Optimization Methods and Software

ISSN: (Print) (Online) Journal homepage: www.tandfonline.com/journals/goms20

Tensor decompositions for count data that leverage stochastic and deterministic optimization

Jeremy M. Myers & Daniel M. Dunlavy

To cite this article: Jeremy M. Myers & Daniel M. Dunlavy (24 Sep 2024): Tensor decompositions for count data that leverage stochastic and deterministic optimization, Optimization Methods and Software, DOI: 10.1080/10556788.2024.2401981

To link to this article: https://doi.org/10.1080/10556788.2024.2401981

OPTIMIZATION METHODS & SOFTWARE
https://doi.org/10.1080/10556788.2024.2401981

Taylor & Francis
Taylor & Francis Group

Tensor decompositions for count data that leverage stochastic and deterministic optimization

Jeremy M. Myers[a] and Daniel M. Dunlavy[b]

[a]Scalable Modeling & Analysis, Sandia National Laboratories, Livermore, CA, USA; [b]Machine Intelligence and Vis, Sandia National Laboratories, Albuquerque, NM, USA

ABSTRACT

There is growing interest to extend low-rank matrix decompositions to multi-way arrays, or *tensors*. One fundamental low-rank tensor decomposition is the *canonical polyadic decomposition (CPD)*. The challenge of fitting a low-rank, nonnegative CPD model to Poisson-distributed count data is of particular interest. Several popular algorithms use local search methods to approximate the maximum likelihood estimator (MLE) of the Poisson CPD model. This work presents two new algorