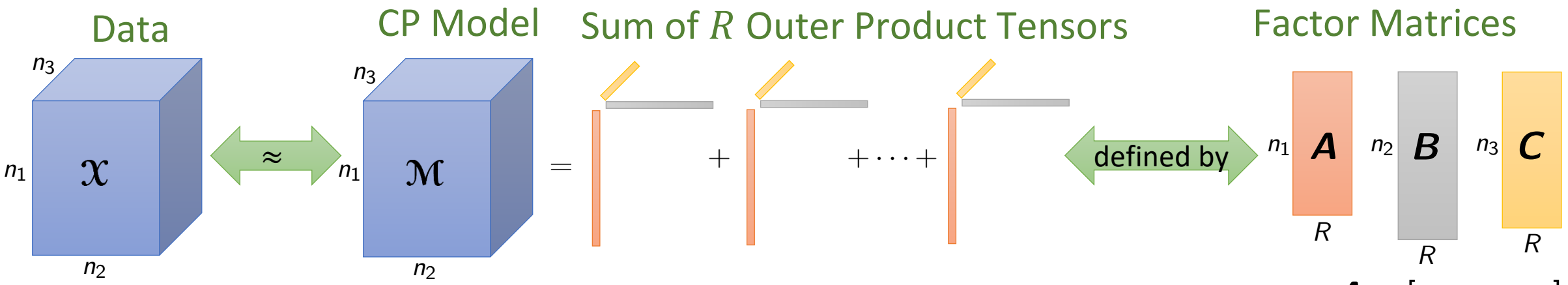
$$\boldsymbol{A} = [\boldsymbol{a}_1 \ \ldots \ \boldsymbol{a}_R]$$
$$\boldsymbol{B} = [\boldsymbol{b}_1 \ \ldots \ \boldsymbol{b}_R]$$
$$\boldsymbol{C} = [\boldsymbol{c}_1 \ \ldots \ \boldsymbol{c}_R]$$

$$x(i,j,k) \approx m(i,j,k) = \sum_{l=1}^{R} a(i,l)b(j,l)c(k,l) = \sum_{j=1}^{R} \lambda_l \hat{a}(i,l)\hat{b}(j,l)\hat{c}(k,l)$$

Order sum with decreasing λ

Vectors comprising each factor describe patterns in the data

Computing CP Decompositions



$$\min_{\mathcal{M}} \ \|\mathcal{X}-\mathcal{M}\|_F^2 = \sum_{i,j,k} \left(x(i,j,k)-m(i,j,k)\right)^2 \ \text{ s.t. } \ \mathcal{M} = [\![\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!]$$

$$\boldsymbol{A} = [\boldsymbol{a}_1 \ \ldots \ \boldsymbol{a}_R]$$
$$\boldsymbol{B} = [\boldsymbol{b}_1 \ \ldots \ \boldsymbol{b}_R]$$
$$\boldsymbol{C} = [\boldsymbol{c}_1 \ \ldots \ \boldsymbol{c}_R]$$

Workhorse algorithm is Alternating Least Squares (CP-ALS):

Using…

$$\mathcal{M}_{(1)} = \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T$$
$$\mathcal{M}_{(2)} = \boldsymbol{B}(\boldsymbol{C} \odot \boldsymbol{A})^T$$
$$\mathcal{M}_{(3)} = \boldsymbol{C}(\boldsymbol{B} \odot \boldsymbol{A})^T$$

Repeat until convergence…

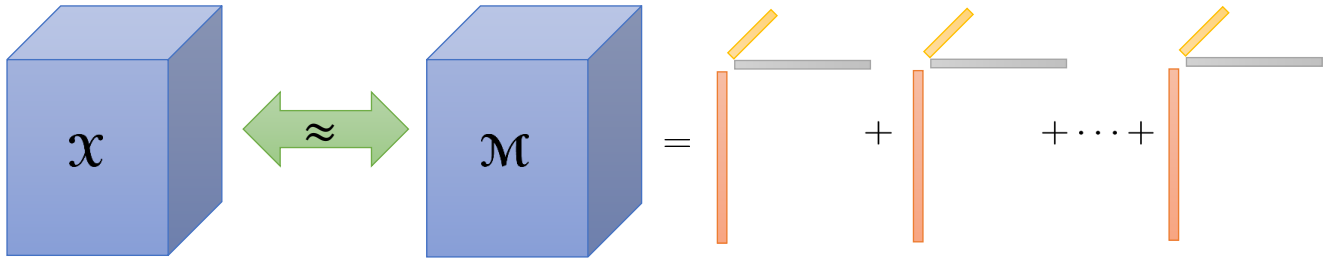
$$\min_{\boldsymbol{A}} \left\|\mathcal{X}_{(1)} - \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T\right\|_F^2$$
$$\min_{\boldsymbol{B}} \left\|\mathcal{X}_{(2)} - \boldsymbol{B}(\boldsymbol{C} \odot \boldsymbol{A})^T\right\|_F^2$$
$$\min_{\boldsymbol{C}} \left\|\mathcal{X}_{(3)} - \boldsymbol{C}(\boldsymbol{B} \odot \boldsymbol{A})^T\right\|_F^2$$

Repeat until convergence…

$$\boldsymbol{A} = \mathcal{X}_{(1)}(\boldsymbol{C} \odot \boldsymbol{B})(\boldsymbol{C}^T\boldsymbol{C} * \boldsymbol{B}^T\boldsymbol{B})^\dagger$$
$$\boldsymbol{B} = \mathcal{X}_{(2)}(\boldsymbol{C} \odot \boldsymbol{A})(\boldsymbol{C}^T\boldsymbol{C} * \boldsymbol{A}^T\boldsymbol{A})^\dagger$$
$$\boldsymbol{C} = \underbrace{\mathcal{X}_{(3)}(\boldsymbol{B} \odot \boldsymbol{A})}_{\text{MTTKRP}}(\boldsymbol{B}^T\boldsymbol{B} * \boldsymbol{A}^T\boldsymbol{A})^\dagger$$

But there are many others (e.g., full Newton, quasi-Newton, nonlinear least squares, …)

Generalized CP (GCP) Tensor Decomposition Allows Flexible Loss Function



Generalized CP (GCP)

$$\min_{\mathcal{M}} F(\mathcal{X}, \mathcal{M}) = \sum_i f(x_i, m_i)$$

$$\text{s.t.} \quad \mathcal{M} = [\![\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!]$$

Example Loss Functions

Normal ($x, m \in \mathbb{R}$)

$$f(x, m) = (x - m)^2$$

Poisson ($x \in \mathbb{N}, m > 0$)

$$f(x, m) = m - x \log m$$

Bernoulli ($x \in \{0,1\}, m > 0$)

$$f(x, m) = \log(m + 1) - x \log m$$

β-divergence ($x > 0, m > 0, \beta = \frac{1}{2}$)

$$f(x, m) = x/\sqrt{m} + \sqrt{m}$$

Developed through SNL LDRD, FY17-19, PI: Tammy Kolda.

Hong, Kolda, Duersch, Generalized Canonical Polyadic Tensor Decomposition, SIREV 2020.
Kolda and Hong, Stochastic Gradients for Large-Scale Tensor Decomposition, SIMODS 2020.
Phipps and Kolda, Software for Sparse Tensor Decomposition on Emerging Computing Architectures, SISC 2019.

Fitting GCP Model[1]

$$\min_{\mathcal{M}} \; F(\mathcal{X}, \mathcal{M}) = \sum_i f(x_i, m_i) \quad \text{s.t.} \quad \mathcal{M} = [\![\boldsymbol{A}_1, \ldots, \boldsymbol{A}_d]\!]$$

Lose the least-squares structure underlying ALS-type algorithms. Instead pursue gradient-based optimization approach.

Define tensor $\boldsymbol{Y}$ such that

$$y(i_1, \ldots, i_d) = y_i = \frac{\partial f}{\partial m}(x_i, m_i)$$

Then gradient of objective function given by

$$\boldsymbol{G}_k = \frac{\partial F}{\partial \boldsymbol{A}_k} = \mathcal{Y}_{(k)}(\boldsymbol{A}_d \odot \cdots \odot \boldsymbol{A}_{k+1} \odot \boldsymbol{A}_{k-1} \odot \cdots \odot \boldsymbol{A}_1)$$

← MTTKRP!

Unfortunately, $\boldsymbol{Y}$ is in general dense, even when $\boldsymbol{X}$ is sparse, making gradient-based optimization infeasible for large, sparse $\boldsymbol{X}$.

Instead, employ Stochastic Gradient Descent (SGD)-type algorithms where $\boldsymbol{Y}$ is only randomly sampled

- ADAM stochastic optimization method
- Specialized sampling methods for sparse tensors that sample both zeros and nonzeros

[1]Kolda and Hong Stochastic Gradients for Large-Scale Tensor Decomposition, SIMODS, 2020.

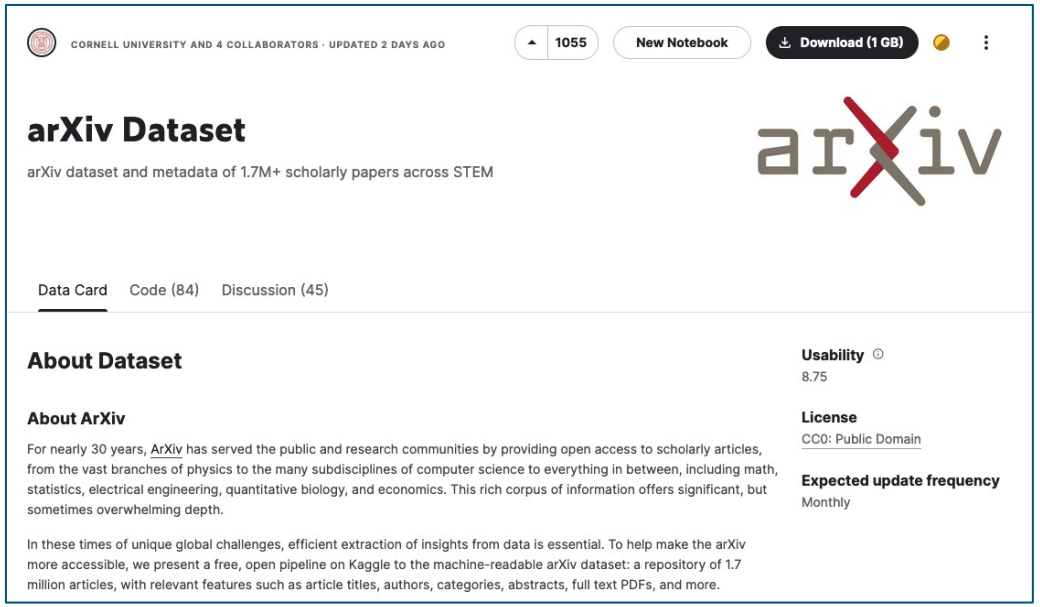
ArXiv Abstract Data

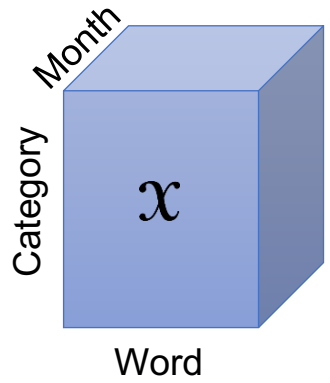
Kaggle provides metadata for ArXiv abstracts that is regularly updated
- https://www.kaggle.com/datasets/Cornell-University/arxiv
- Publication date, going back to first paper backdated to 1986
- Paper category (e.g., cs.LG:Machine Learning)
- Abstracts text

Process abstracts into unique words using Spacy's English core parser
- https://spacy.io/
- Resulted in ~25K unique words

For each month, construct 2-way tensor (matrix):
- Mode 1: Paper category
- Mode 2: Word
- Value: Number of abstracts of the given category containing the given word

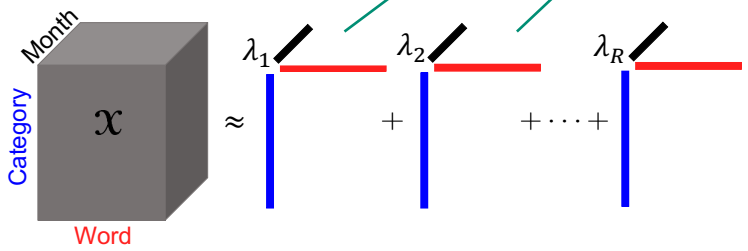
Collection of matrices for each month since start-date forms a sparse 3-way tensor
- Start date is 10 years after first paper (1986) to allow for sufficient volume
- $172 \times 24558 \times 300$, ~2.4% sparsity

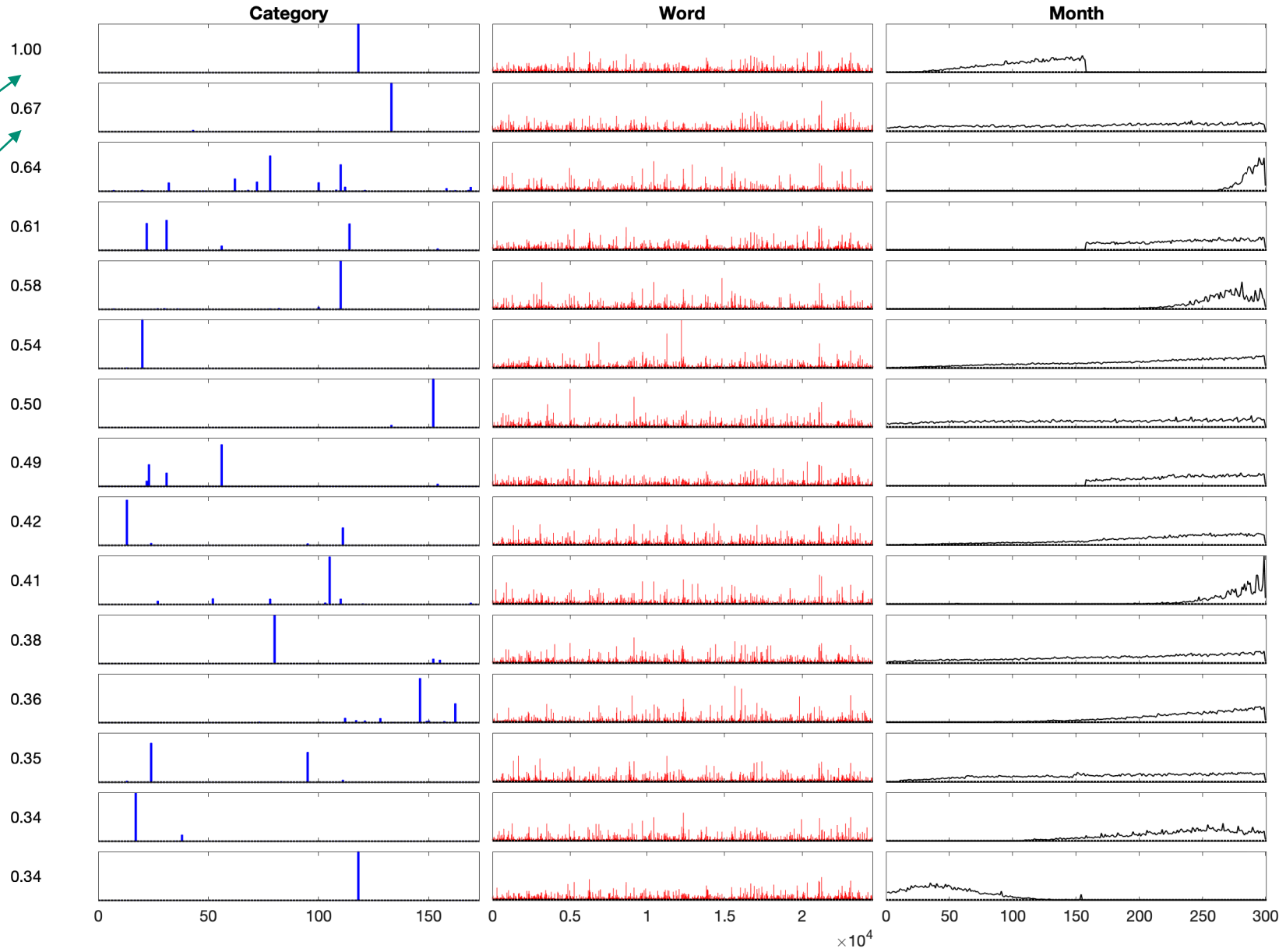




Phipps, Johnson, Kolda, Streaming Generalized Canonical Polyadic Tensor Decompositions, In PASC'23, 2023.

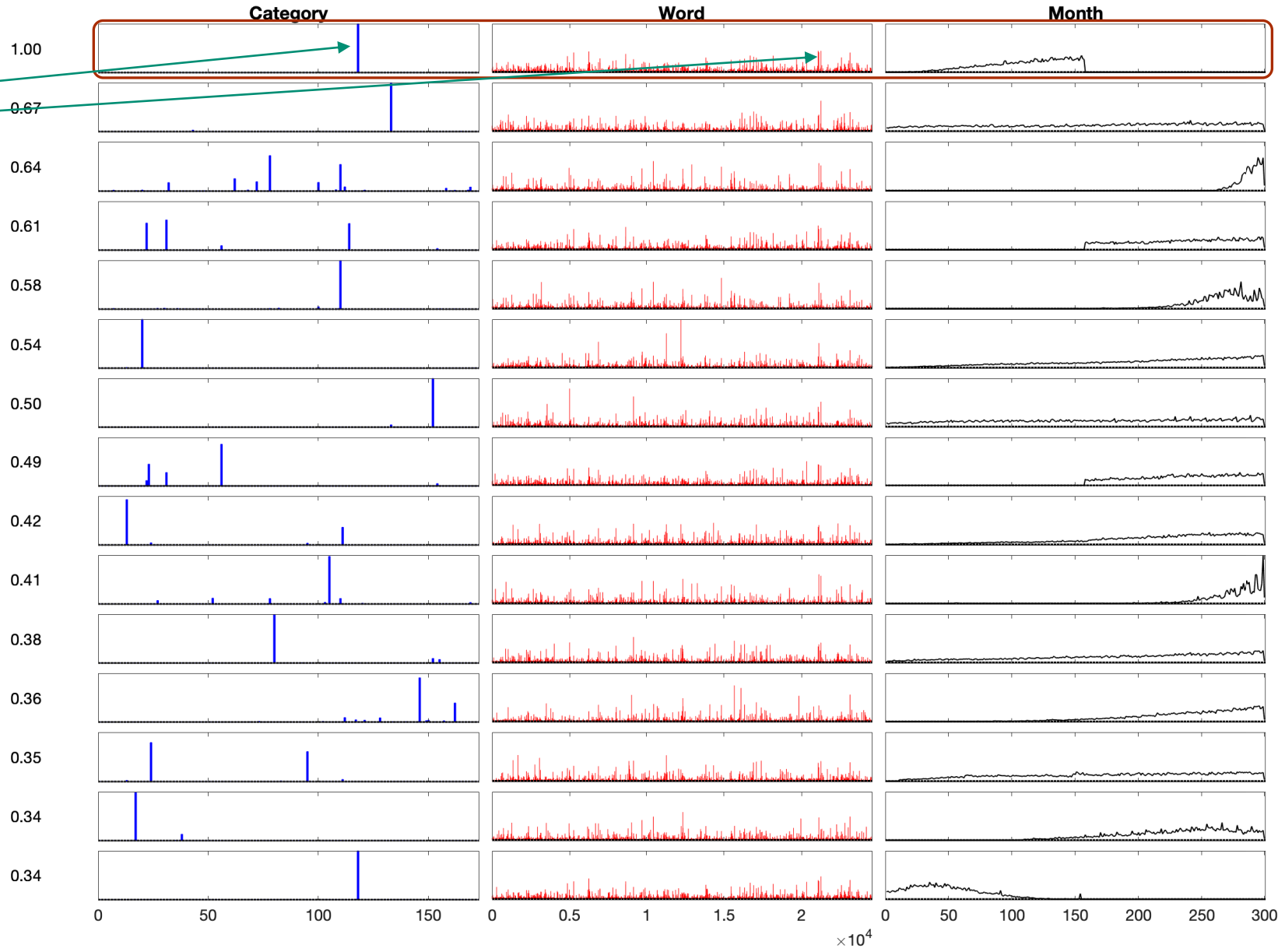
Top 15 (of 50) ArXiv GCP Factors (Poisson Loss)





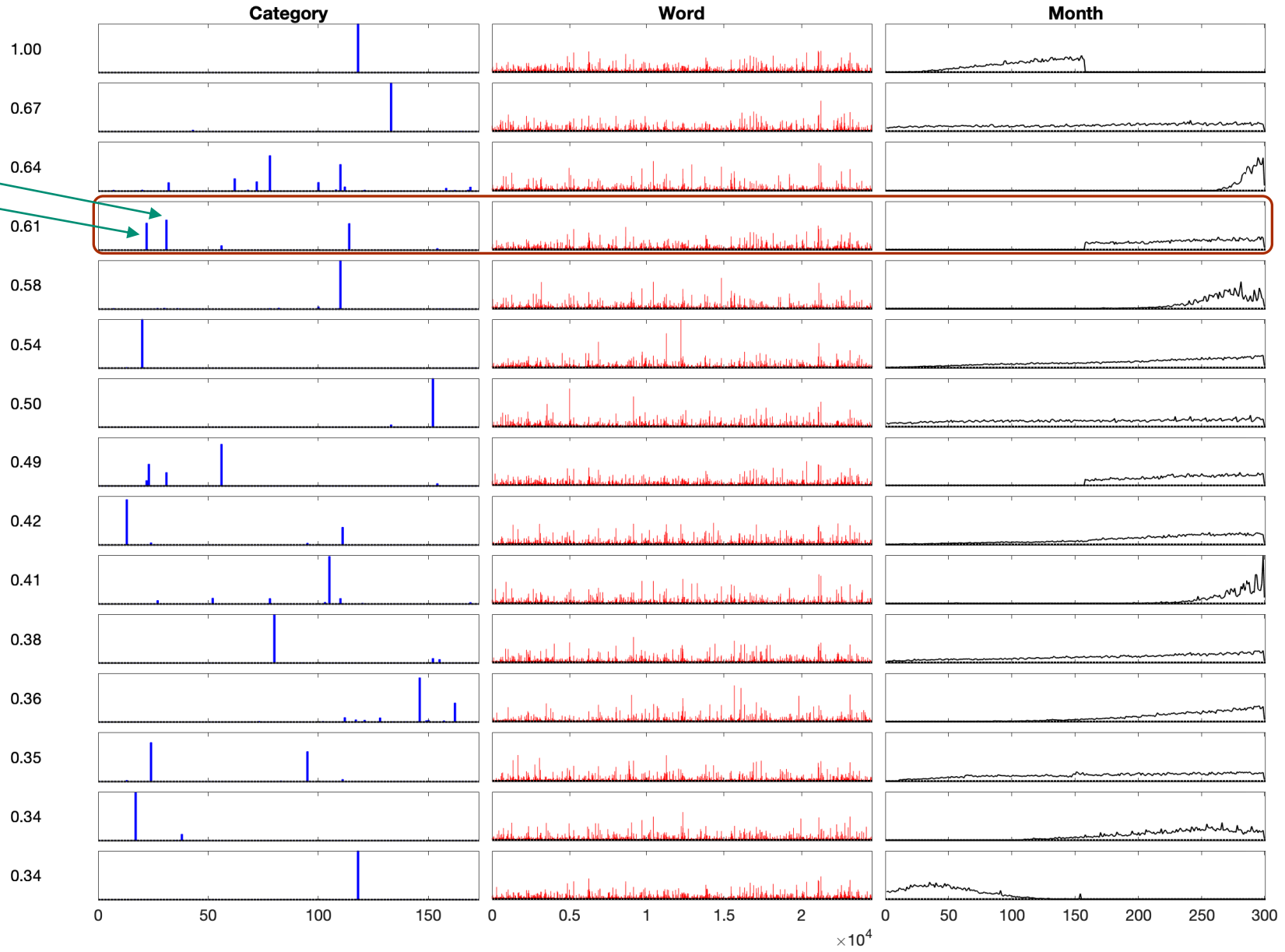
Component #1 (of 50): Astrophysics

Top Categories	Category Weight	Top Words	Word Weight
astro-ph	1.00	find	1.00
		model	1.00
		present	0.97
		use	0.94
		star	0.92
		high	0.92
		result	0.92
		galaxy	0.85
		observation	0.77
		large	0.77
		observe	0.75
		mass	0.72
		datum	0.69
		low	0.68
		study	0.67
		ray	0.64
		emission	0.64
		time	0.62
		spectrum	0.62
		field	0.60



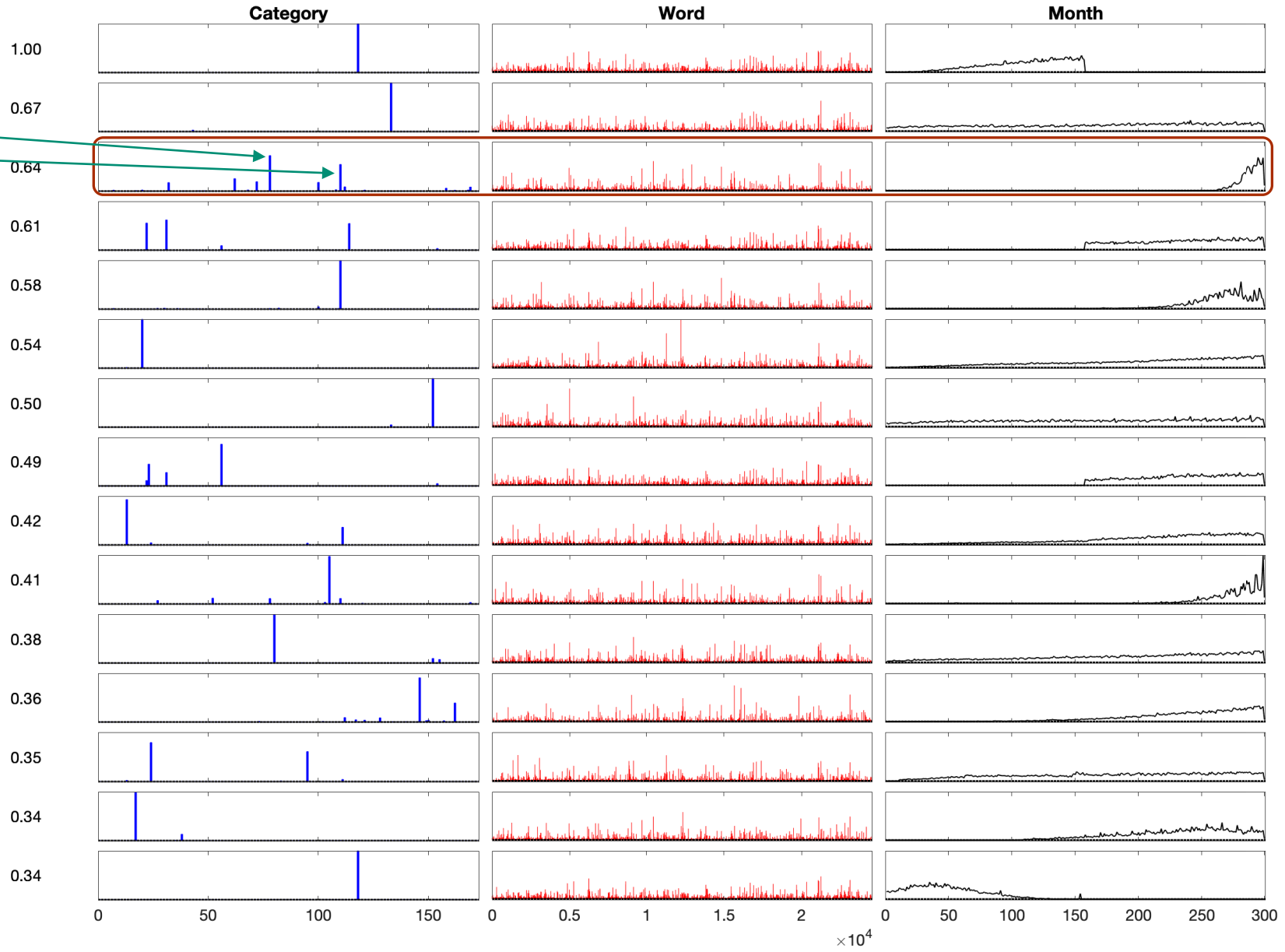
Component #4 (of 50): Astrophysics Reorganized

Top Categories	Category Weight	Top Words	Word Weight
astro-ph.GA	1.00	find	1.00
astro-ph.HE	0.90	galaxy	0.94
astro-ph.CO	0.88	model	0.90
astro-ph.SR	0.15	high	0.85
		use	0.85
		star	0.75
		large	0.72
		mass	0.72
		result	0.71
		present	0.70
		observation	0.69
		observe	0.64
		study	0.63
		ray	0.62
		datum	0.62
		emission	0.61
		low	0.60
		energy	0.56
		spectrum	0.55
		source	0.54



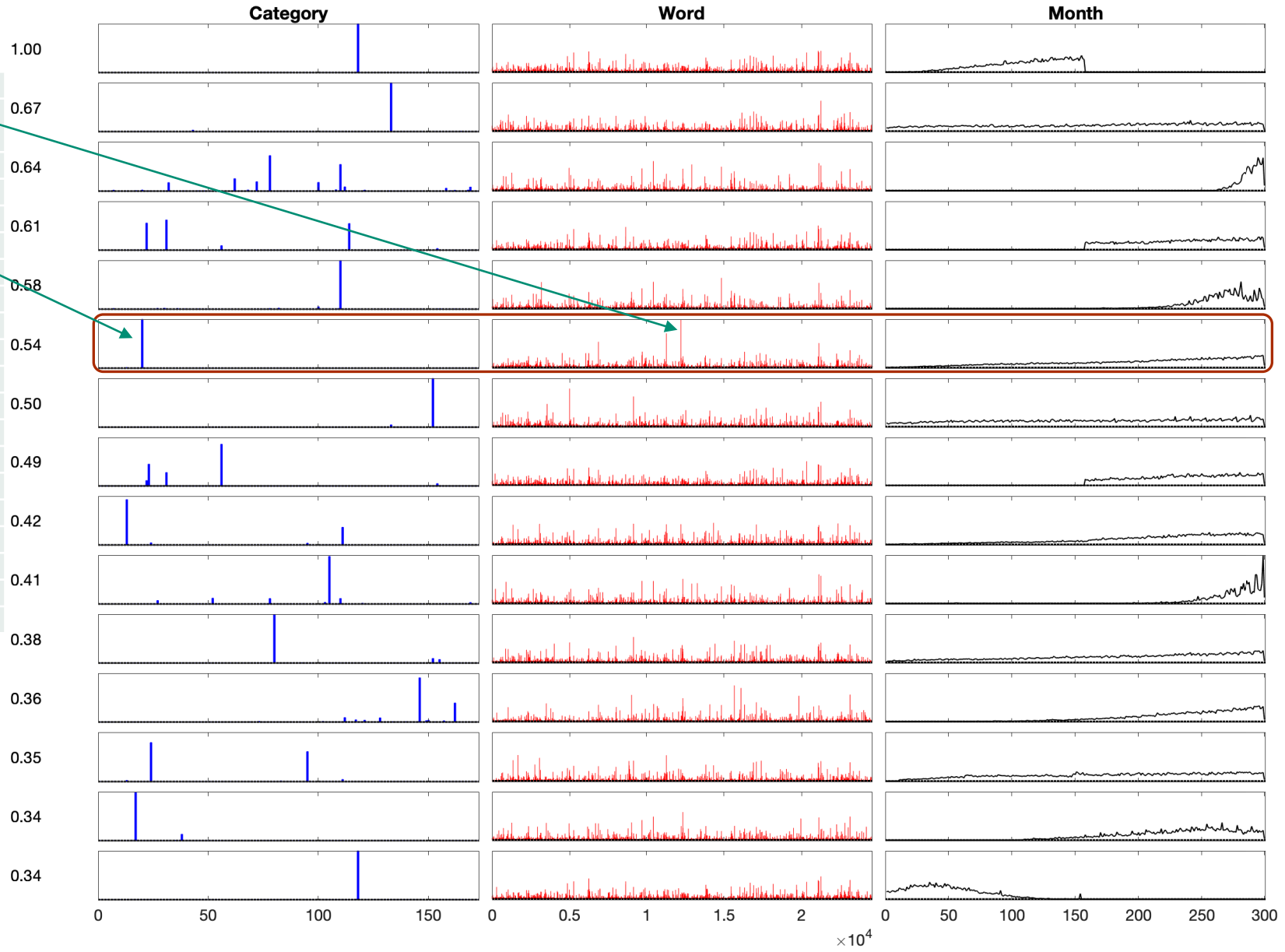
Component #3 (of 50): Computer Vision/Machine Learning

Top Categories	Category Weight	Top Words	Word Weight
cs.LG	1.00	propose	1.00
cs.CV	0.75	use	0.93
eess.IV	0.35	learn	0.89
eess.SP	0.27	model	0.85
cs.RO	0.25	method	0.81
eess.SY	0.24	network	0.77
math.OC	0.12	base	0.75
eess.AS	0.12	paper	0.74
		result	0.73
		train	0.65
		datum	0.65
		performance	0.60
		approach	0.59
		state	0.56
		dataset	0.56
		neural	0.52
		deep	0.51
		problem	0.49
		achieve	0.49
		demonstrate	0.48



Component #6 (of 50): Quantum Physics

Top Categories	Category Weight	Top Words	Word Weight
quant-ph	1.00	quantum	1.00
		state	0.71
		system	0.53
		use	0.51
		result	0.36
		classical	0.33
		time	0.32
		measurement	0.30
		entanglement	0.30
		qubit	0.30
		present	0.29
		information	0.29
		non	0.27
		study	0.26
		single	0.26
		base	0.25
		photon	0.25
		propose	0.24
		provide	0.23
		case	0.23



GenTen : Software for Generalized Canonical Polyadic Tensor Decompositions

GenTen : Software for Generalized Canonical Polyadic Tensor Decompositions[1]

- Started in 2017 through Sandia LDRD
- HPC-oriented toolbox for CP decompositions
- CP-ALS, CP-OPT, GCP-OPT, GCP-SGD, OnlineGCP
- Supports both sparse and dense tensors

Targeted at extreme-scale architectures

- Distributed memory parallelism via MPI
- Shared memory parallelism via Kokkos
  - Multicore CPUs (OpenMP, pThreads)
  - NVIDIA (CUDA), AMD (HIP) and Intel (SYCL) GPUs
- First (and I believe still only), performance-portable CP decomposition tool!

Callable from Matlab and Python

- Integrates with Matlab[2] and Python[3] Tensor Toolboxes
- Manipulate/analyze tensor data but call GenTen HPC decomposition algorithms

[1] https://github.com/sandialabs/GenTen
[2] https://www.tensortoolbox.org/
[3] https://github.com/sandialabs/pyttb

Software Architecture

Predominantly C++-17 code with CMake build system

Mandatory dependencies:

- Kokkos: performance portability
- Kokkos-Kernels: performance portable dense BLAS/LAPACK
- JSON/JSON-Validator: JSON-based input file
- GoogleTest: Test harness
- Pybind11: Python interoperability
- L-BFGS-B: Limited Memory BFGS Quasi-Newton optimization solver (serial, CPU only)

All of these are snapshotted into the GenTen repo using git-subtree and integrated into the CMake build

- Makes it much easier for users to compile without having to download, build, and install dependencies (users only need a compiler, CMake, and BLAS/LAPACK)
- Can also link to externally compiled Kokkos and/or Kokkos-Kernels to support library use-cases

Optional Trilinos dependencies:

- Tpetra: Distributed parallelism
- ROL: Optimization-based CP decompositions
- Teuchos: Stacked timer
- SEACAS: For reading tensors from Exodus files
- Can also be used to provide Kokkos and/or Kokkos-Kernels
- Can pull in Trilinos configure information (compilers, flags, TPLs, etc…) for consistent build and no duplication of settings

Performance Portability with Kokkos*

Performance dominated by MTTRKP computation

- $\boldsymbol{V} = \mathcal{X}_{(1)}(\boldsymbol{C} \odot \boldsymbol{B}) \implies v(i,l) = \sum_{(i,j,k)\in\mathcal{N}(\mathcal{X})} x(i,j,k)b(j,l)c(k,l), \quad l = 1,\ldots,R$
- Coordinate-based sparse tensor storage format
- Row-based factor matrix storage

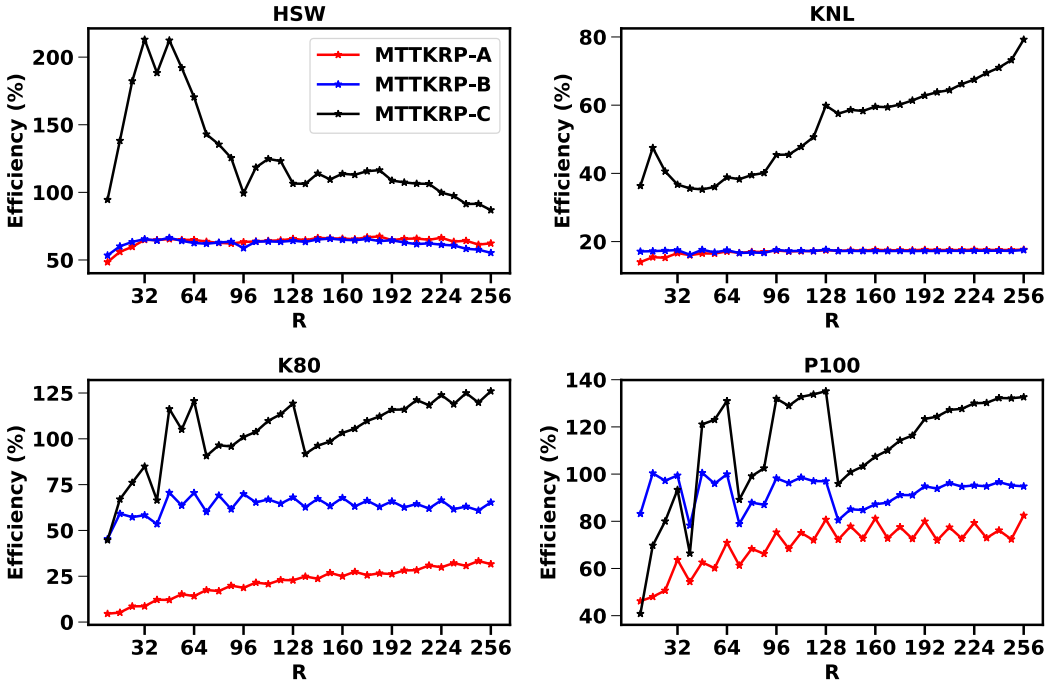
MTTKRP hierarchical parallelism

- Parallelize over nonzeros across thread-teams
- Parallelize over factor matrix columns within a thread-team
- Atomic writes to resolve thread race conditions (Can now also do thread privatization)

MTTKRP algorithms

- A: Kokkos scratch pad arrays for accumulating factor matrix contributions
- B: Store factor matrix contributions in registers through "compile-time polymorphic arrays"
- C: Compute permutation arrays to re-order nonzero loop to reduce atomic contention

MTTKRP Performance as Percentage of Peak Bandwidth



Synthetic data tensor, 30K x 40K x 50K, 10M, non-zeros

Architectures

- HSW (OpenMP): Intel Xeon E5-2698v3 CPU, 2.3 GHz, 64 threads
- KNL (OpenMP): Intel Xeon Phi 7250, 256 threads, 16 GB HBM in cache mode
- K80 (Cuda): Nvidia Kepler K80 GPU, 12 GB GDDR5
- P100 (Cuda): Nvidia Pascal P100 GPU, 16 GB HBM

*Phipps and Kolda, Software for Sparse Tensor Decomposition on Emerging Computing Architectures, SIAM SISC, 2019.

Performance on Real Tensors*

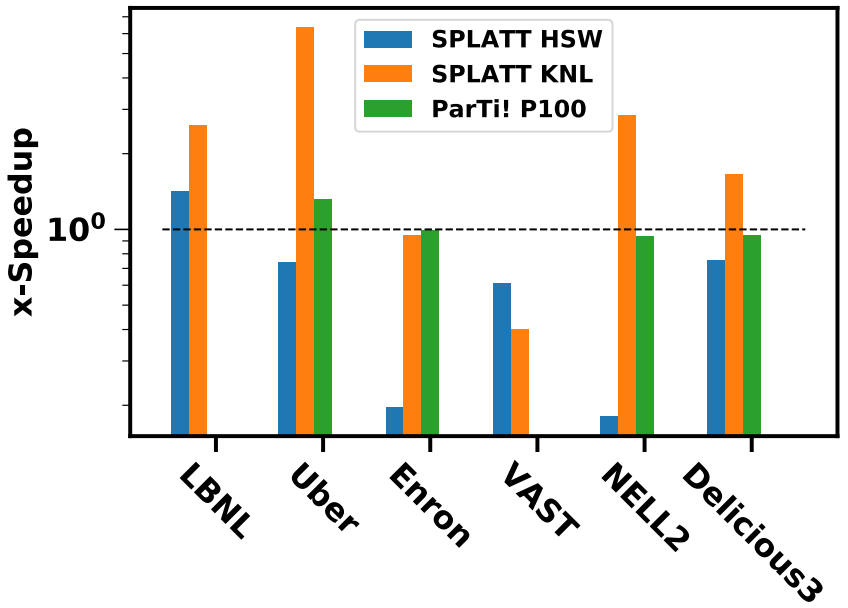
Compare best GenTen MTTKRP to SPLATT (CPU/KNL) and ParTI! (GPU)

- SPLATT: Shaden Smith et al,
  - Compressed Sparse Fiber (CSF) storage format (Smith and Karypis, 2015)
  - Use default 2 CSF modes
- ParTI!: J. Li et al
  - Only works with order-3 and order-4 tensors
  - Requires double-precision atomics (no K80)

Comparable performance to state-of-the-art implementations while retaining performance portability

FROSTT Data Tensors

Name	Order	Dimensions	Nonzeros
LBNL	5	1.6K × 4.2K × 1.6K × 4.2K × 868K	1.7M
Uber	4	183 × 24 × 1.1K × 1.7K	13M
Enron	4	6.0K × 5.7K × 244K × 1.2K	54M
VAST	5	165K × 11K × 2 × 100 × 89	26M
NELL2	3	12K × 9.1K × 29K	77M
Delicious3	3	532K × 17M × 2.5M	140M



*Phipps and Kolda, Software for Sparse Tensor Decomposition on Emerging Computing Architectures, SIAM SISC, 2019.

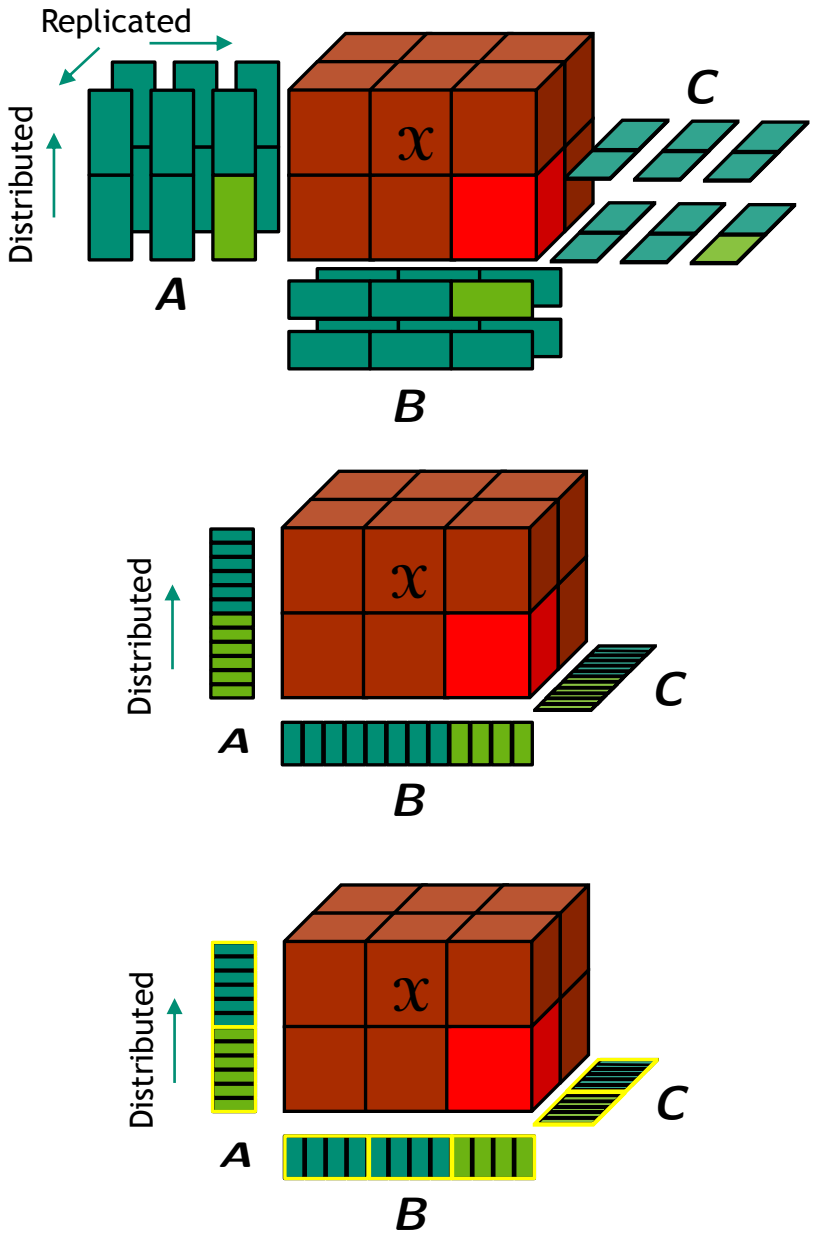
Distributed Parallelism with Tpetra

All approaches uses geometric "medium-grained" partitioning of the tensor
- Supports cartesian MPI sub-communicators

GenTen "All-reduce" approach
- Factor matrices for each dimension distributed across corresponding "fiber" sub-communicator
- Factor matrices replicated across "slice" sub-communicator
- Each processor already has all the data it needs for its local MTTKRP
- Global update is just an all-reduce across slice communicator
- Dense update regardless of sparsity

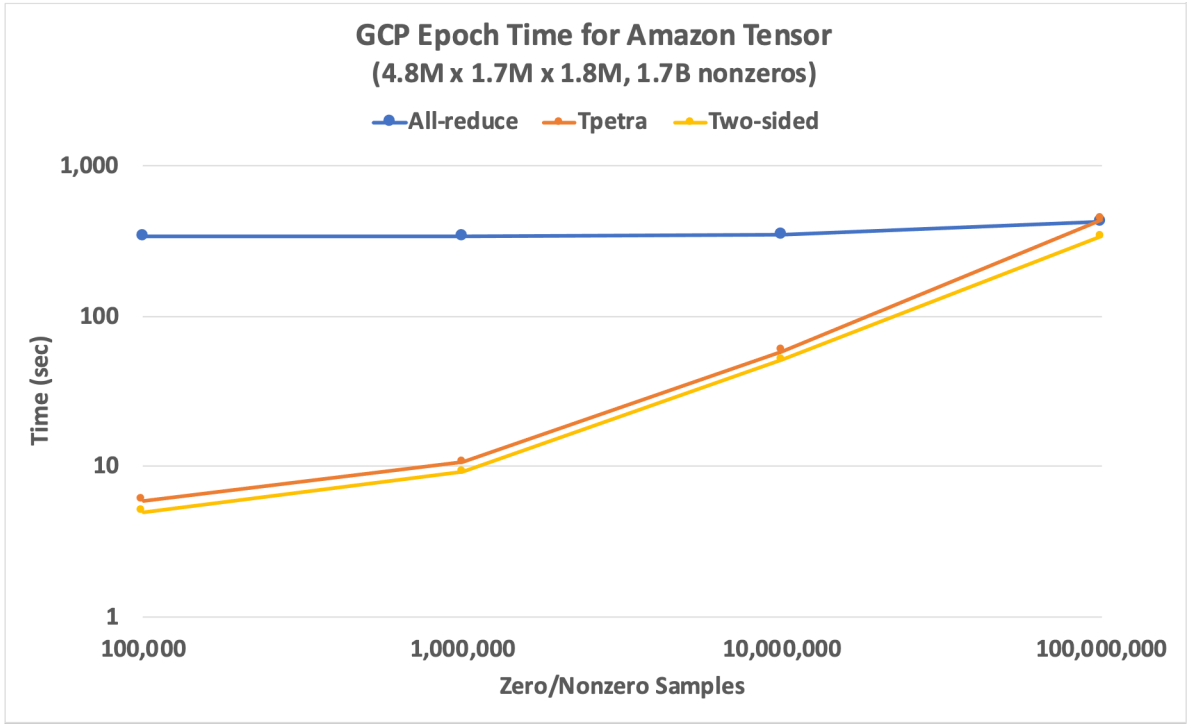
GenTen "Tpetra" approach
- Uniform distribution of factor matrices across all processors
- Sparse, 2-sided point-to-point communication to import needed off-processor factor matrix rows for local MTTKRP
- Similar 2-sided export to send local MTTKRP contributions to owning processors
- Communication pattern depends on sparsity pattern of the tensor (and thus must be updated each GCP step)

GenTen "2-sided" approach
- Similar to Tpetra approach but implemented natively in GenTen
- Factor matrices distributed across all processors, but conformal to geometric partitioning
- Leverages cartesian structure by replacing point-to-point with slice-communicator all-to-all in import/export

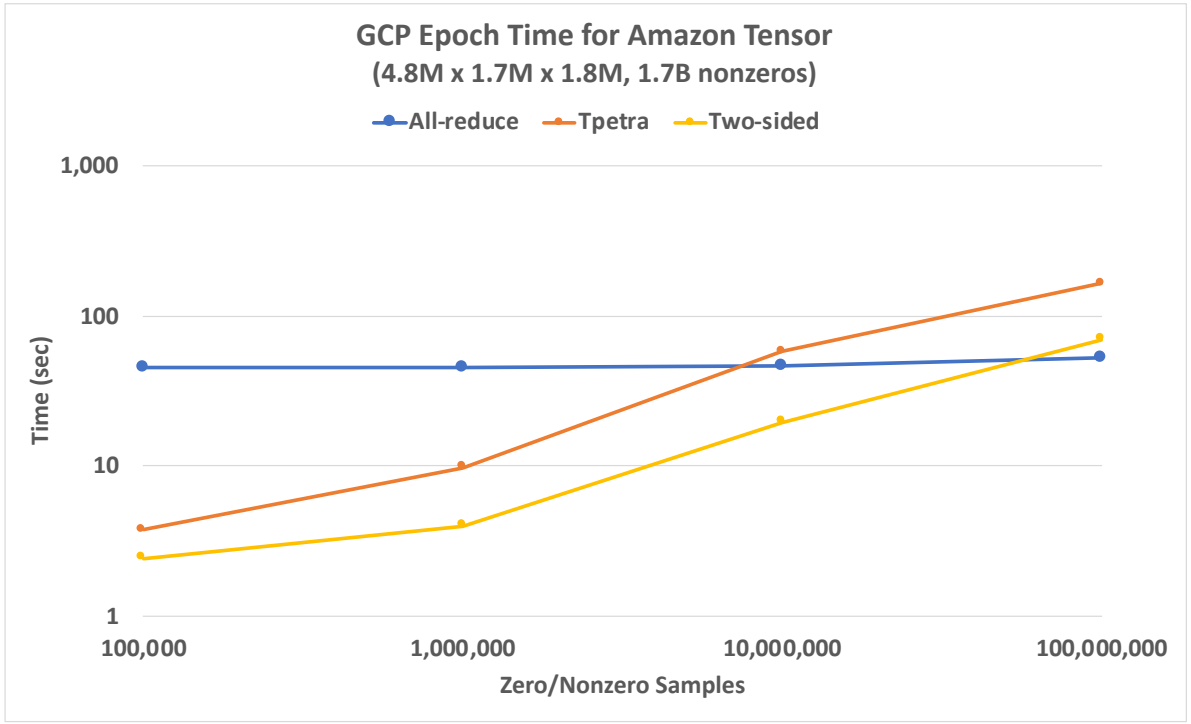


GCP Performance on FROSTT Amazon-Reviews Tensor

CPU (AMD Zen3): 20 Nodes, 28 MPI ranks/node



GPU (NVIDIA A100): 20 Nodes, 1 MPI rank/node



Communication costs for All-reduce approach are essentially constant

Costs grow for Tpetra/2-sided approach as number of samples increase

Some reduction in communication costs by leveraging cartesian structure

- Can this be done in Tpetra

Physics-Informed Tensor Decompositions for Reduction of Scientific Data

Motivation

- Sandia simulations and experiments generate voluminous multidimensional data
- Tensor decompositions (akin to matrix factorizations such as PCA for 2-D data) are a powerful mechanism for reducing multidimensional data
- Existing approaches facilitate massive compression, but don't preserve important physical Quantities of Interest (QoIs) or invariants (e.g., conservation laws)
- Sandia needs algorithmic techniques and software implementations suitable for extreme-scale scientific data and computing architectures

PhITed: In Situ Data Analysis Through Physics-Informed Tensor Decompositions

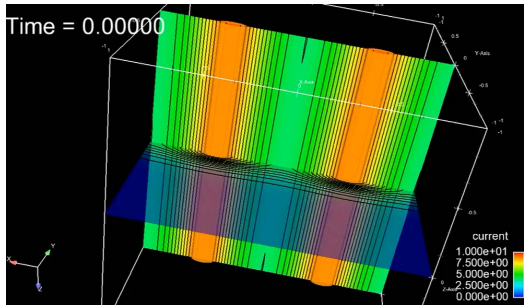
- FY23-25 CIS LDRD, E. Phipps (PI), D. Dunlavy, H. Kolla, J. Shadid
- Developing new goal-oriented approaches that directly target given QoIs/invariants while also maintaining overall reconstruction error

$$\min_{\mathcal{M}} \alpha_0 f(\mathcal{X}, \mathcal{M}) + \sum_{q=1}^{N_q} \alpha_q (G_q(\mathcal{X}) - G_q(\mathcal{M}))^2$$

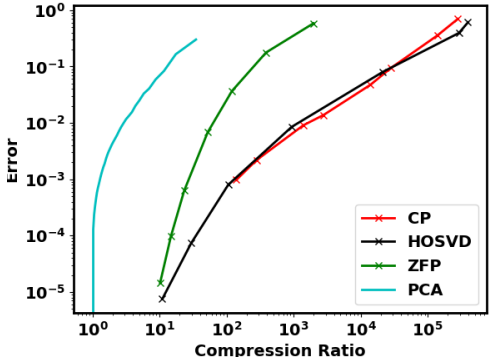
s.t. $\mathcal{M}$ is of CP/Tucker form

Key technologies to make this practical for real scientific applications:

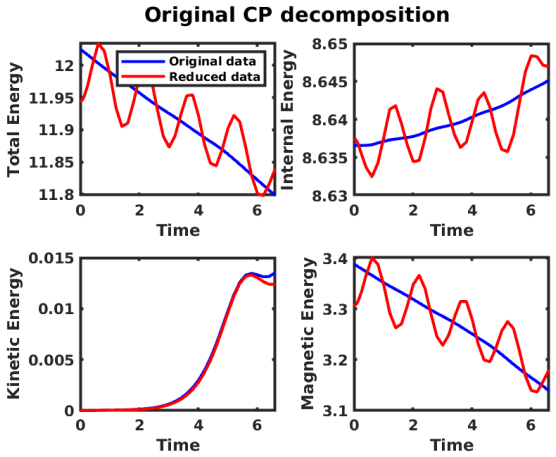
- ROL for efficient and scalable optimization
- Sacado for goal gradients/hessian-vector products



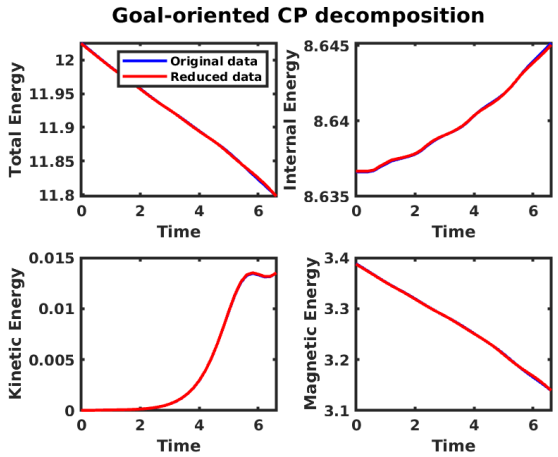
(a)



(b)



(c)



(d)

(a) *MHD plasma physics simulation data relevant for tokamak fusion (3-D island coalescence).* ***(b)*** *Least-squares error versus compression rate comparing tensor methods (CP, HOSVD) with state-of-the-art matrix-based (PCA) and floating-point based (ZFP) methods, demonstrating superiority of tensor-based methods.* ***(c)*** *Several relevant energy QoIs for the original and CP-compressed (~10,000x) MHD data, demonstrating substantial error in these QoIs for the compressed data.* ***(d)*** *Improved energy QoIs for goal-oriented CP tensor compression with no loss in overall reconstruction accuracy or reduction in compression ratio.*

Python Interoperability

GenTen provides python wrappers through pybind11 called pygenten
- GenTen tensor classes and decomposition routines callable from within python
- Thanks to Kim Liegeois for prototyping this capability!
- Works with all Kokkos backends supported by GenTen (OpenMP, CUDA, HIP, SYCL) and MPI
- Provides limited facilities for data manipulation and I/O

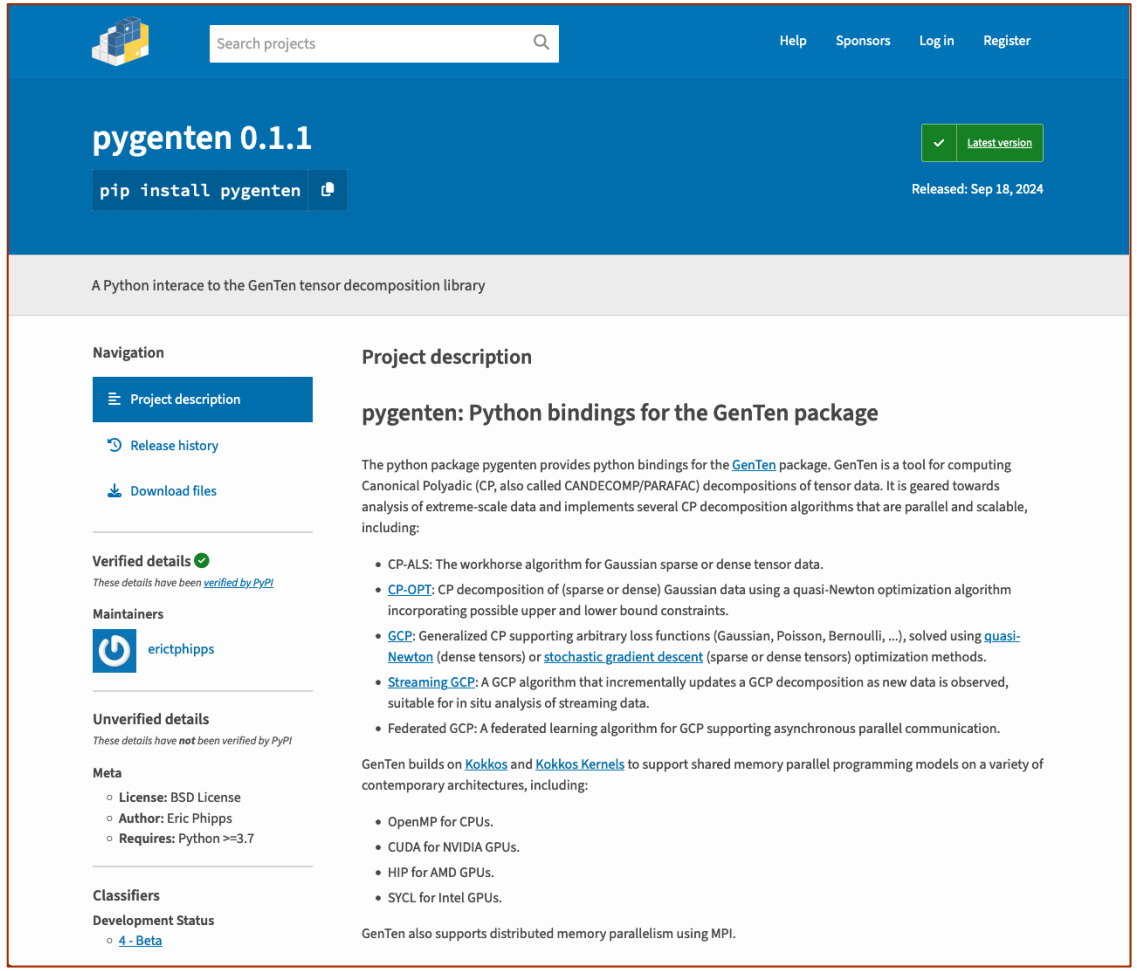
Integrates with the Python Tensor Toolbox (pyttb)
- GenTen decomposition routines accept pyttb tensors as input and return pyttb decomposition results
- Supports passing data back and forth without copies on the CPU
- Tensor data copied from host to device and results back from device to host for GPU
- In the future, pyttb decomposition routines may call pygenten if it is available

Expect pygenten+pyttb to be the primary user-interface for GenTen going forward
- Expect to integrate with pytorch in the future as well to support AI use-cases
- Can also call python code for objective function in, e.g., goal-oriented CP decomposition

Available on pypi.org and installable through pip: "`pip install pygenten`"
- Pre-compiled binary wheels for Intel CPU + OpenMP builds (currently linux only)
- Source build for any other configuration, e.g., for a CUDA+MPI build on V100:

```
pip install --no-binary pygenten                          \
  --config-settings=cmake.define.PYGENTEN_CUDA=ON         \
  --config-settings=cmake.define.Kokkos_ARCH_VOLTA70=ON   \
  --config-settings=cmake.define.PYGENTEN_MPI=ON          \
  pygenten
```

https://pypi.org/project/pygenten/



Long-Term Vision

GenTen is a powerful tool for analysis of high-dimensional data
- Derives a lot of capability from Trilinos

Offline data analysis
- Matlab and Python tensor toolboxes provide powerful data analysis tools in friendly, high-productivity environments
- GenTen integration with these toolkits enable efficient and scalable data analysis
- Work needed on integration with DL frameworks (e.g., pytorch), documentation, porting to Windows, and providing pre-compiled builds (especially GPU builds)

In situ reduction/analysis
- Integration of GenTen into application codes as a library
- Decomposition/reduction of data as it is generated (e.g., each time step) via streaming algorithms
- Goal-oriented approaches to target application-specific QoIs
- Work needed on in situ APIs, file support for compressed tensor formats (e.g., exodus), analysis/viz of compressed tensor data (e.g., paraview)

Trilinos packages play critical roles:
- Kokkos/Kokkos-kernels for performance portability
- Tpetra for distributed parallelism
- ROL for scalable, efficient optimization methods
- Sacado for goal gradients/hessian-vector products
- SEACAS/IOSS for in situ APIs and compressed tensor I/O

Trilinos build system continues to be a stumbling block
- I refuse to have a mandatory dependency on Trilinos as a TPL
- Need to be able to snapshot and build individual Trilinos packages

For More Information

Tensors and tensor decompositions:

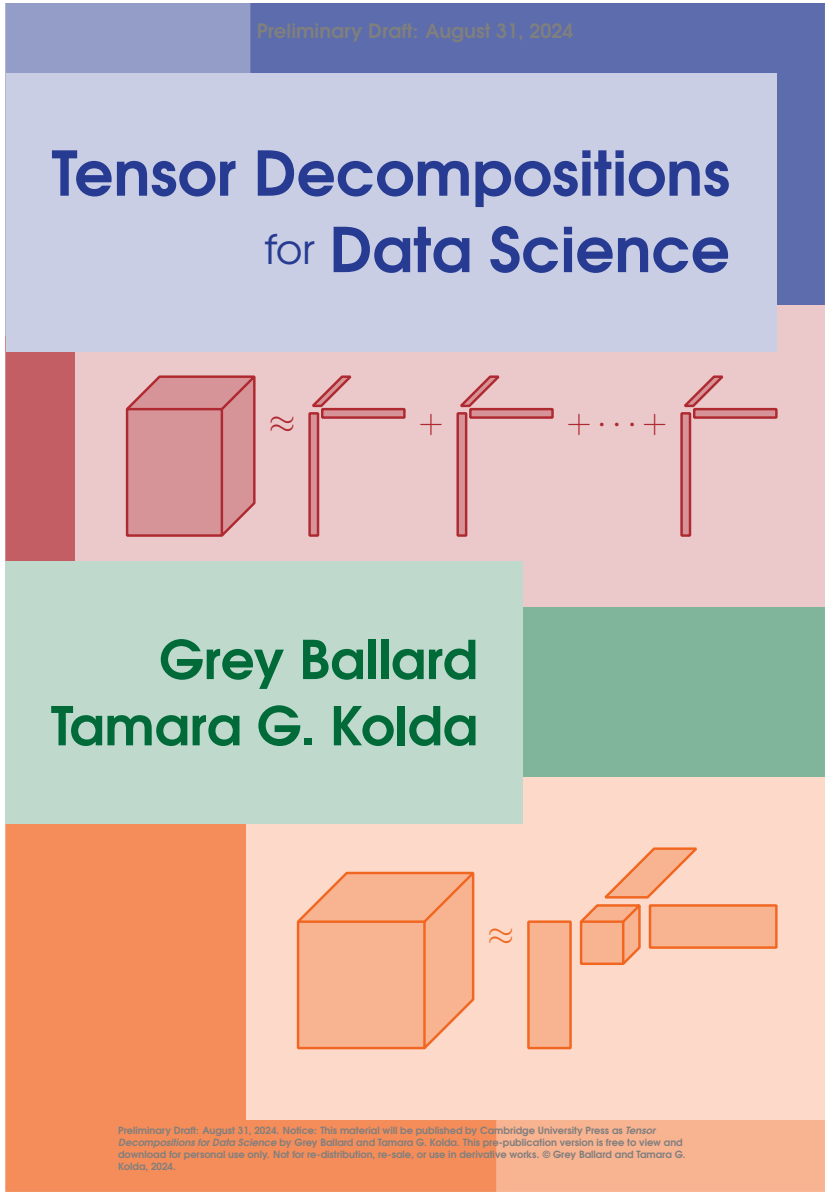
- Kolda and Bader, Tensor Decompositions and Applications, SIAM SIREV, 2009.
- Ballard and Kolda, Tensor Decompositions for Data Science, Cambridge University Press, 2024.

Software:

- GenTen: https://github.com/sandialabs/GenTen
- Python Tensor Toolbox: https://github.com/sandialabs/pyttb
- Matlab Tensor Toolbox: https://www.tensortoolbox.org/

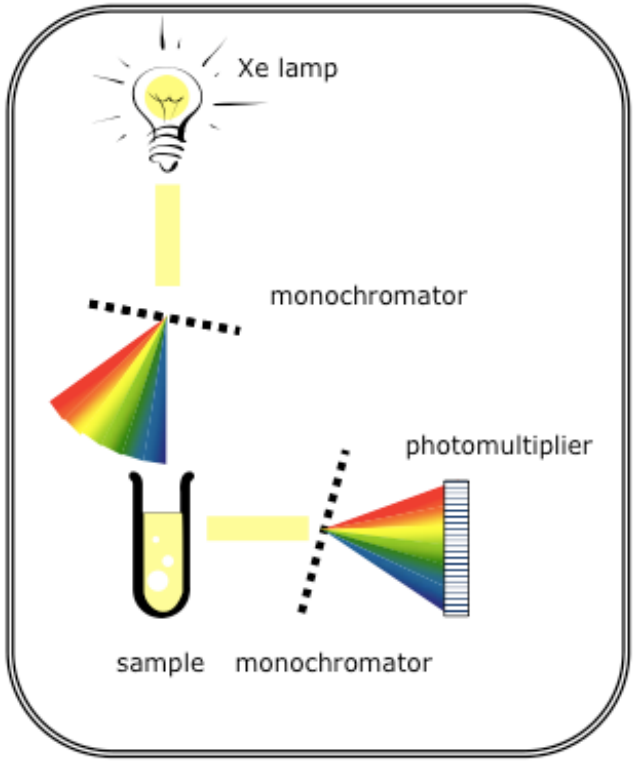
GenTen papers:

- Phipps, Johnson, Kolda, Streaming Generalized Canonical Polyadic Tensor Decompositions, in PASC'23, 2023.
- Lewis and Phipps, Low-Communication Asynchronous Distributed Generalized Canonical Polyadic Tensor Decomposition in HPEC, 2021.
- Phipps and Kolda, Software for Sparse Tensor Decomposition on Emerging Computing Architectures, SIAM SISC, 2019.

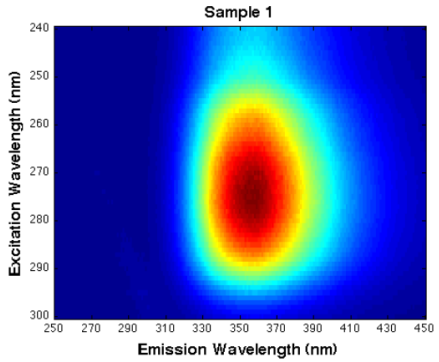


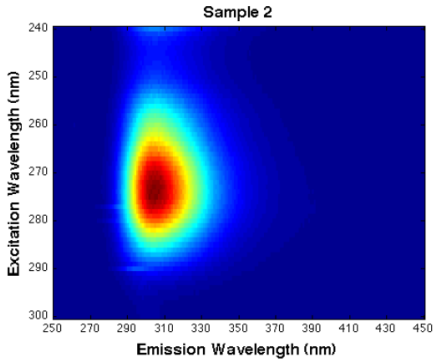
Backup Slides

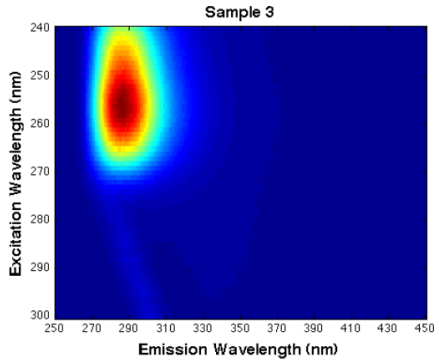
Chemometrics: Amino Acid Example*

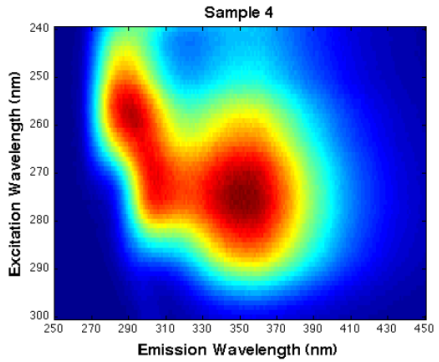


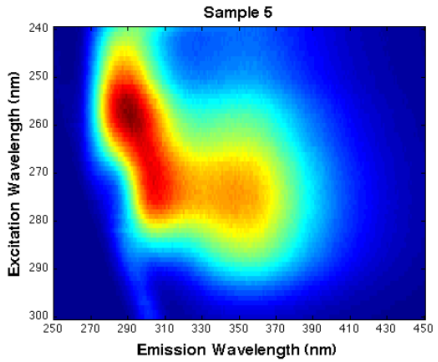
- Mixtures containing different amounts of amino acids
  - Tryptophan, Tyrosine, Phenylalanine
- Fluorescence spectroscopy
  - *sample* X *emission wavelength* X *excitation wavelength* (5 X 61 X 201)
- **Question:** Can we recover the spectra of the analytes in the samples?

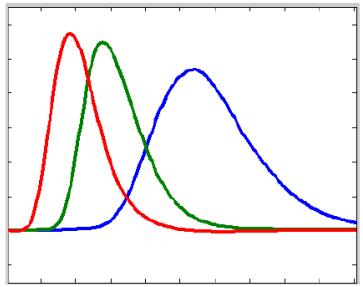








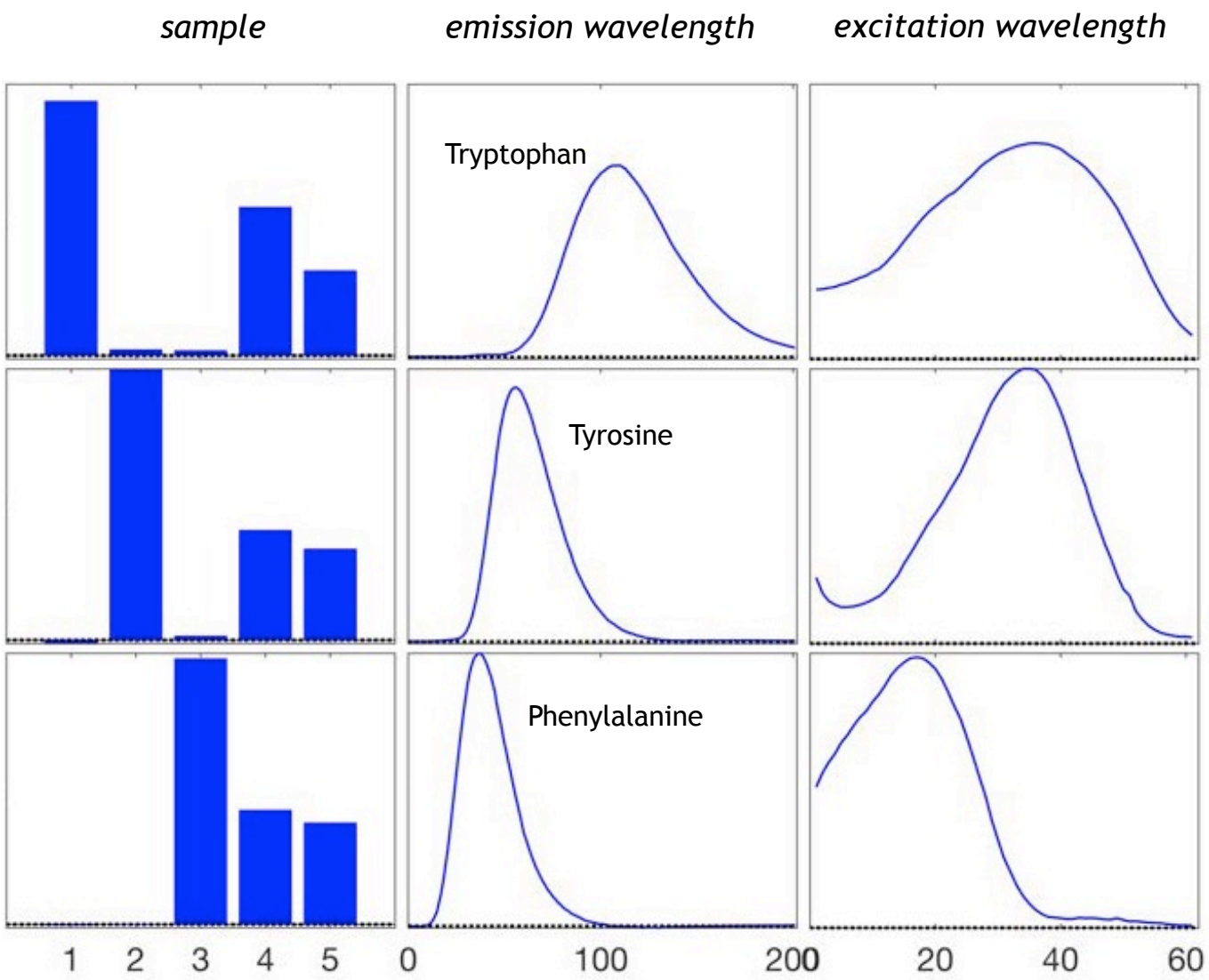




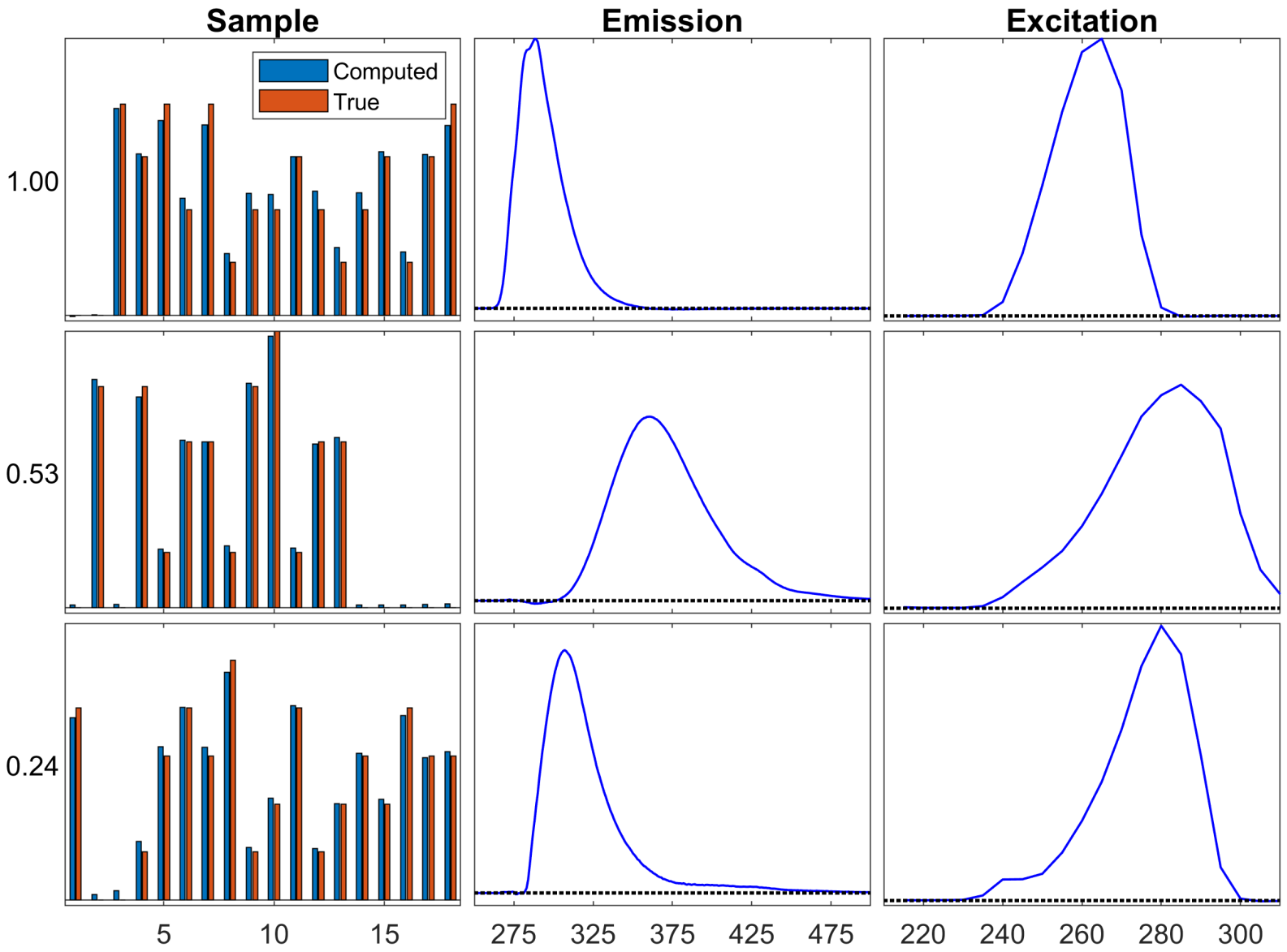
pure spectra of amino acids

*Rasmus Bro, PARAFAC: Tutorial and applications, Chemometrics and Intelligent Laboratory Systems, 1997.

Amino Acid Example: Rank-3 Decomposition Results (Gaussian Loss)



Amino Acid Example: More Samples, More Mixture Variation



GenTen Command-Line Application

```
{
    "solver-method": "cp-als",
    "timings": false,
    "tensor":
    {
        "input-file": "data/aminoacid_data_nn.txt",
        "index-base": 0
    },
    "k-tensor":
    {
        "rank": 16,
        "output-file": "aminoacid.ktn",
        "initial-guess": "rand",
        "distributed-guess": "serial",
        "seed": 12345,
        "prng": false
    },
    "cp-als":
    {
        "maxiters": 20,
        "tol": 1e-4,
        "mttkrp":
        {
            "method": "default"
        }
    }
}
```

```
etphipp@eolus opt_mpi_serial $ mpirun -np 8 ./bin/genten --json data/aminoacid-cpals.json

[ASCII art banner]

--------------------------------------------------------------------------------
GenTen:  Software for Generalized Canonical Polyadic Tensor Decompositions

Copyright 2017 National Technology & Engineering Solutions of Sandia, LLC
(NTESS). Under the terms of Contract DE-NA0003525 with NTESS, the U.S.
Government retains certain rights in this software.
--------------------------------------------------------------------------------

Reading tensor from file data/aminoacid_data_nn.txt
  Read file in: 0.016907s
  Data import took  0.020 seconds

Sparse tensor:
  5 x 201 x 61 (61305 total entries)
  61305 (100.0%) Nonzeros and 0 (0.0%) Zeros
  4.8e+04 Frobenius norm

Execution environment:
  MPI grid: 1 x 8 x 1 processes (8 total)
  Execution space: serial

CP-ALS:
  CP Rank: 16
  MTTKRP method: single
  Gram formulation: symmetric

Iter   1: fit =  9.710805e-01 fitdelta =  9.1e-01
Iter   2: fit =  9.865534e-01 fitdelta =  1.5e-02
Iter   3: fit =  9.876203e-01 fitdelta =  1.1e-03
Iter   4: fit =  9.880996e-01 fitdelta =  4.8e-04
Iter   5: fit =  9.883227e-01 fitdelta =  2.2e-04
Iter   6: fit =  9.884722e-01 fitdelta =  1.5e-04
Iter   7: fit =  9.885804e-01 fitdelta =  1.1e-04
Iter   8: fit =  9.886466e-01 fitdelta =  6.6e-05
Final fit =  9.886466e-01

Saving final Ktensor to aminoacid.ktn
  Ktensor export took  0.003 seconds
```

Python Interoperability

Untitled.ipynb

Code | Python 3 (ipykernel)

```
[1]: import pyttb as ttb
     import pygenten as gt

[2]: x = ttb.import_data("data/aminoacid_data_dense.txt")

[3]: u,_ = gt.cp_als(x, rank=3, maxiters=20, tol=1e-4)
```

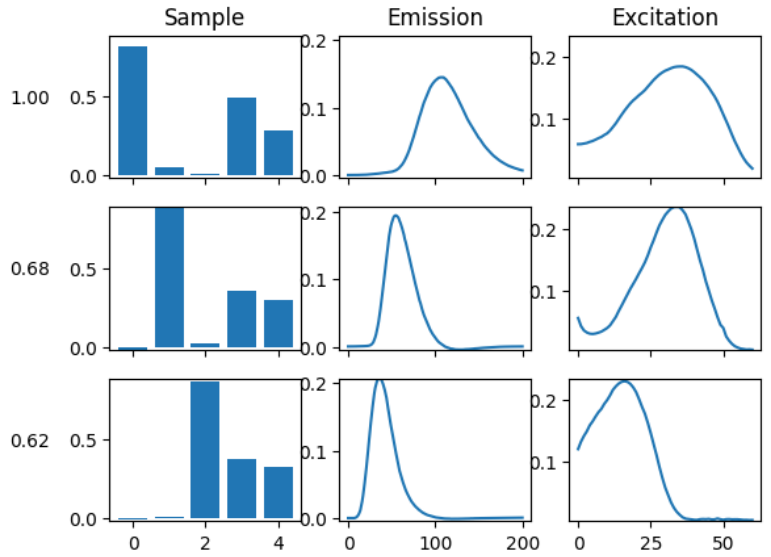
```
Dense tensor:
  5 x 201 x 61 (61305 total entries)
  4.8e+04 Frobenius norm

Execution environment:
  MPI grid: 1 x 1 x 1 processes (1 total)
  Execution space: openmp (8 threads)

CP-ALS:
  CP Rank: 3
  MTTKRP method: row-based
  Gram formulation: symmetric

Iter   1: fit =  4.808158e-01 fitdelta =  4.8e-01
Iter   2: fit =  7.324099e-01 fitdelta =  2.5e-01
Iter   3: fit =  7.736535e-01 fitdelta =  4.1e-02
Iter   4: fit =  7.835859e-01 fitdelta =  9.9e-03
Iter   5: fit =  7.869399e-01 fitdelta =  3.4e-03
Iter   6: fit =  7.887221e-01 fitdelta =  1.8e-03
Iter   7: fit =  7.900780e-01 fitdelta =  1.4e-03
Iter   8: fit =  7.914101e-01 fitdelta =  1.3e-03
Iter   9: fit =  7.929118e-01 fitdelta =  1.5e-03
Iter  10: fit =  7.947348e-01 fitdelta =  1.8e-03
Iter  11: fit =  7.970677e-01 fitdelta =  2.3e-03
Iter  12: fit =  8.002061e-01 fitdelta =  3.1e-03
Iter  13: fit =  8.046694e-01 fitdelta =  4.5e-03
Iter  14: fit =  8.114547e-01 fitdelta =  6.8e-03
Iter  15: fit =  8.226425e-01 fitdelta =  1.1e-02
Iter  16: fit =  8.428068e-01 fitdelta =  2.0e-02
Iter  17: fit =  8.805666e-01 fitdelta =  3.8e-02
Iter  18: fit =  9.357368e-01 fitdelta =  5.5e-02
Iter  19: fit =  9.676515e-01 fitdelta =  3.2e-02
Iter  20: fit =  9.724381e-01 fitdelta =  4.8e-03
Final fit =  9.724381e-01
```

```
[4]: bar=lambda v,ax: ax.bar(range(0,v.shape[0]),v)
     line=lambda v,ax: ax.plot(v)
     u.vis([bar, line, line],mode_titles=["Sample","Emission","Excitation"]);
```




```
[ ]:
```

pyttb+pygenten equivalent to
https://www.tensortoolbox.org/cp_als_doc.html

Chicago Crime Data

4-way count tensor
- 6,186 Days
- 24 Hours of the Day
- 77 Community Areas
- 32 Crime Types

Non-zeros: 5,330,673
- 0.21GB for sparse tensor

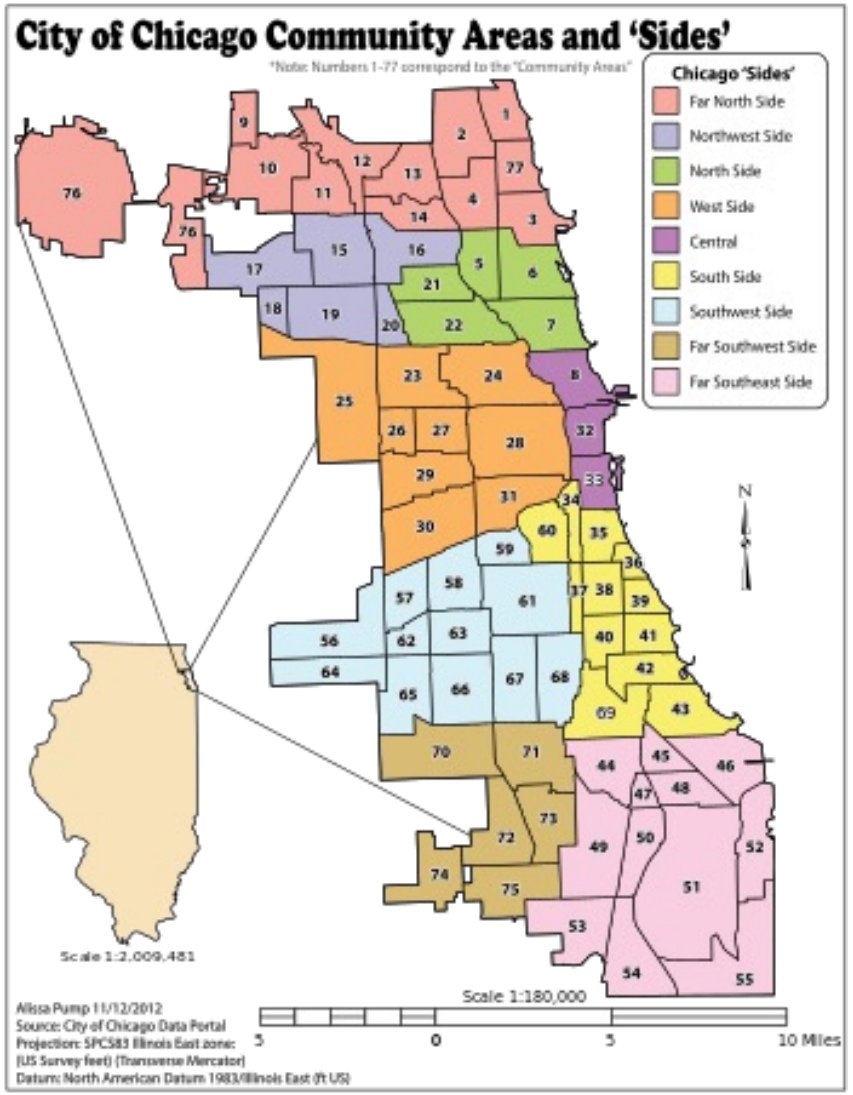
Distribution of entries
- 0: 98.54%
- 1: 1.33%
- ≥ 2: 0.12%

Obtained from FROSTT (http://frostt.io/tensors/chicago-crime/)

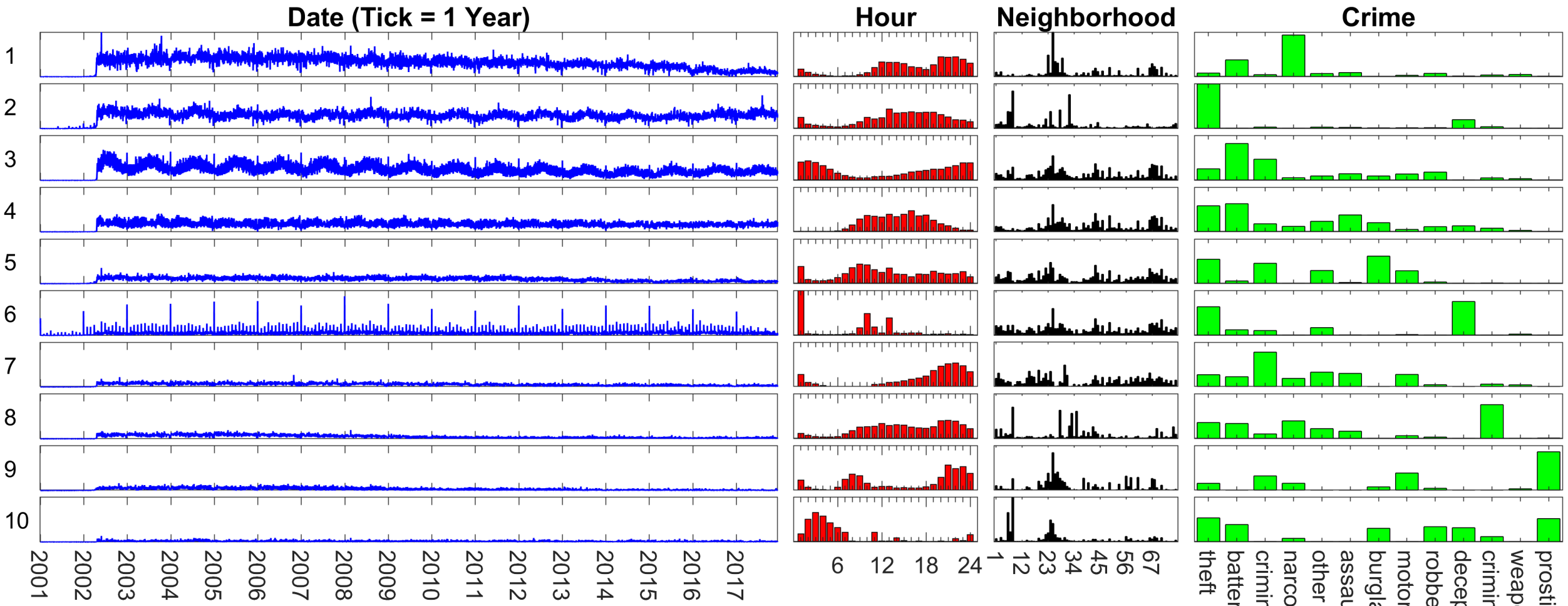
Data originally from Chicago Data Portal (https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2)

Bernoulli Loss
Rank = 10
Samples $s = 6{,}319$

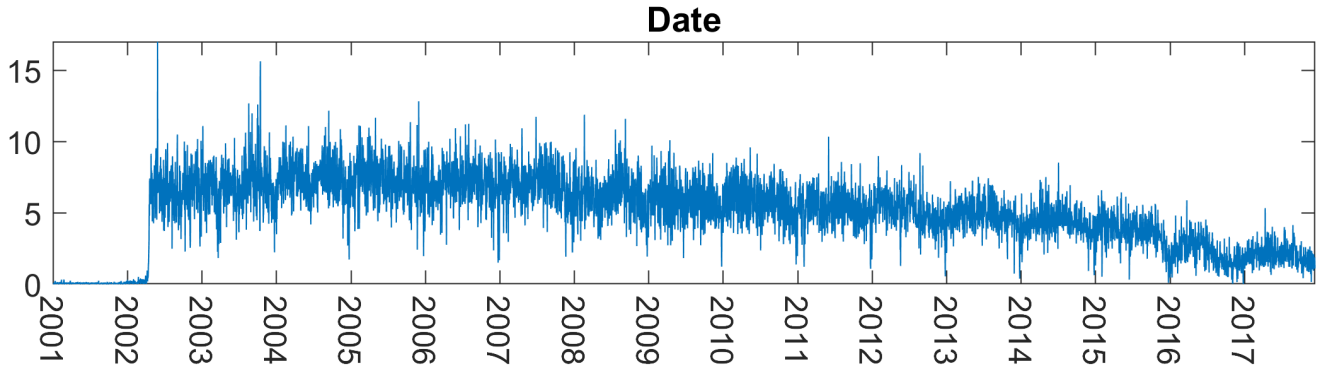
$$f(x, m) = \log(m + 1) - x \log m$$

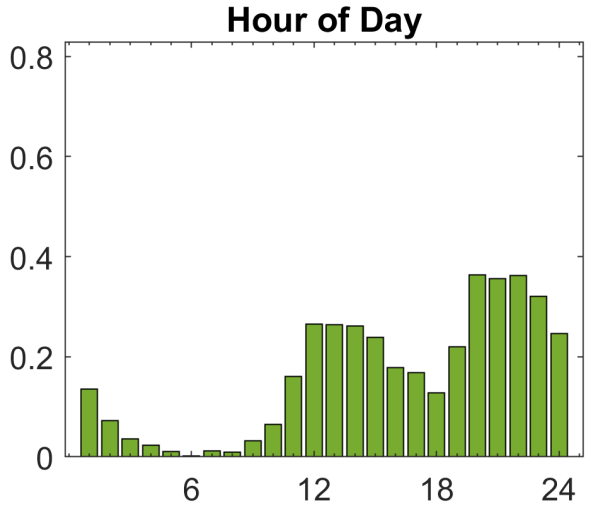


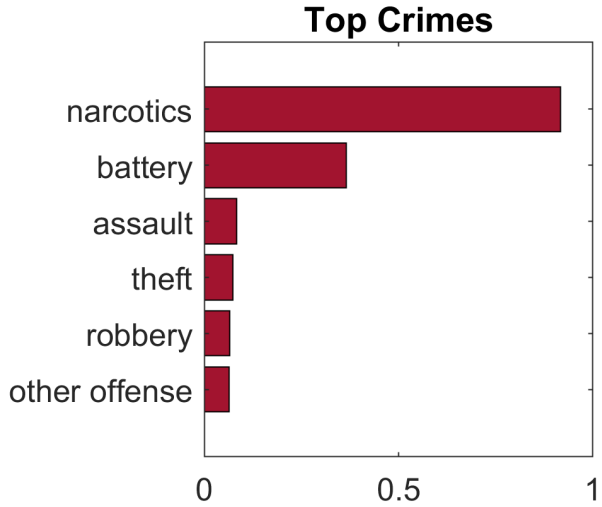
GCP Decomposition of Sparse Crime Binary Tensor (Bernoulli Loss)

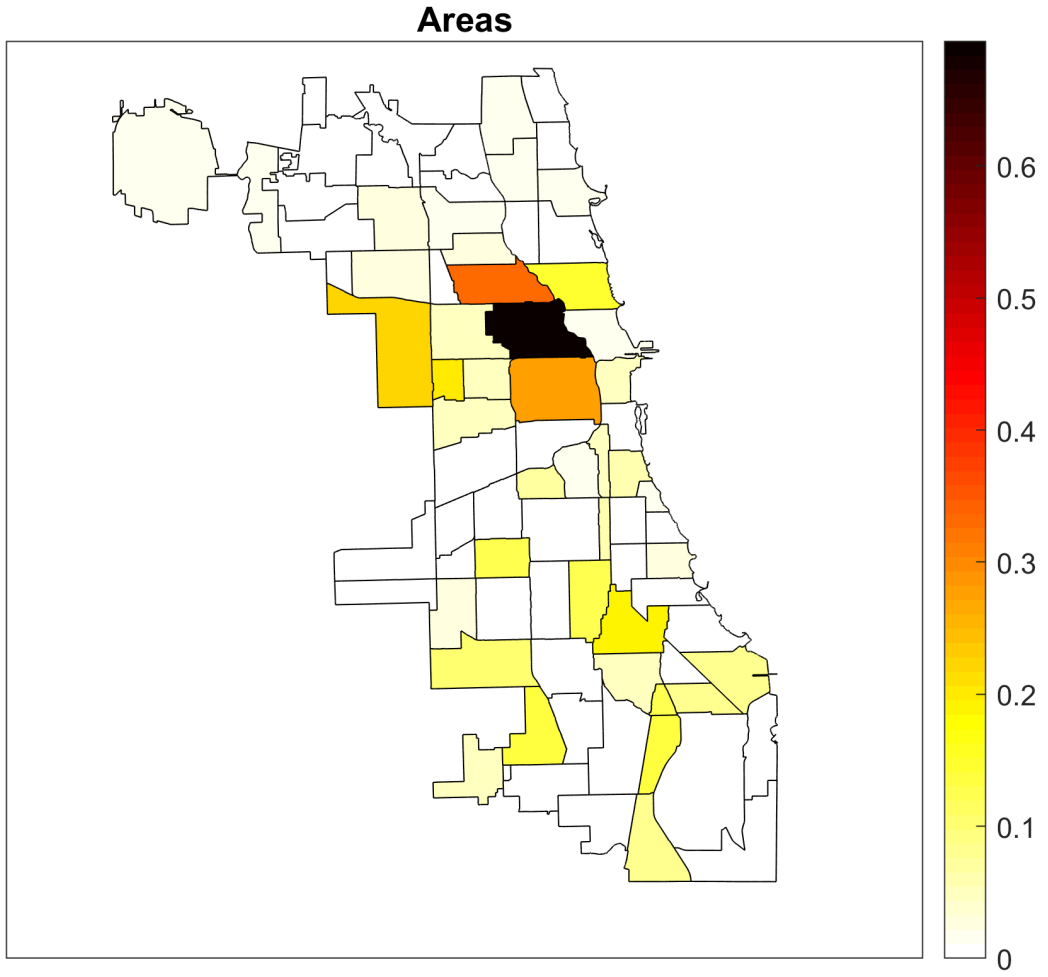


Component #1

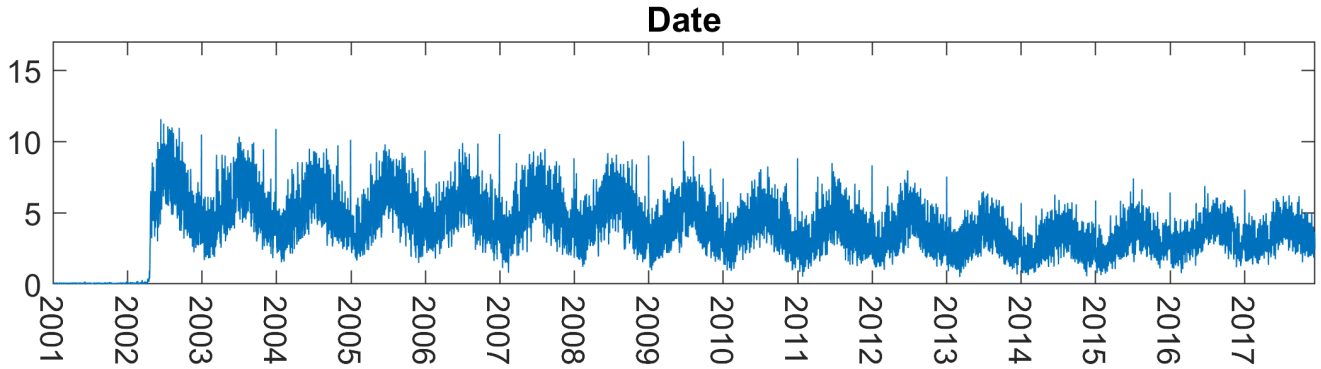

Date
15
10
5
0
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017

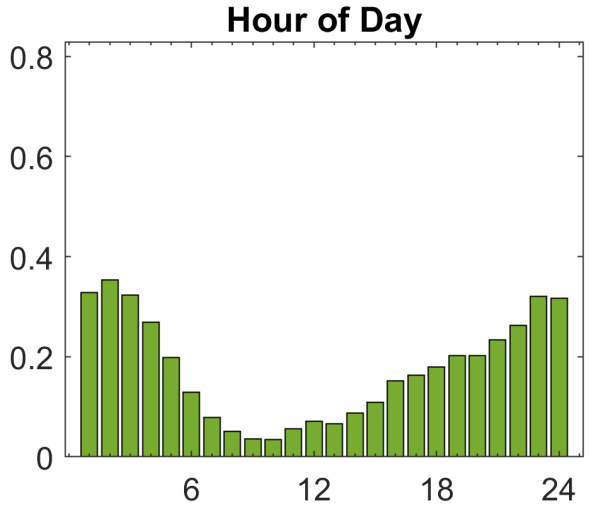

Hour of Day
0.8
0.6
0.4
0.2
0
6
12
18
24

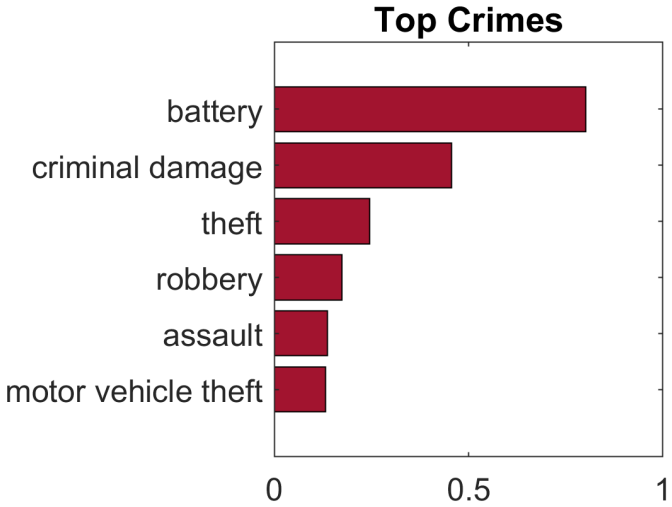

Top Crimes
narcotics
battery
assault
theft
robbery
other offense
0
0.5
1

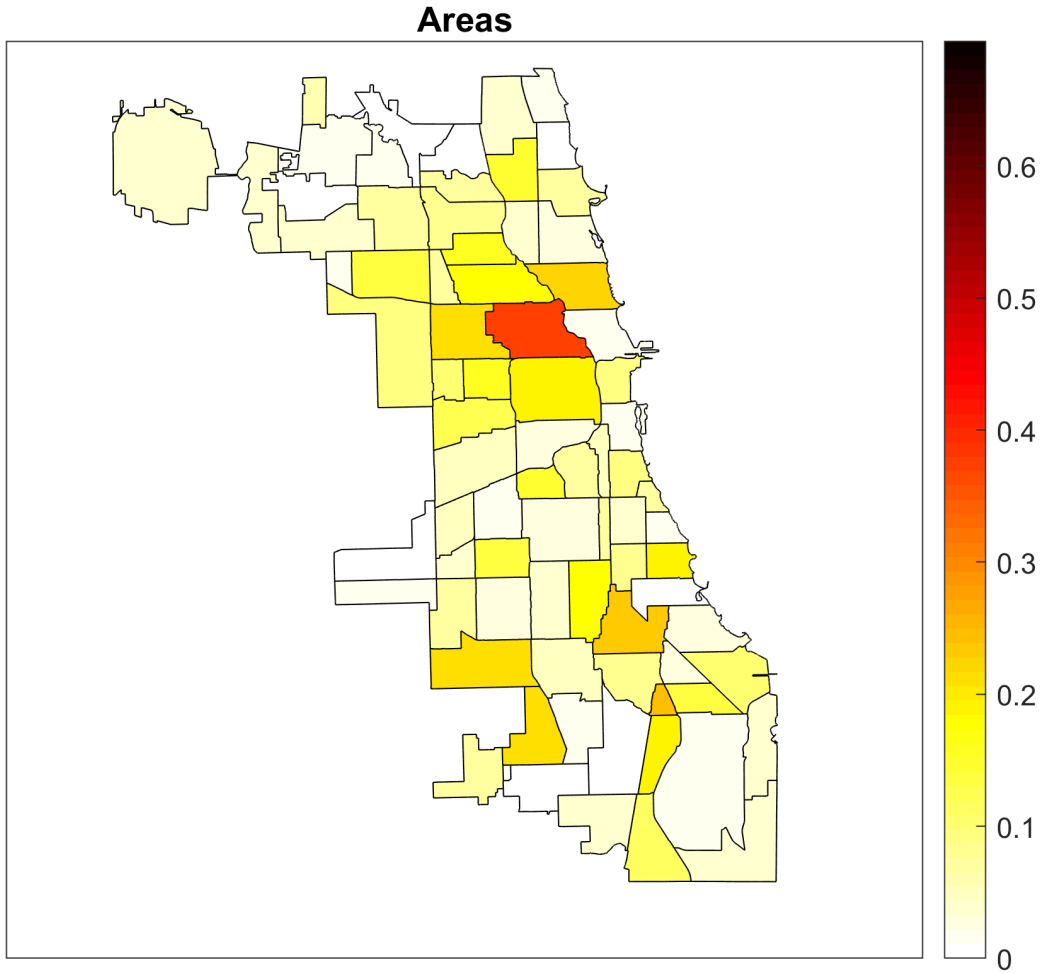

Areas
0.6
0.5
0.4
0.3
0.2
0.1
0

Component #3

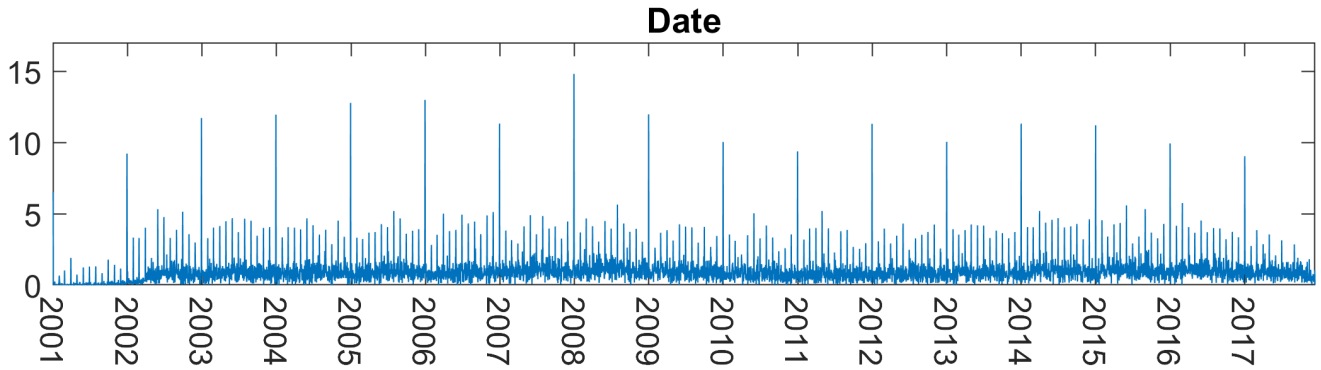

Date
15
10
5
0
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017

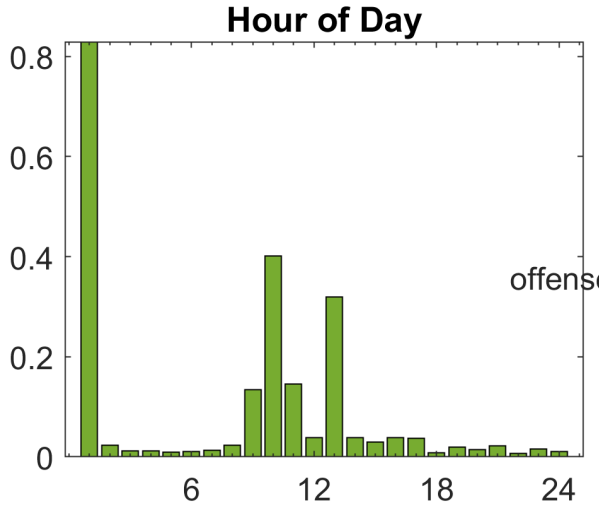

Hour of Day
0.8
0.6
0.4
0.2
0
6
12
18
24

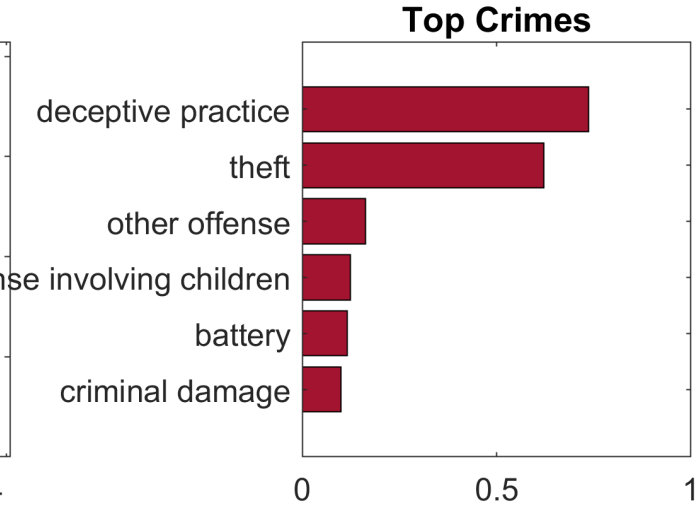

Top Crimes
battery
criminal damage
theft
robbery
assault
motor vehicle theft
0
0.5
1

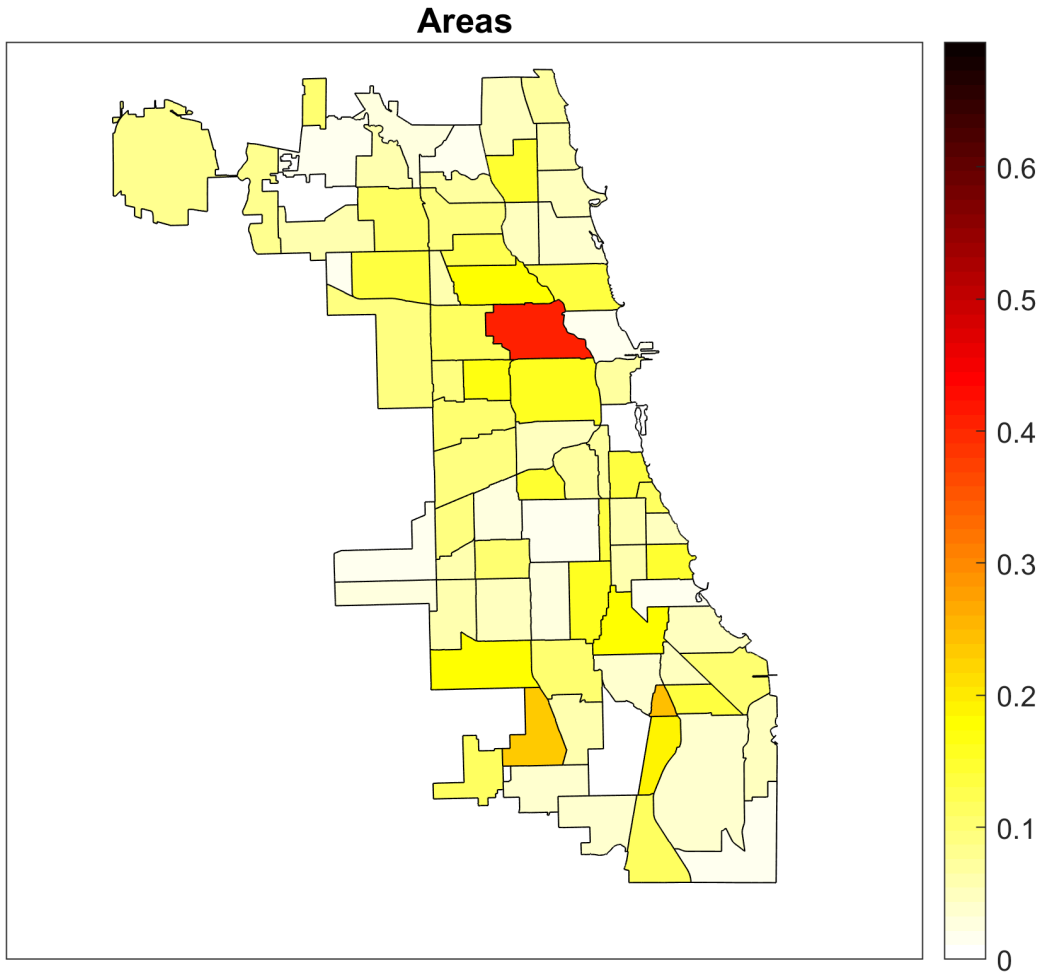

Areas
0.6
0.5
0.4
0.3
0.2
0.1
0

Component #6


Date
15
10
5
0
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017

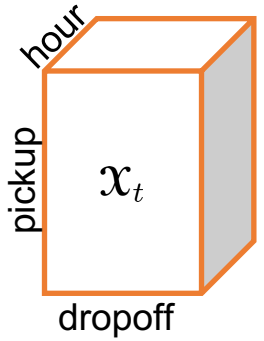

Hour of Day
0.8
0.6
0.4
0.2
0
6
12
18
24

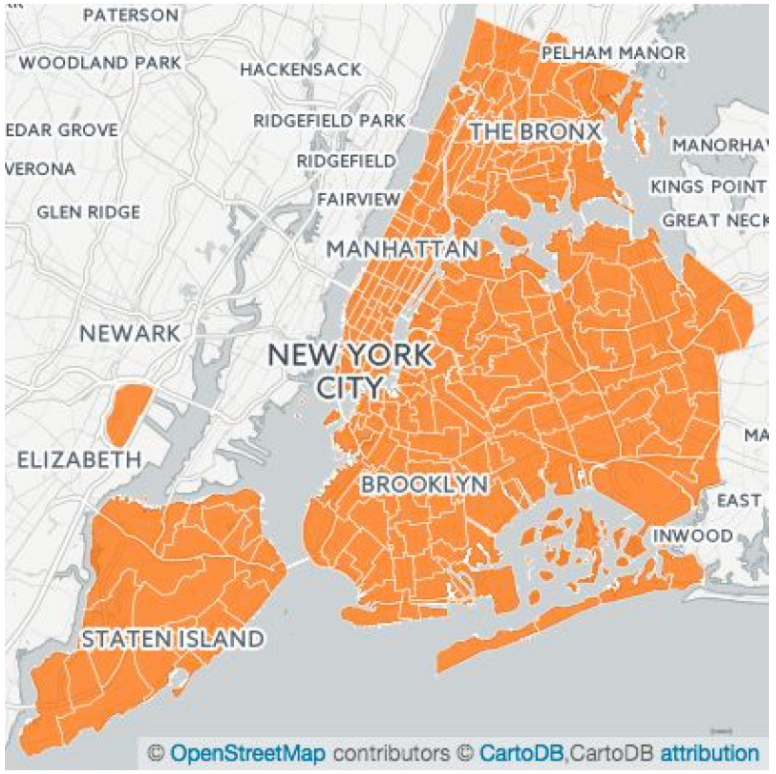

Top Crimes
deceptive practice
theft
other offense
offense involving children
battery
criminal damage
0
0.5
1


Areas
0.6
0.5
0.4
0.3
0.2
0.1
0

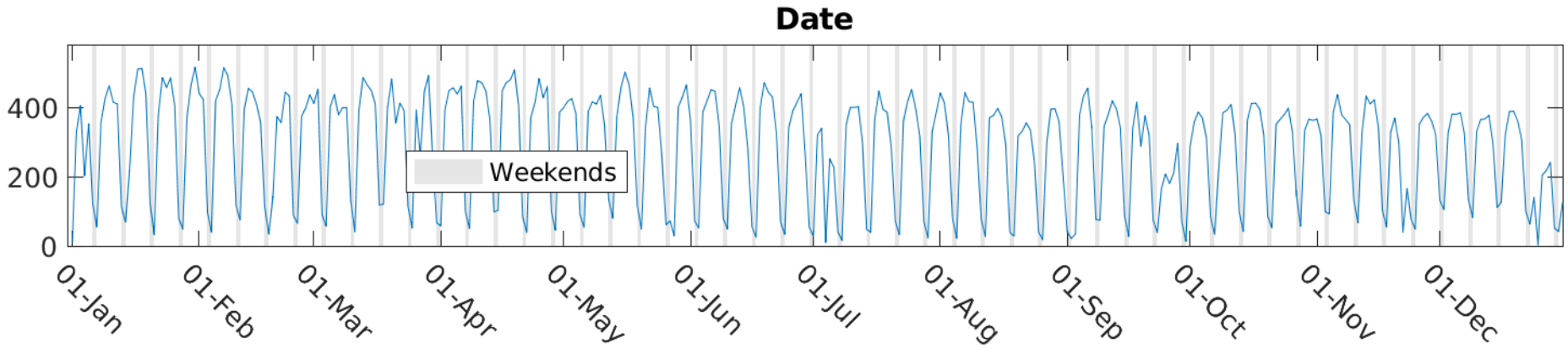
NYC Taxi Dataset – 4-way Tensor

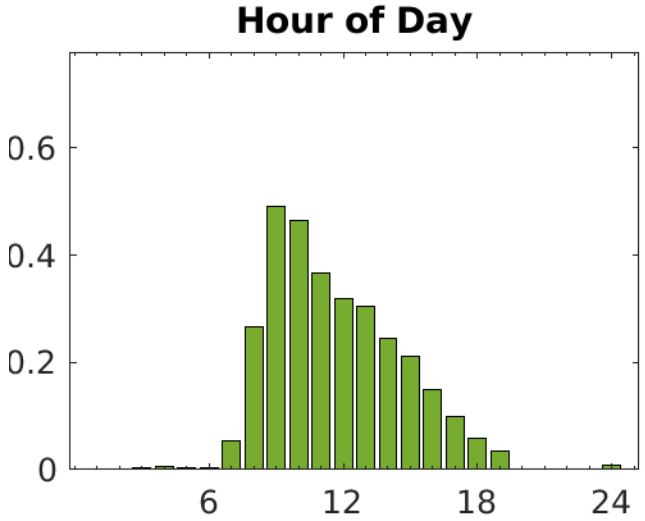
- **Data from NYC public records**
  https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page
- **10+ Years of Data**
- **4-way Tensor, Updated Daily**
  - Pickup Zone
  - Dropoff Zone
  - Pickup Hour
  - *New 3-way sparse tensor each day*
- **265 Taxi Zones**
  https://catalog.data.gov/dataset/nyc-taxi-zones

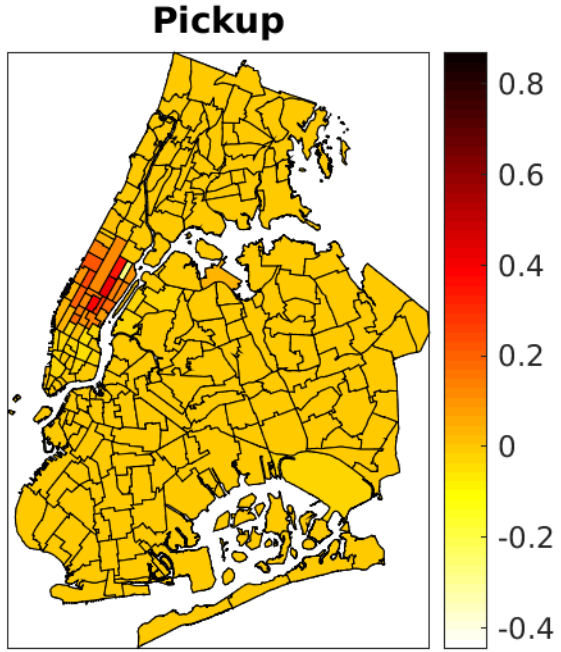


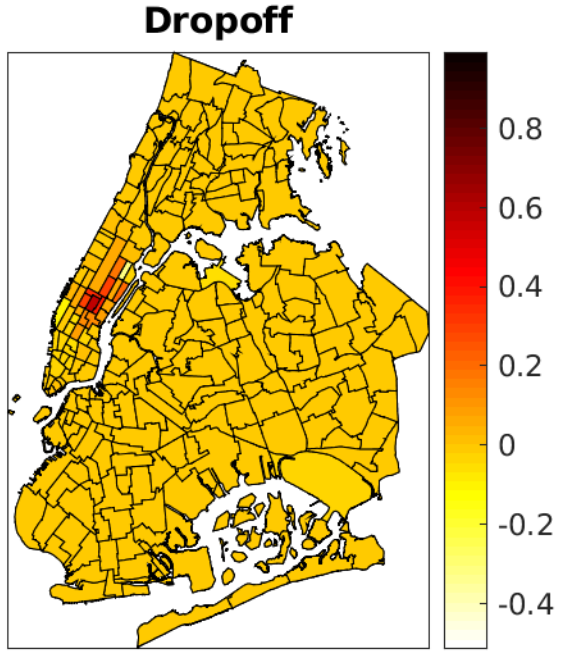


Component #1 (of 50) – Standard Mid-morning Weekday Traffic

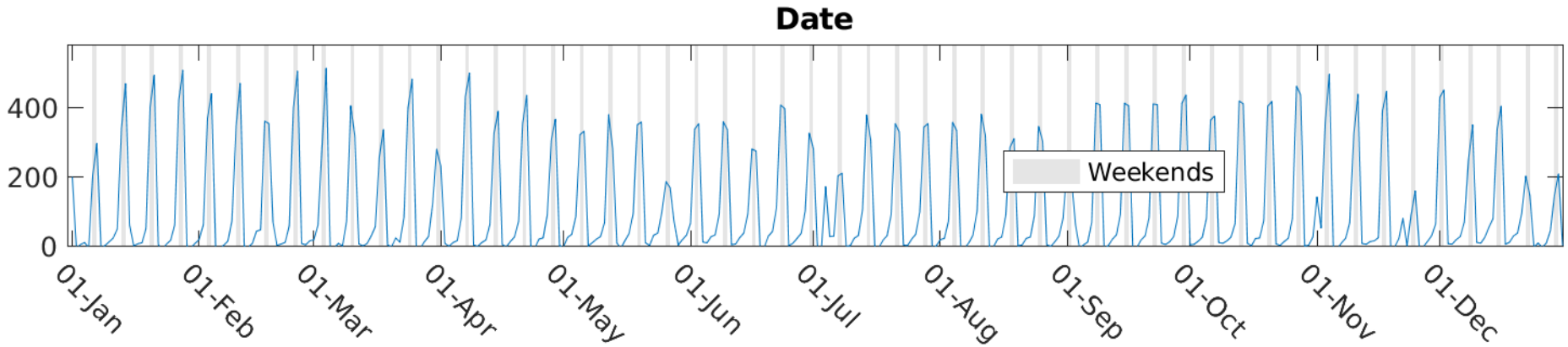


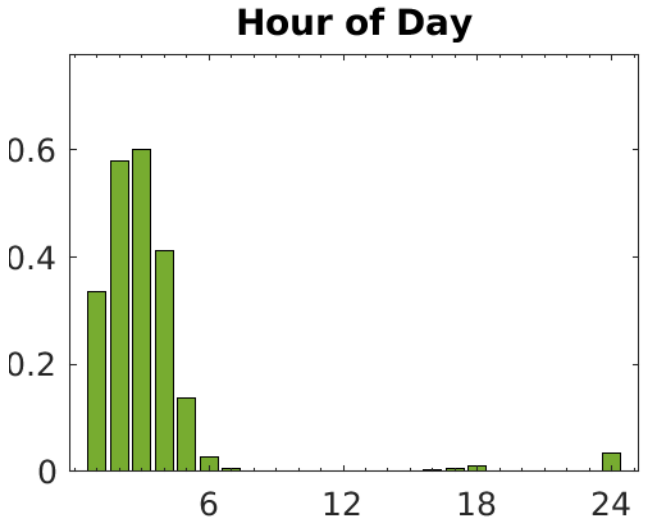


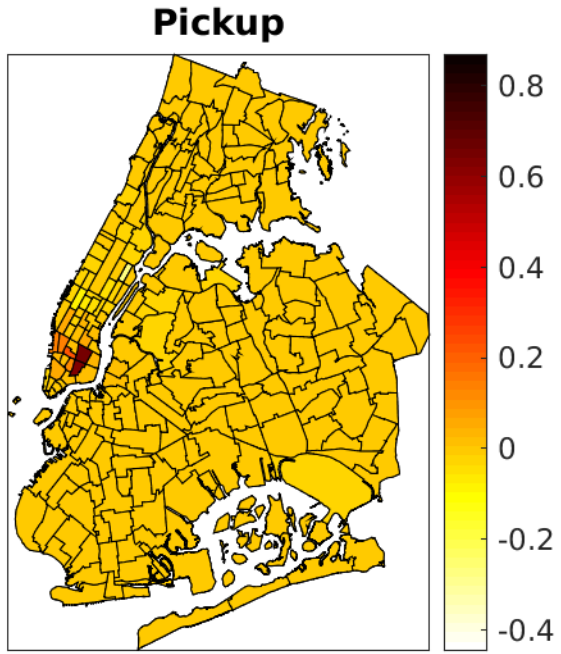


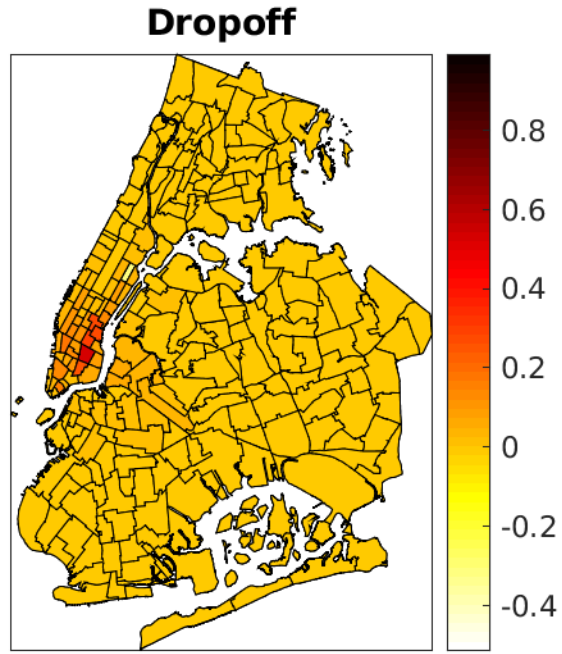


Component #21 (of 50): Weekend Nightlife

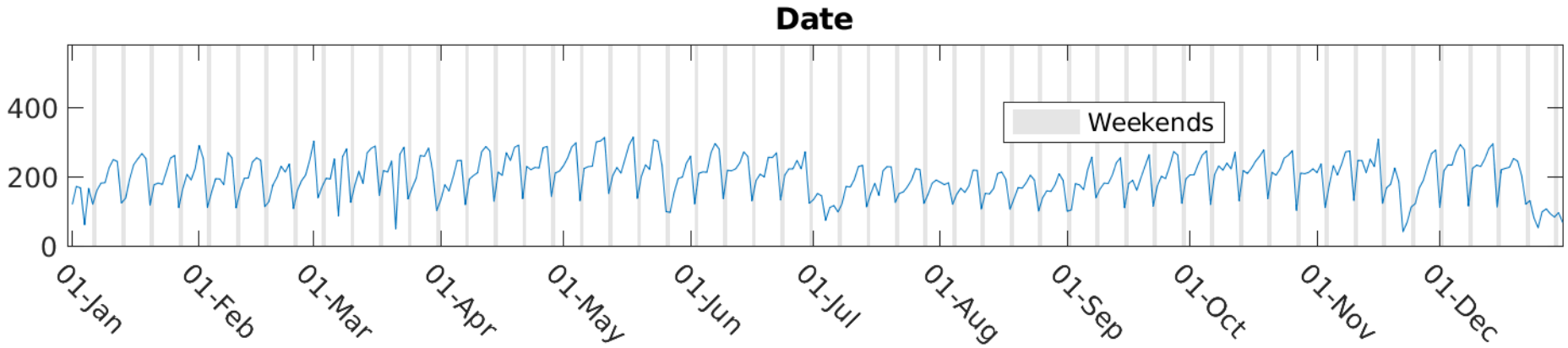


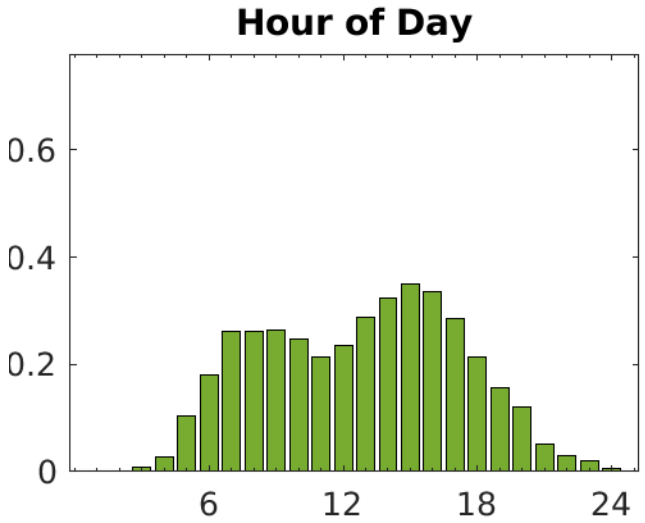


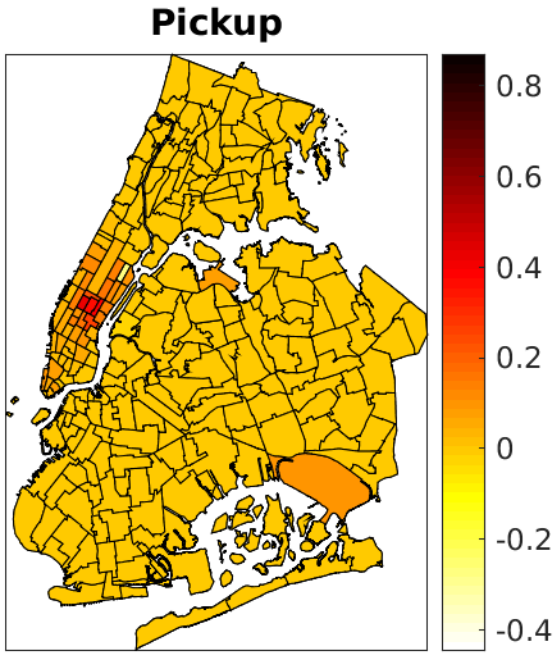


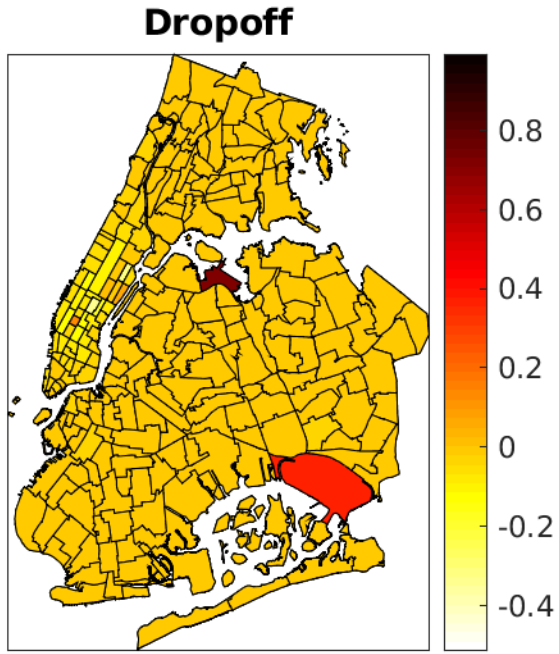


Component #17 (of 50): Travel to JFK and La Guardia Airports

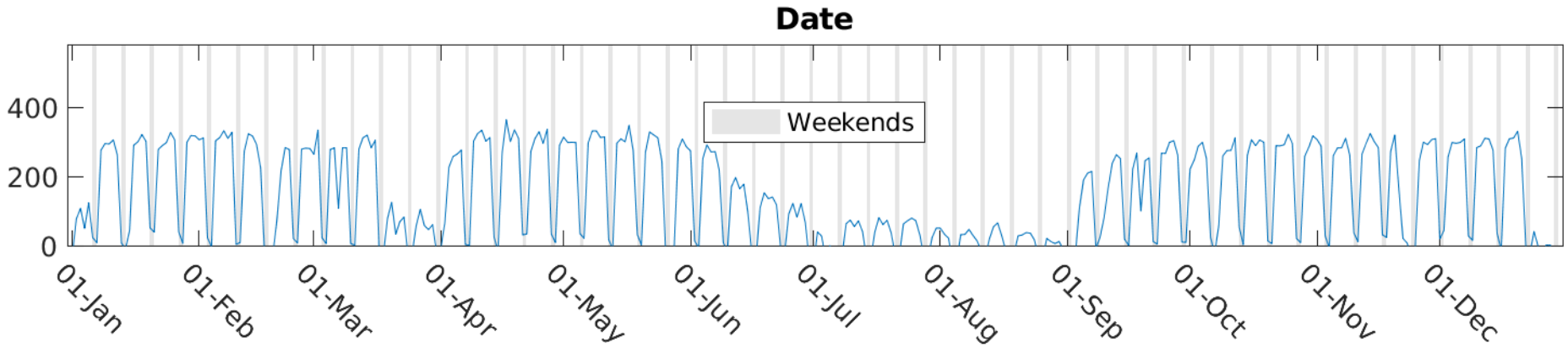


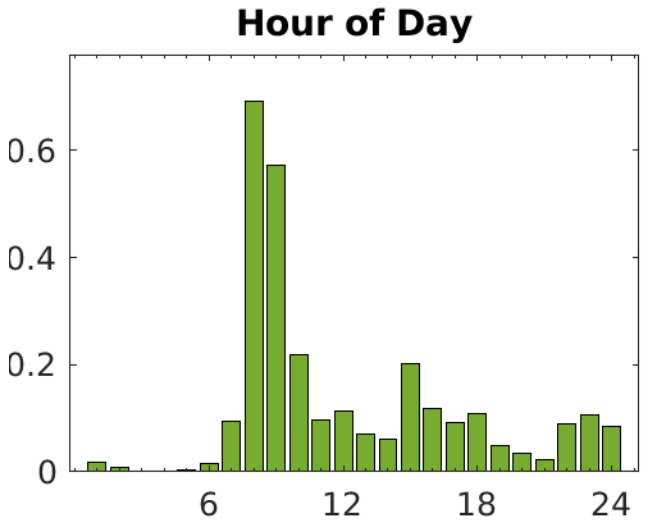


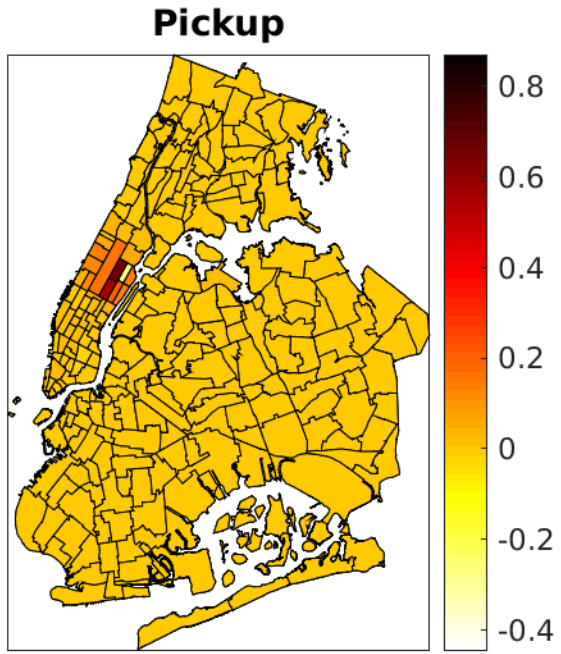


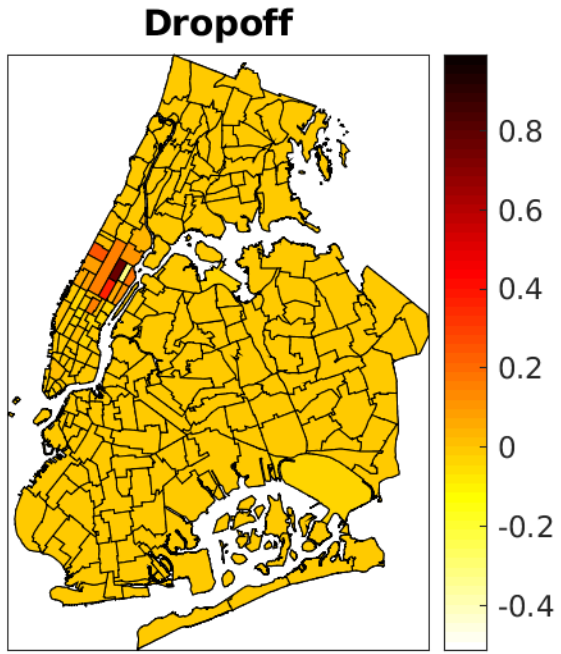


Component #20 (of 50): School Morning Dropoff

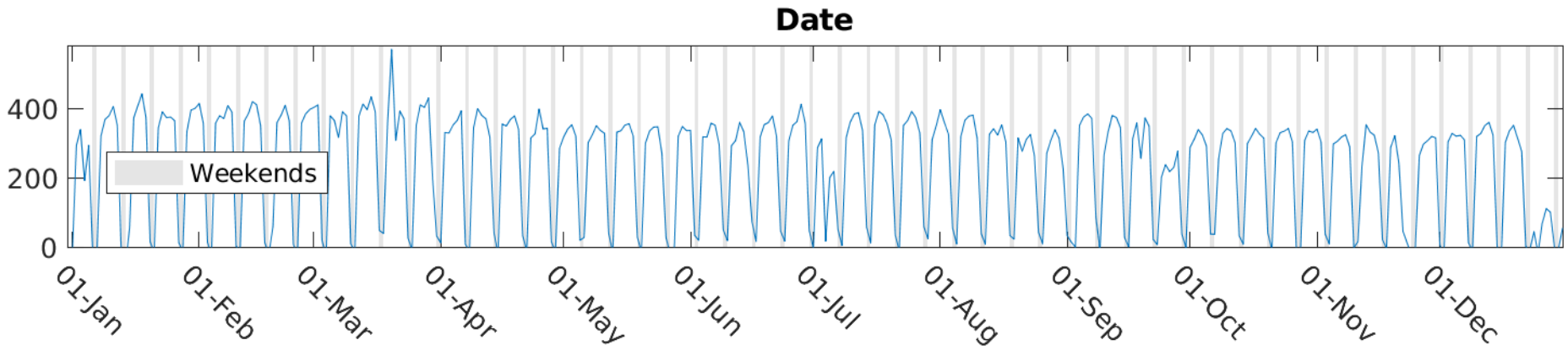


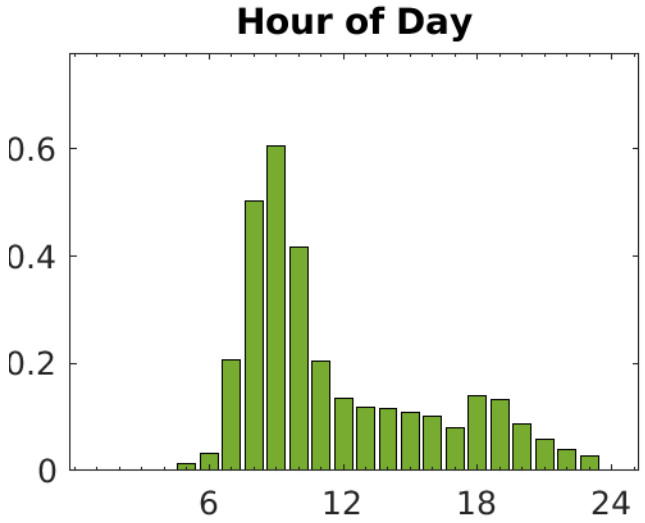


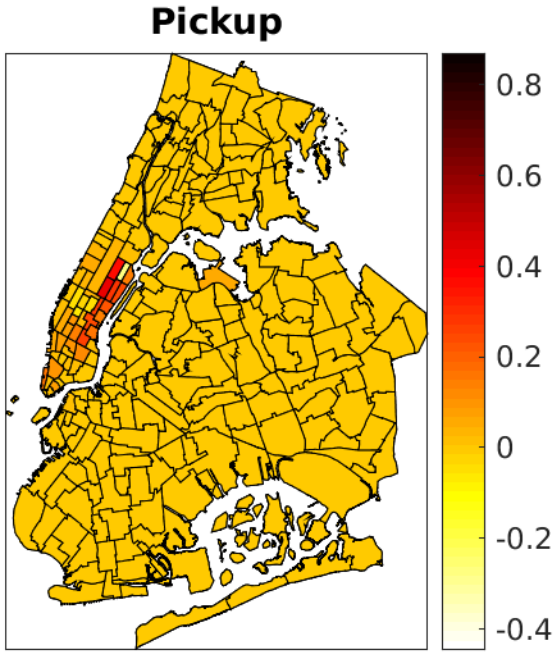


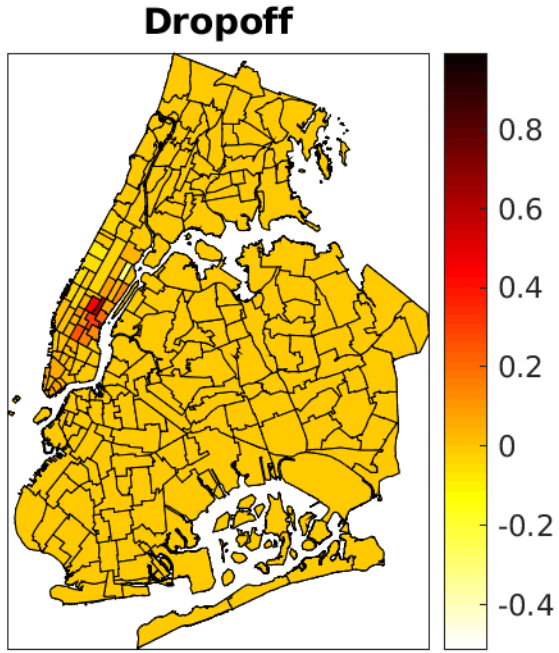


Component #4 (of 50): Morning Commute to Rockefeller Center





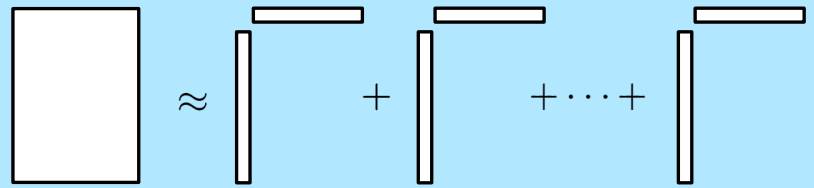




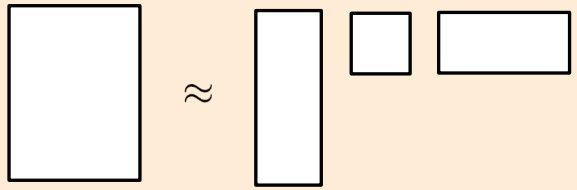
Low-Rank Decompositions: Two Points of View

Low-Rank Matrix Decompositions

Viewpoint 1: Sum of vector outer products, useful for interpretation



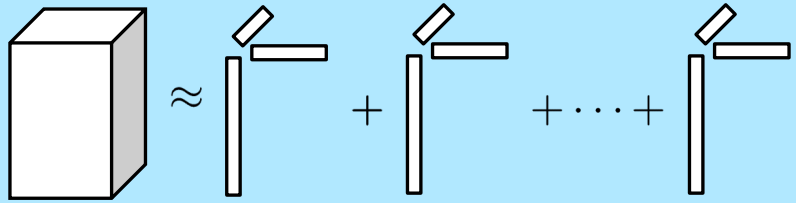
Viewpoint 2: High-variance subspaces, useful for compression



Singular value decomposition (SVD), eigendecomposition (EVD), nonnegative matrix factorization (NMF), etc.

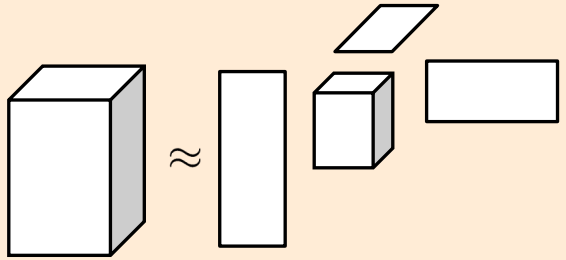
Low-Rank Tensor Decompositions

CP Model: Sum of d-way vector outer products, useful for interpretation



Canonical Polyadic, CANDECOMP, PARAFAC, CP

Tucker Model: Project onto high-variance subspaces to reduce dimensionality



HO-SVD, Best Rank-$(R_1, R_2, \ldots, R_d)$ decomposition

Other models for compression include hierarchical Tucker, tensor train, tensor ring, tensor network, etc.

Kolda and Bader (2009), Tensor Decompositions and Applications, https://doi.org/10.1137/07070111X

Why Not Just Use SVD/PCA?

Any tensor can be converted to a matrix (called matricization).

SVD/PCA gives us a low-rank model that:

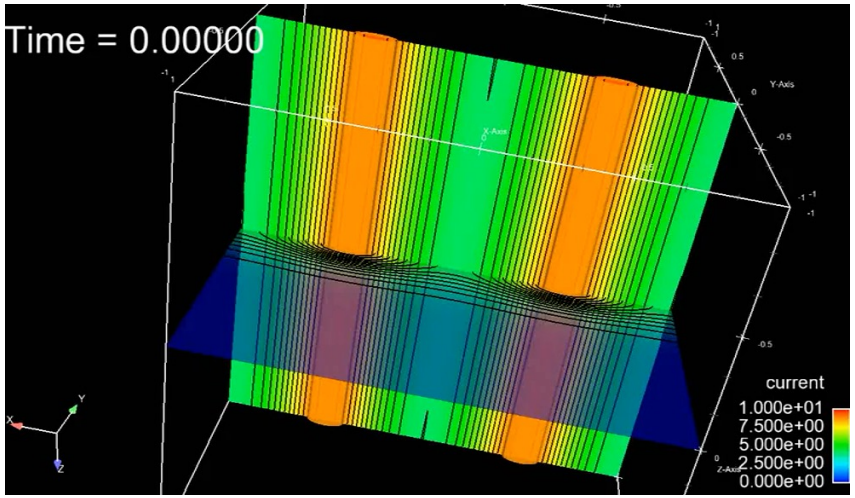
- Is easy to compute
- Gives us known, exact truncation error
- Is optimal

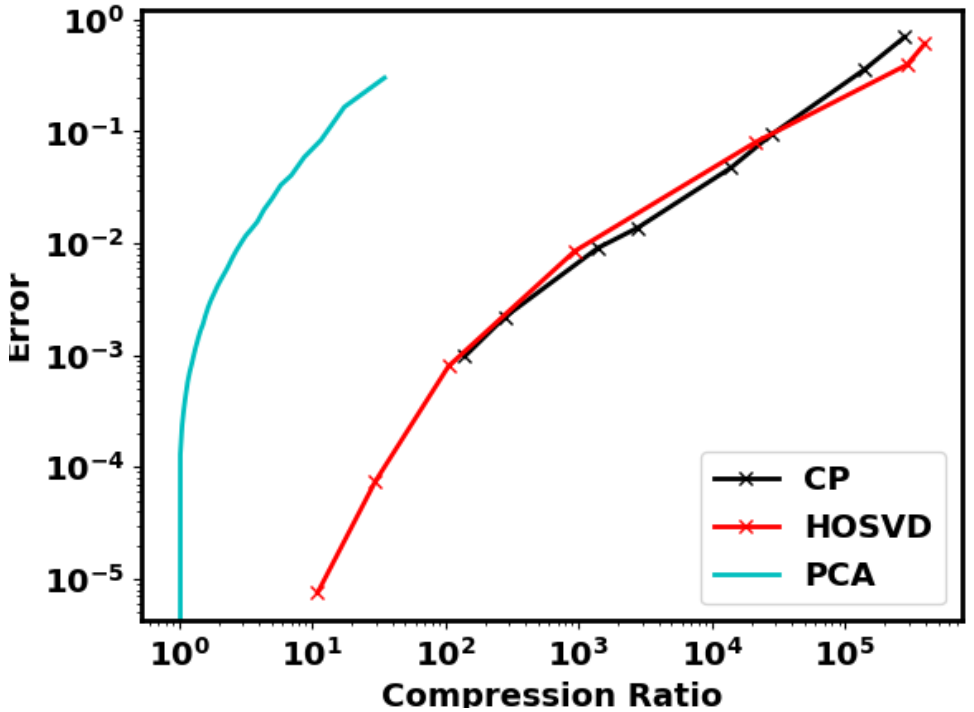
So why do tensor decomposition? Greater compression.

- Consider a $N_x \times N_y \times N_z \times N_v \times N_t$ tensor
- Typical approach to PCA is to compute truncated SVD of $N_x N_y N_z N_v \times N_t$ matrix, resulting in a low-rank model of size $R_{PCA} N_x N_y N_z N_v$
- CP decomposition: $R_{CP}(N_x + N_y + N_z + N_v + N_t)$
- Tucker decomposition: $R_x N_x + R_y N_y + R_z N_z + R_v N_v + R_t N_t + R_x R_y R_z R_v R_t$
- For given level of error ranks are different though

10-1000x more compression for same level of error!

3-D Island Coalescence Problem





Goal-Oriented Tensor Decompositions

Penalty formulation for constrained tensor decomposition:

- Data tensor $\mathcal{X}$
- Error metric, e.g., $f(\mathcal{X}, \mathcal{M}) = \|\mathcal{X} - \mathcal{M}\|_F^2$
- QoI/constraint functionals $G_q, q = 1, \ldots, N_q$
- Weighting coefficients $\alpha_q, q = 0, \ldots, N_q$

$$\min_{\mathcal{M}} \alpha_0 f(\mathcal{X}, \mathcal{M}) + \sum_{q=1}^{N_q} \alpha_q (G_q(\mathcal{X}) - G_q(\mathcal{M}))^2 \quad \text{s.t. } \mathcal{M} \text{ is of CP/Tucker form}$$

Solution methodology:

- First compute unconstrained solution using, e.g., CP-ALS or ST-HOSVD ($\alpha_0 = 1, \alpha_1, \ldots, \alpha_{N_q} = 0$)
- Choose weights α_q so that each objective function term is $\frac{1}{N_q+1}$ at initial guess
  - Each term contributes equally at initial guess and initial objective function is 1.0
- Solve minimization problem using quasi-Newton with limited-memory BFGS Hessian approximation, using unconstrained solution as initial guess
- Implemented using Matlab Tensor Toolbox with L-BFGS-B code by S. Becker
- Hand-coded gradients (can also us Matlab's automatic differentiation tools in the Deep Learning Toolbox)

Plasma Physics Example: 2-D Tearing Sheet Instability

Compressible Visco-resistive MHD:

- SNL Drekar (Shadid et al):

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0,$$
$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot [(\rho \mathbf{u} \otimes \mathbf{u}) + pI + \boldsymbol{\pi}] - \mathbf{j} \times \mathbf{B} = \mathbf{0},$$
$$\frac{n}{\gamma - 1}\frac{\partial T}{\partial t} + \frac{n}{\gamma - 1}\mathbf{u} \cdot \nabla T + p(\nabla \cdot \mathbf{u}) + \nabla \cdot \mathbf{q} - \eta \|\mathbf{j}\|^2 - \boldsymbol{\pi} : \nabla \mathbf{u} = 0,$$
$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \cdot \left[\mathbf{u} \otimes \mathbf{B} - \mathbf{B} \otimes \mathbf{u} - \frac{\eta}{\mu_0}(\nabla \mathbf{B} - (\nabla \mathbf{B})^T)\right] = \mathbf{0},$$
$$\nabla \cdot \mathbf{B} = 0,$$

Dense data tensor:

- 401 x 201 x 12 x 501
- 2-D cartesian spatial domain (401 x 201)
- 12 variables
- 501 time steps

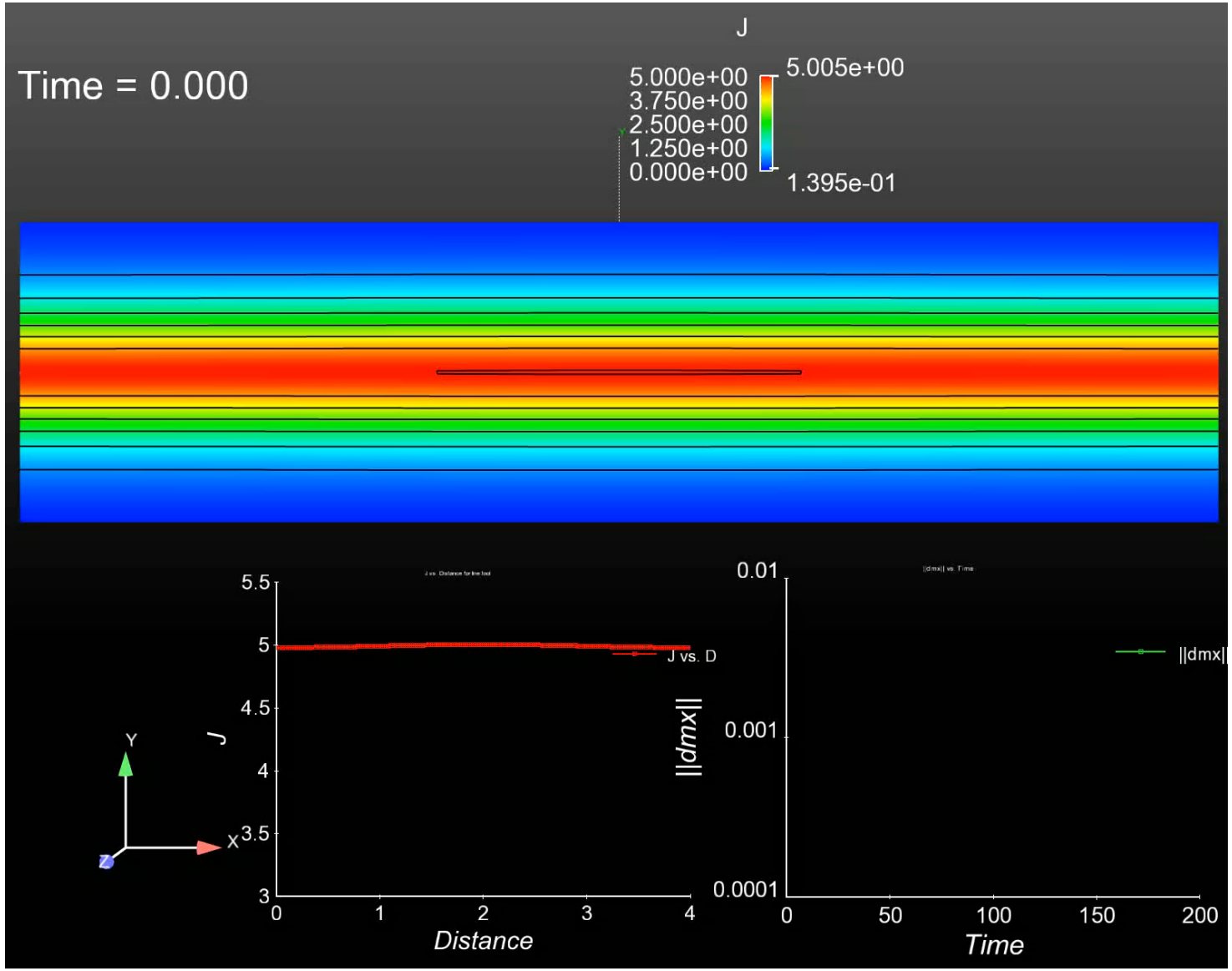
Several important QoIs:

- Momentum:

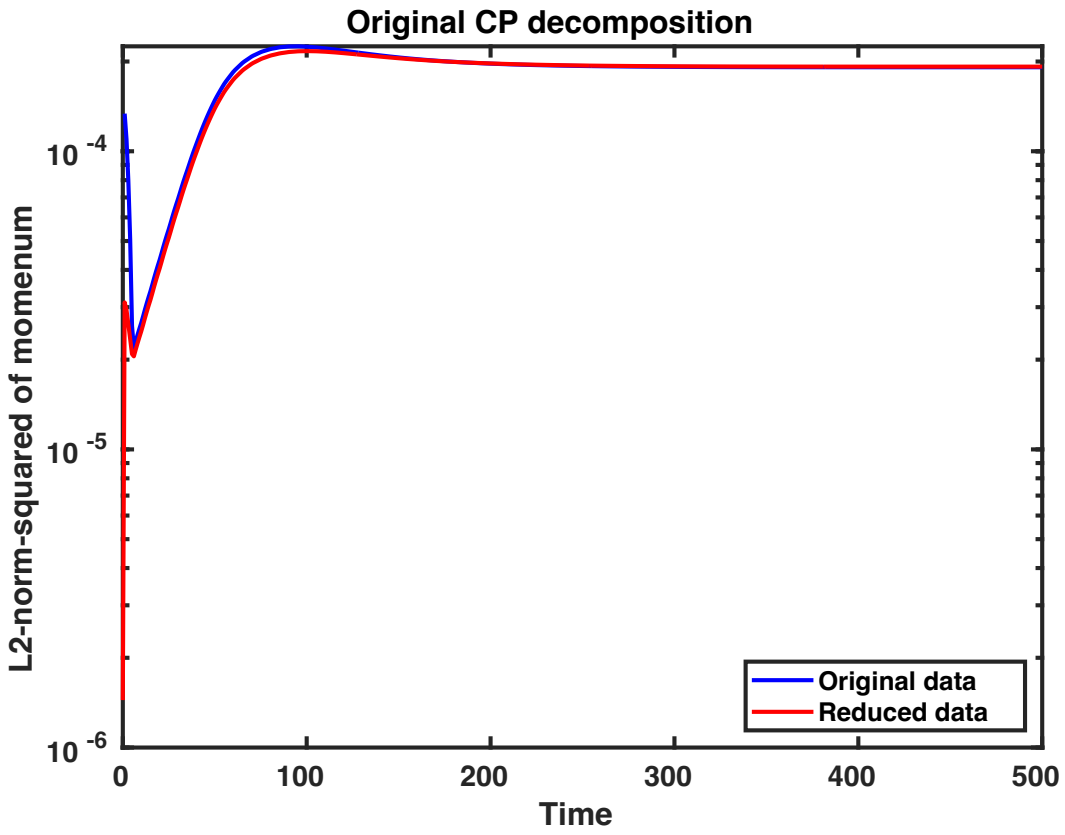
$$p = \left(\int_\Omega \|\rho \boldsymbol{v}\|^2 d\boldsymbol{x} \right)^{1/2}$$

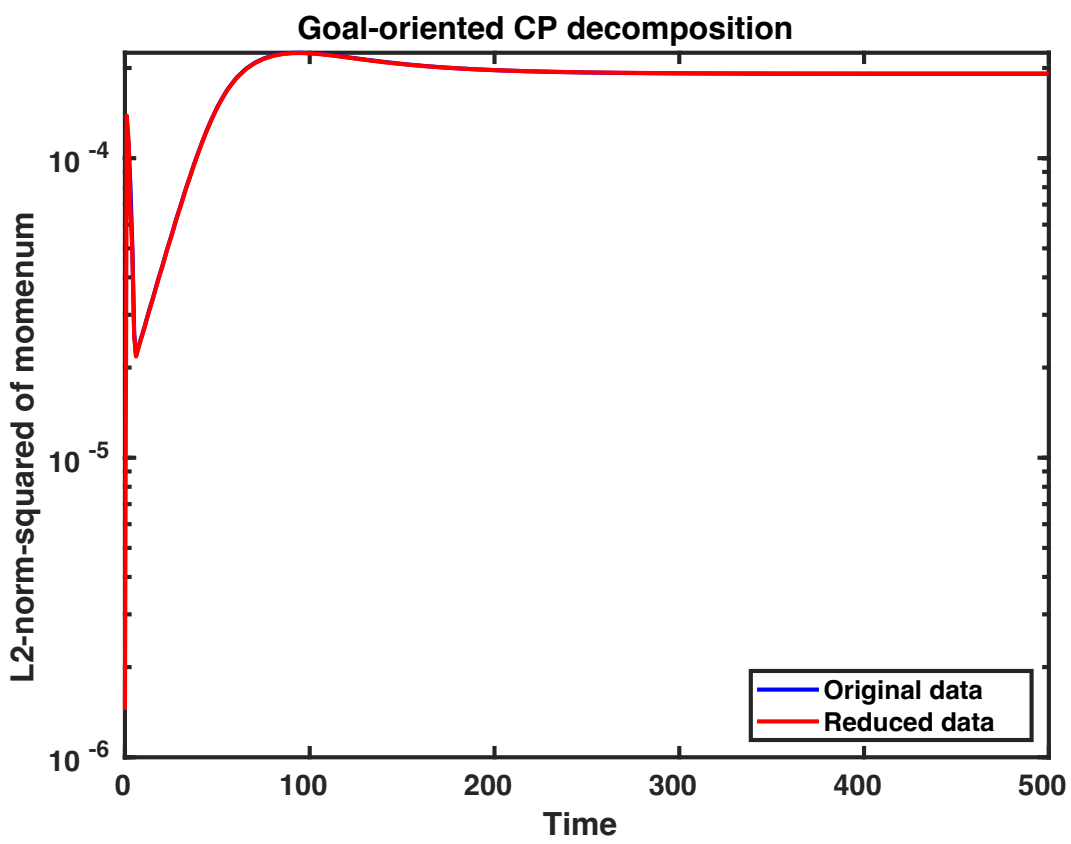
- Energy:

$$E_{total} = E_{internal} + E_{kinetic} + E_{magnetic}$$
$$= \int_\Omega \left(\rho C_v T + \frac{\|\rho \boldsymbol{v}\|^2}{2\rho} + \frac{\|\boldsymbol{B}\|^2}{2\mu_0} \right) d\boldsymbol{x}$$



2-D "Tearing Sheet" CP Decomposition: Momentum QoI

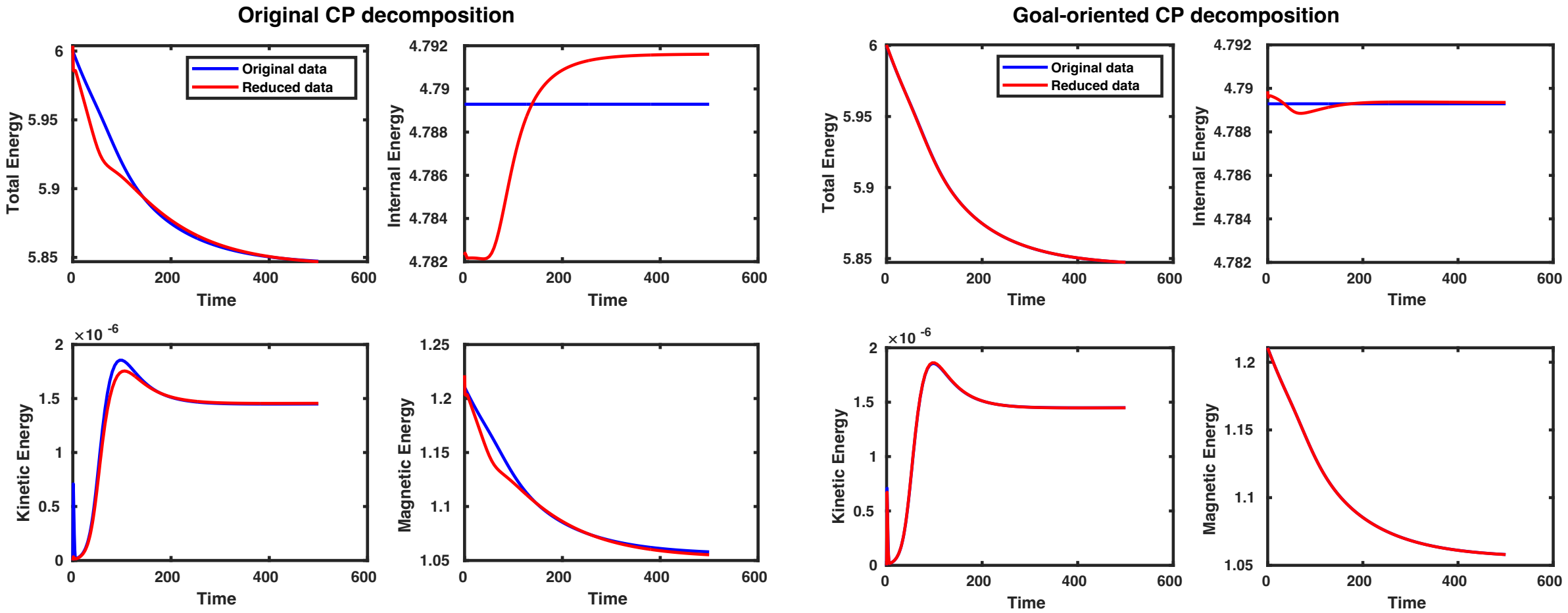




Rank = 15, initial fit = 0.9996, final fit = 0.9996

$$\text{fit} = 1 - \frac{\|\mathcal{X} - \mathcal{M}\|_F}{\|\mathcal{X}\|_F}$$

2-D "Tearing Sheet" CP Decomposition: Energy QoIs

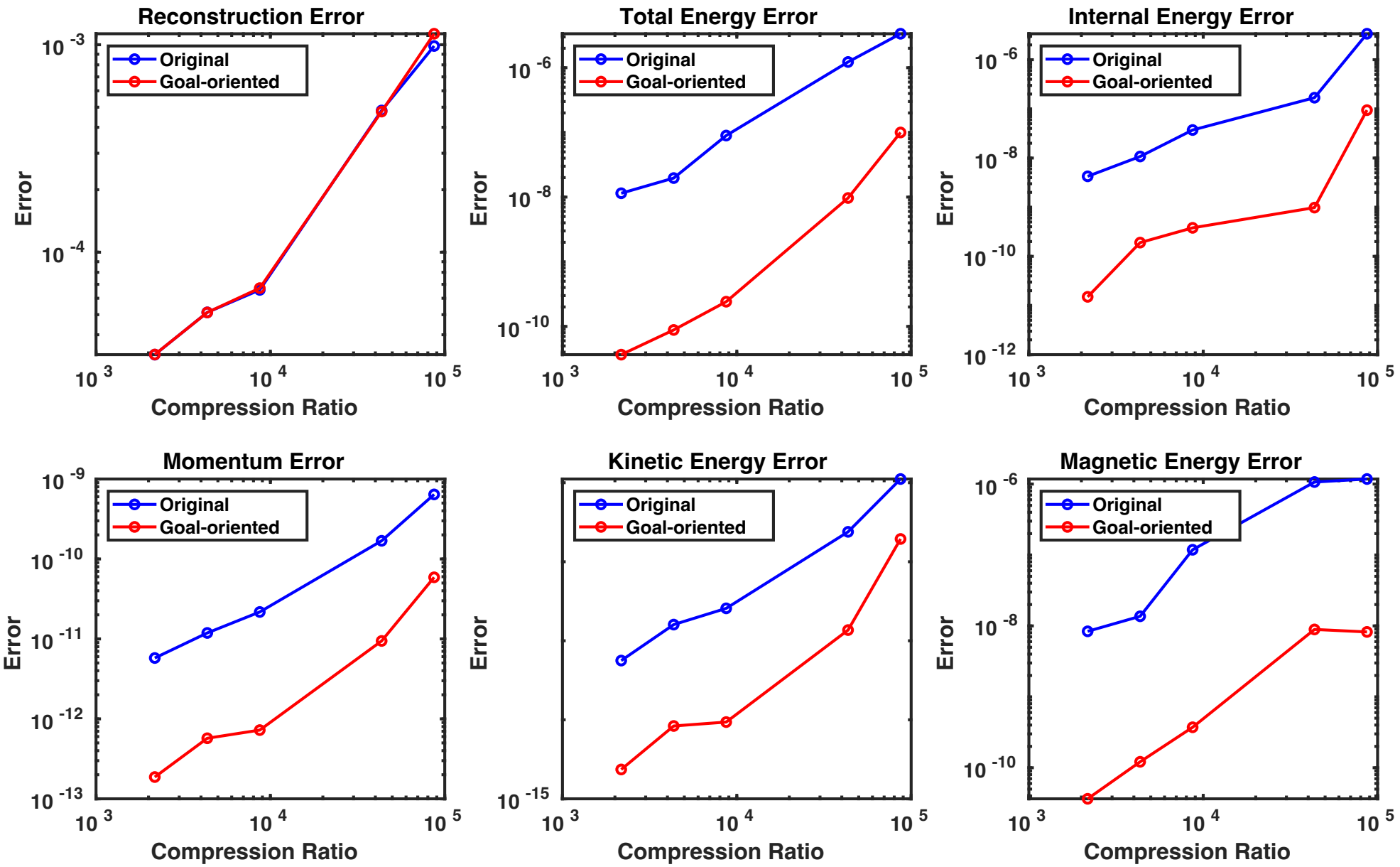


Not including total energy QoI in objective function for efficiency but is still improved

Rank = 15, initial fit = 0.9996, final fit = 0.9996

$$\text{fit} = 1 - \frac{\|\mathcal{X} - \mathcal{M}\|_F}{\|\mathcal{X}\|_F}$$

2-D "Tearing Sheet" CP Decomposition: Compression Ratios



Streaming Tensor Decompositions

Desire algorithms that can compute GCP decompositions as data streams in

Many formulations of this problem and assumptions on the type of data and processes generating it

Assumptions in this work:

- Updates are processed in discrete batches indexed by time $t = 1,2, \ldots$
- A complete *d*-way tensor is observed at each time step
- The dimensions of each update are fixed throughout time
- The CP rank is known and fixed
- The number of time steps is unbounded (infinite streaming)

Requires algorithm that can incrementally update the decomposition without revisiting old data

Development of streaming variant of GCP inspired by three prior works (for Gaussian loss):

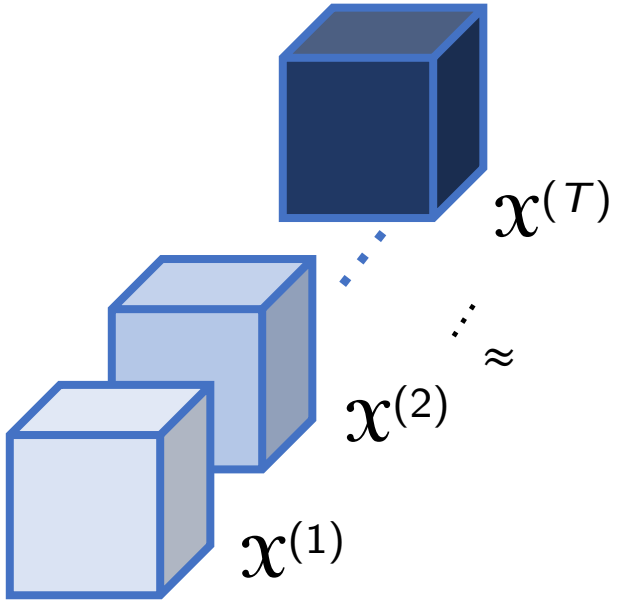
- Online SGD [Mardani et al, 2015] – gradient descent updates for factor matrices
- CP-Stream [Smith et al, 2018] – approximating old data by factor matrices from prior step

Honorable mention:

- OnlineCP [Zhou et al] – very fast method for incorporating prior time steps (but specific to Gaussian)

Streaming Tensors – Two Points-of-View

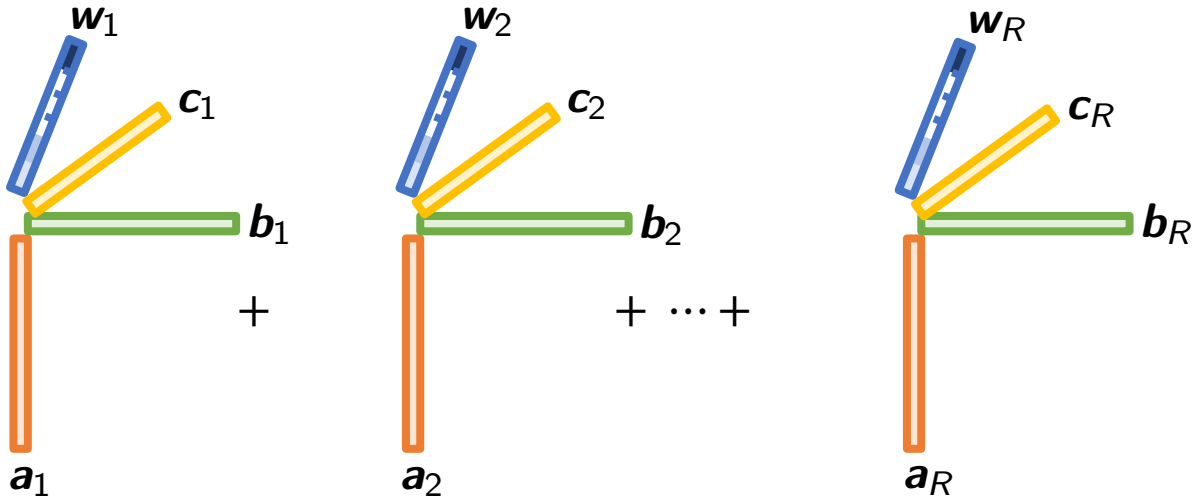
At each time step t, new 3-way hyperslice added



$$\mathcal{X} \in \mathbb{R}^{N_1 \times N_2 \times N_3 \times T}$$

$$\mathcal{X}^{(t)} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$$

If we assume factor matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ *fixed through time*, then the ideal factorization looks like…



$$\mathcal{X} \approx \sum_{j=1}^{R} \boldsymbol{a}_j \circ \boldsymbol{b}_j \circ \boldsymbol{c}_j \circ \boldsymbol{w}_j = [\![\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{W}]\!]$$

$$\boldsymbol{A} = [\boldsymbol{a}_1 \ \boldsymbol{a}_2 \ \ldots \ \boldsymbol{a}_R] \in \mathbb{R}^{N_1 \times R}$$

$$\boldsymbol{B} = [\boldsymbol{b}_1 \ \boldsymbol{b}_2 \ \ldots \ \boldsymbol{b}_R] \in \mathbb{R}^{N_2 \times R}$$

$$\boldsymbol{C} = [\boldsymbol{c}_1 \ \boldsymbol{c}_2 \ \ldots \ \boldsymbol{c}_R] \in \mathbb{R}^{N_3 \times R}$$

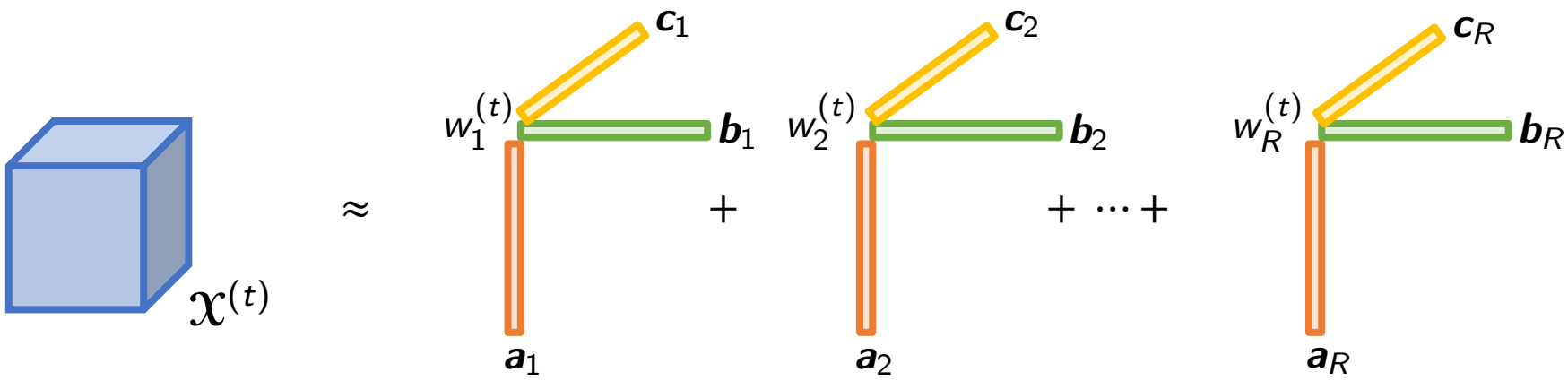
$$\boldsymbol{W} = [\boldsymbol{w}_1 \ \boldsymbol{w}_2 \ \ldots \ \boldsymbol{w}_R] \in \mathbb{R}^{T \times R}$$

Generic d-way setup is similar.

Streaming Tensors – Two Points-of-View

At each time step t, new 3-way hyperslice added

If we assume factor matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ *fixed through time*, then the ideal factorization looks like…



$$\mathcal{X}^{(t)} \approx \sum_{j=1}^{R} w_j^{(t)} \boldsymbol{a}_j \circ \boldsymbol{b}_j \circ \boldsymbol{c}_j = [\![\boldsymbol{w}^{(t)}; \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!]$$

$$\boldsymbol{A} = [\boldsymbol{a}_1 \ \boldsymbol{a}_2 \ \ldots \ \boldsymbol{a}_R] \in \mathbb{R}^{N_1 \times R}$$

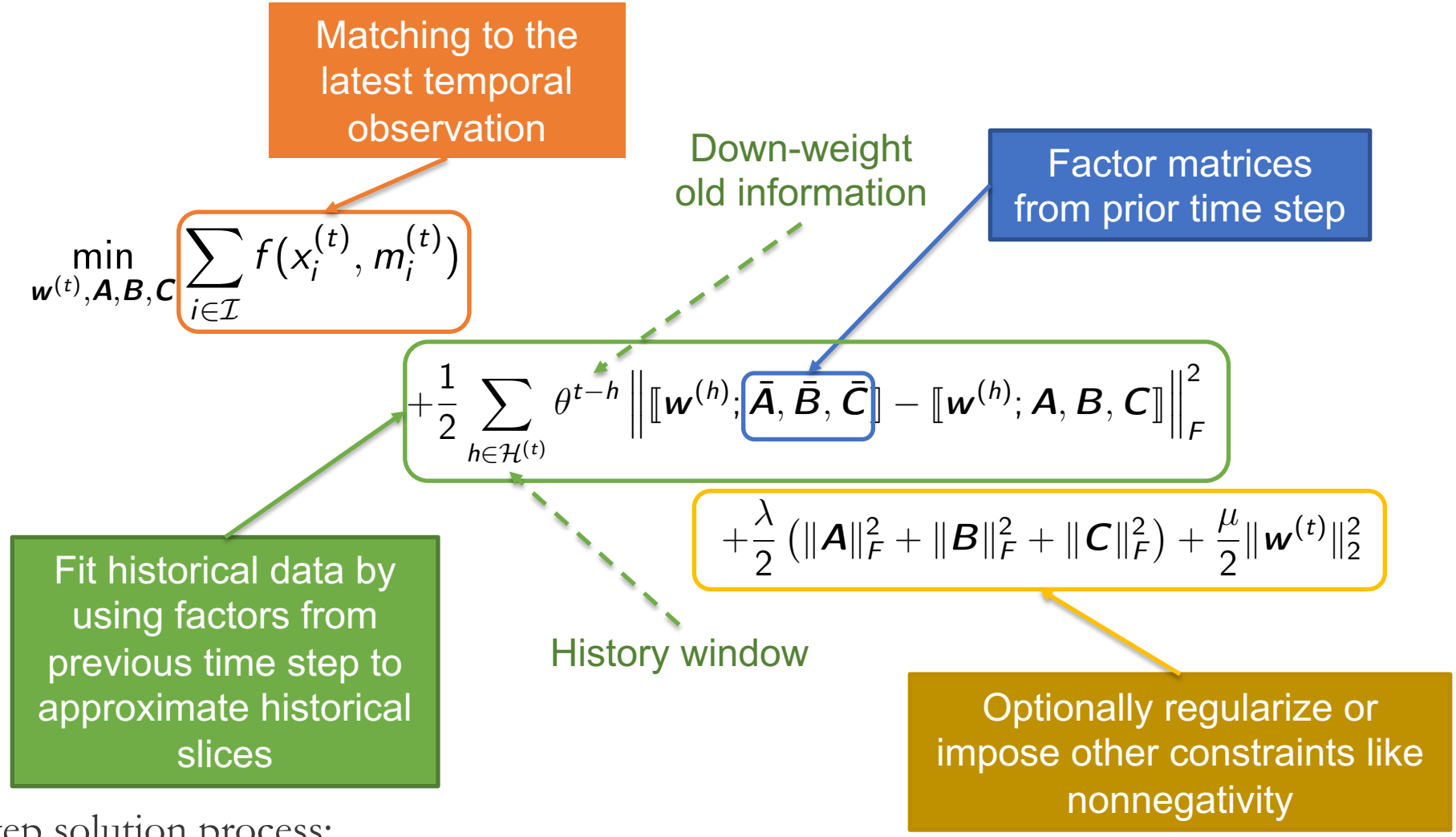
$$\boldsymbol{B} = [\boldsymbol{b}_1 \ \boldsymbol{b}_2 \ \ldots \ \boldsymbol{b}_R] \in \mathbb{R}^{N_2 \times R}$$

$$\boldsymbol{C} = [\boldsymbol{c}_1 \ \boldsymbol{c}_2 \ \ldots \ \boldsymbol{c}_R] \in \mathbb{R}^{N_3 \times R}$$

$$\mathcal{X}^{(t)} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$$

Generic d-way setup is similar.

Streaming GCP "OnlineGCP" Formulation*



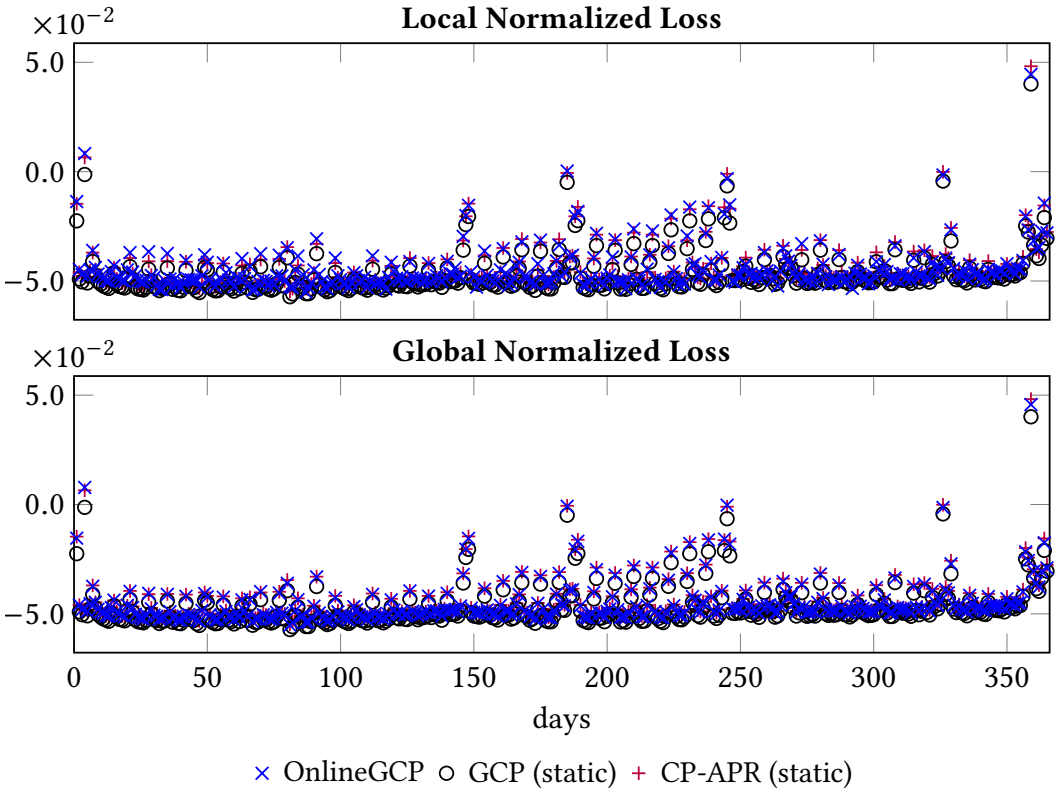
Two step solution process:

- Solve for temporal weights $\boldsymbol{w}^{(t)}$ with factor matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ held fixed using GCP-SGD
- Solve for updated factor matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ using GCP-SGD

*Phipps, Johnson, Kolda, Streaming Generalized Canonical Polyadic Tensor Decompositions, In PASC'23, 2023.

Real Data Experiments (Poisson)

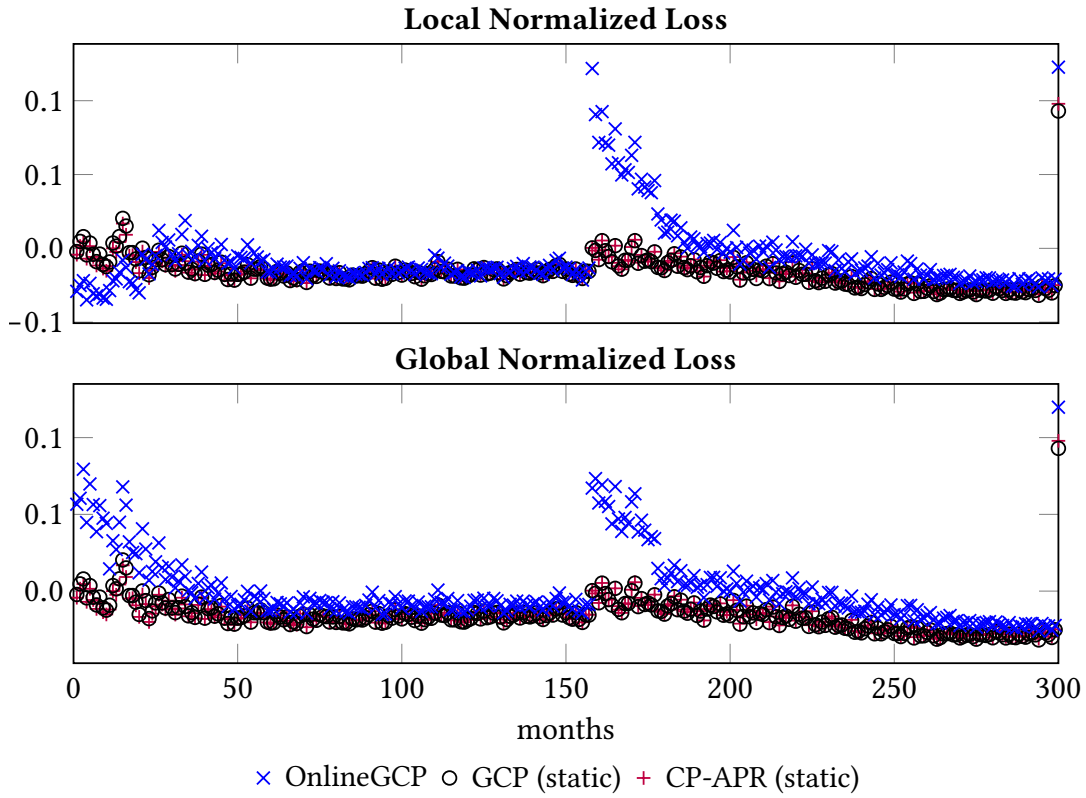
NYC Taxicab



Four-way tensor of counts of taxi rides in NYC in 2018

- Pickup zone (263)
- Dropoff zone (263)
- Pickup hour (24)
- Day (265)
- 3.8% sparsity

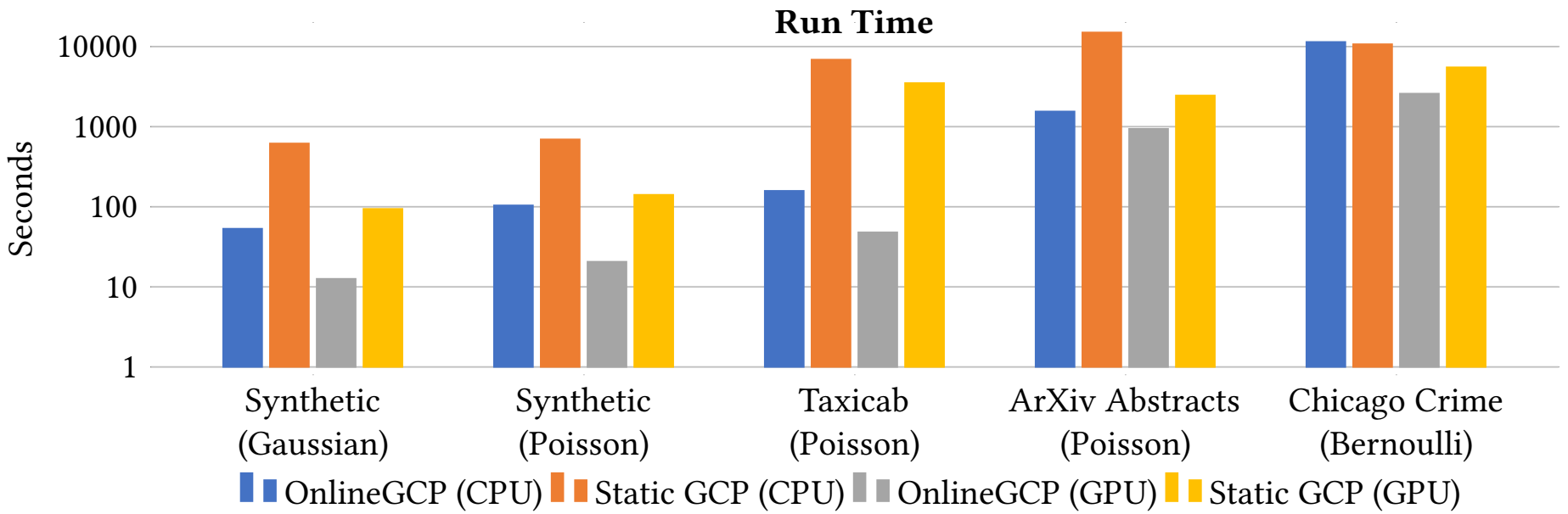
ArXiv Abstracts



Three-way tensor of counts of words in abstracts published on arxiv.org

- Category (172)
- Word (24558)
- Month since 10 years after first abstract (300)
- 2.4% sparsity

Performance Portability with Kokkos*



CPU:
- Dual-socket Intel Xeon Platinum 8260, 24 cores/socket
- OpenMP Kokkos backend

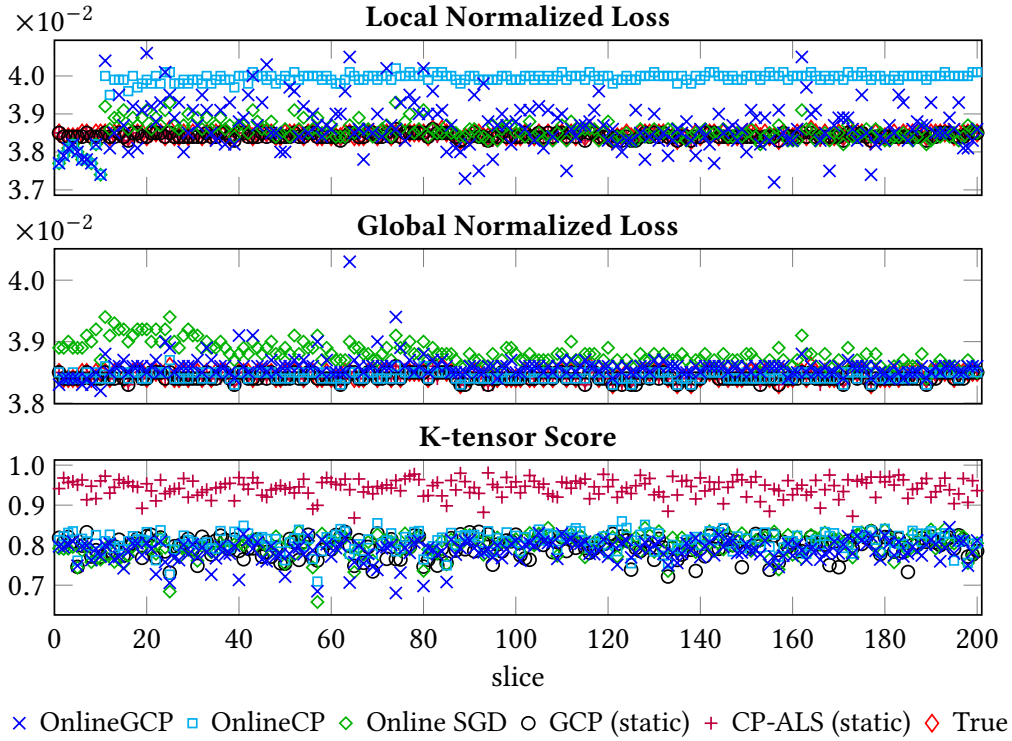
GPU:
- NVIDIA Volta V100
- CUDA Kokkos backend

First (and I believe only), performance-portable streaming CP decomposition tool!

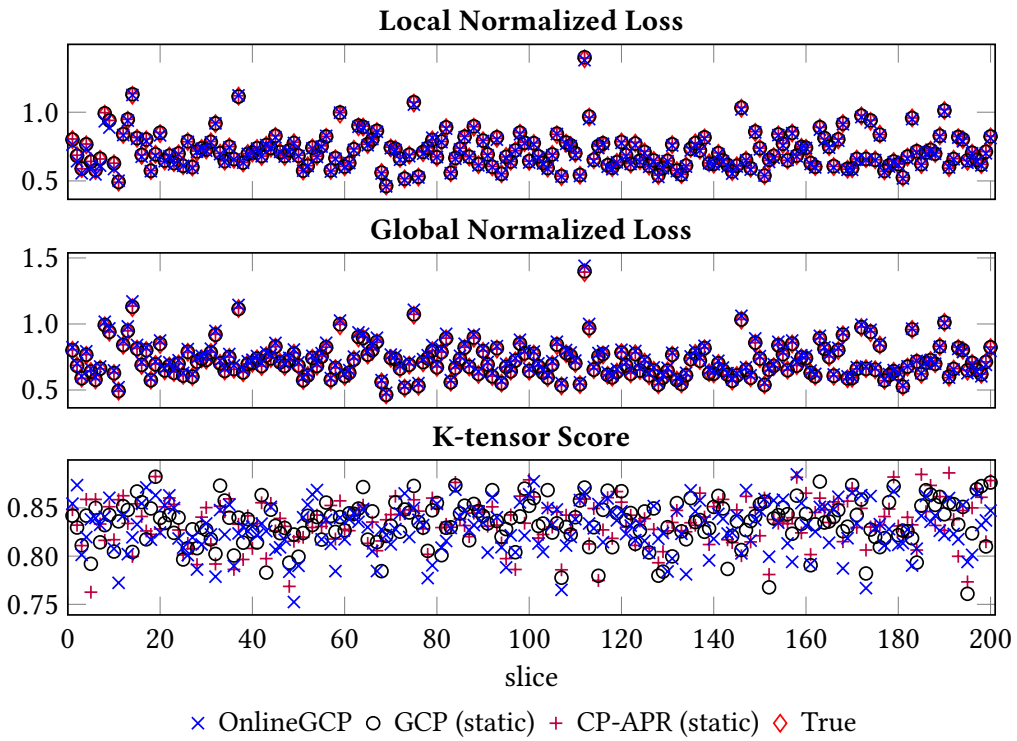
*Phipps, Johnson, Kolda, Streaming Generalized Canonical Polyadic Tensor Decompositions, In PASC'23, 2023.

Synthetic Data Experiments

Gaussian 300x300x200, dense, R=20



Poisson 300x300x200, 3.2% sparsity, R=20



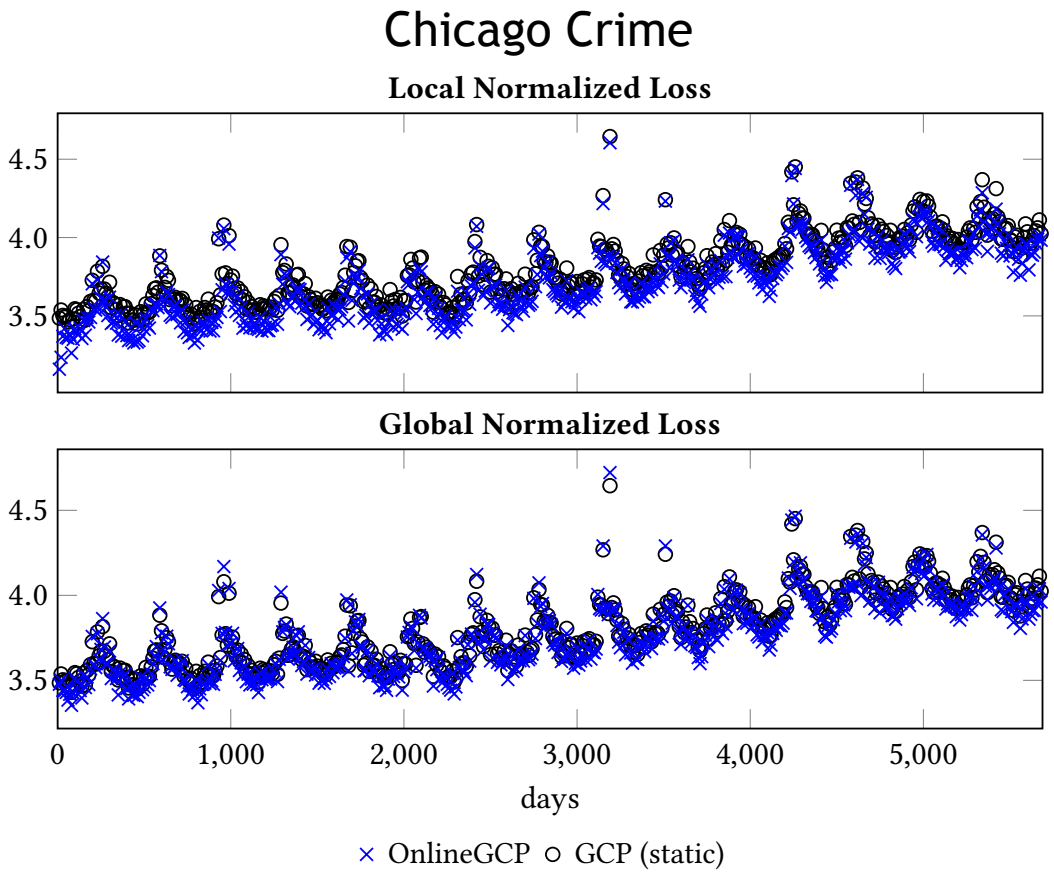
Local Normalized Loss: $F_{local}(\mathcal{X}^{(t)}, \mathcal{M}^{(t)}) = \frac{1}{\|\mathcal{X}^{(t)}\|_F^2} \sum_{i \in \mathcal{I}} f(x_i^{(t)}, m_i^{(t)}),; \ \mathcal{M}^{(t)} = [\![\boldsymbol{w}^{(t)}; \boldsymbol{A}_1^{(t)}, \ldots, \boldsymbol{A}_d^{(t)}]\!]$

Global Normalized Loss: $F_{global}(\mathcal{X}^{(t)}, \tilde{\mathcal{M}}^{(t)}) = \frac{1}{\|\mathcal{X}^{(t)}\|_F^2} \sum_{i \in \mathcal{I}} f(x_i^{(t)}, \tilde{m}_i^{(t)}),; \ \tilde{\mathcal{M}}^{(t)} = [\![\boldsymbol{w}^{(t)}; \boldsymbol{A}_1^{(T)}, \ldots, \boldsymbol{A}_d^{(T)}]\!]$

Real Data Experiments (Bernoulli)

Four-way tensor of counts of crimes in the city of Chicago

- Hour of the day (24)
- Crime type (77)
- Neighborhood (32)
- Day since start date (5687)
- 1.6% sparsity
- Most counts are just 1, so use Bernoulli instead of Poisson



Statistical Opportunities for Locality in Tensor Computations (Solstice)

DOD SPP, ~$540K/yr dropping to ~300K/yr FY24-

- Started in FY19 as follow-on funding from GCP LDRD, originally led by Tammy Kolda
- Collaboration with Prof. Rich Vuduc at Georgia Tech and many SNL contributors

Motivation

- Acquisition of tensor data of interest to sponsor has geographic scope from a variety of sensors

Approach

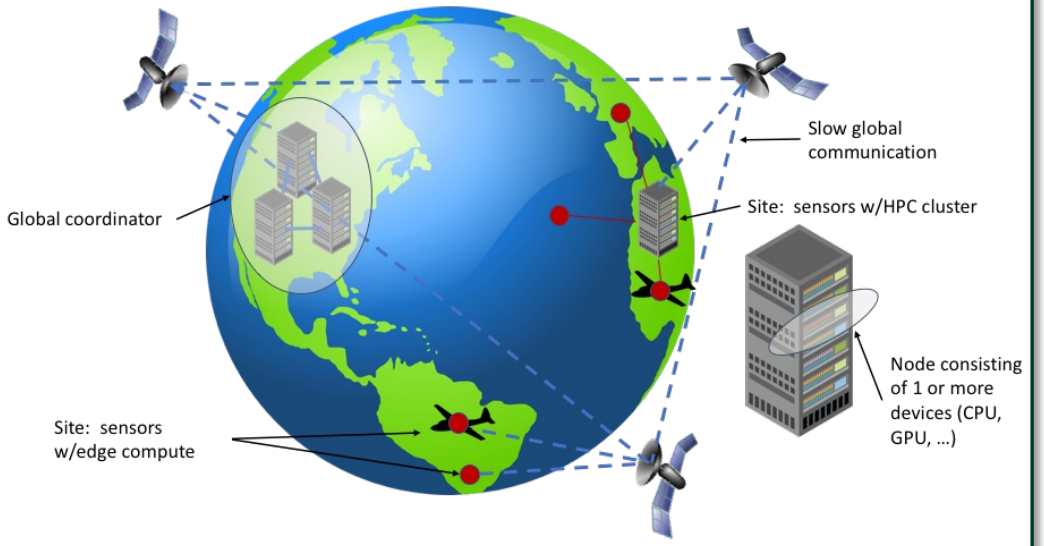
- GCP decompositions exploiting asynchronous parallelism
- Randomized and streaming algorithms
- HPC algorithm development and software delivery

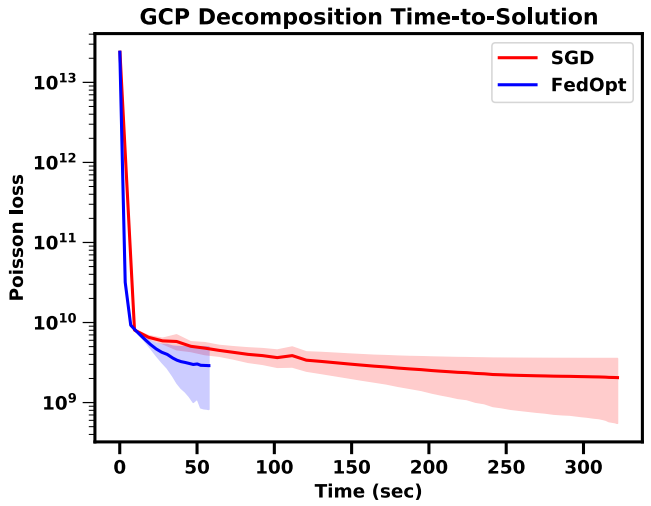
Potential Impact

- Real-time analysis of geographically distributed data

Accomplishments

- Developed federated learning algorithm for GCP decompositions capable of leveraging both synchronous and asynchronous distributed parallelism[1]
- Developed streaming GCP decomposition algorithm that can produce GCP factors with a single pass over the data (suitable for real-time computation)[2]
- Developed GCP decomposition approaches in the Chapel programming environment important to the sponsor





Top: *Diagram depicting conceptual framework for geographically distributed tensor decomposition.* ***Right:*** *Time-to-solution for GCP decomposition of a cyber security-related data set on 640 cores, comparing synchronous stochastic gradient descent (SGD) and asynchronous federated learning (FedOpt), demonstrating > 6x speedup.*

[1]Lewis and Phipps, Low-Communication Asynchronous Distributed Generalized Canonical Polyadic Tensor Decomposition in Proceedings of HPEC, 2021.
[2]Phipps, Johnson, Kolda, Streaming Generalized Canonical Polyadic Tensor Decompositions, in Proceedings of PASC, 2023.

Python Interoperability

GenTen provides python wrappers through pybind11 called pygenten
- GenTen tensor classes and decomposition routines callable from within python
- Thanks to Kim Liegeois for prototyping this capability!
- Works with all Kokkos backends supported by GenTen (OpenMP, CUDA, HIP, SYCL) and MPI
- Provides limited facilities for data manipulation and I/O

Integrates with the Python Tensor Toolbox (pyttb)
- GenTen decomposition routines accept pyttb tensors as input and return pyttb decomposition results
- Supports passing data back and forth without copies on the CPU
- Tensor data copied from host to device and results back from device to host for GPU
- In the future, pyttb decomposition routines may call pygenten if it is available

Expect pygenten+pyttb to be the primary user-interface for GenTen going forward
- Expect to integrate with pytorch in the future as well to support AI use-cases
- Can also call python code for objective function in, e.g., goal-oriented CP decomposition

Available on pypi.org and installable through pip: "`pip install pygenten`"
- Pre-compiled binary wheels for Intel CPU + OpenMP builds (currently linux only)
- Source build for any other configuration, e.g., for a CUDA+MPI build on V100:

```
pip install --no-binary pygenten                          \
  --config-settings=cmake.define.PYGENTEN_CUDA=ON         \
  --config-settings=cmake.define.Kokkos_ARCH_VOLTA70=ON   \
  --config-settings=cmake.define.PYGENTEN_MPI=ON          \
  pygenten
```

amino-acid.ipynb

Code | Python 3 (ipykernel)

```
[1]: import pyttb as ttb
     import pygenten as gt

[2]: x = ttb.import_data("data/aminoacid_data_dense.txt")

[7]: u,_ = gt.cp_als(x, rank=3, maxiters=50, tol=1e-5, printitn=5)
```

```
Dense tensor:
  5 x 201 x 61 (6.13e+04 total entries)
  4.8e+04 Frobenius norm

Execution environment:
  MPI grid: 1 x 1 x 1 processes (1 total)
  Execution space: openmp (8 threads)

CP-ALS:
  CP Rank: 3
  MTTKRP method: row-based
  Gram formulation: symmetric

Iter   5: fit =  7.869399e-01 fitdelta =  3.4e-03
Iter  10: fit =  7.947348e-01 fitdelta =  1.8e-03
Iter  15: fit =  8.226425e-01 fitdelta =  1.1e-02
Iter  20: fit =  9.724381e-01 fitdelta =  4.8e-03
Iter  25: fit =  9.740721e-01 fitdelta =  1.6e-04
Iter  30: fit =  9.745770e-01 fitdelta =  6.9e-05
Iter  35: fit =  9.747905e-01 fitdelta =  2.9e-05
Iter  40: fit =  9.748814e-01 fitdelta =  1.3e-05
Final fit =  9.749011e-01
```

```
[8]: bar=lambda v,ax: ax.bar(range(0,v.shape[0]),v)
     line=lambda v,ax: ax.plot(v)
     u.vis([bar, line, line],mode_titles=["Sample","Emission","Excitation"]);
```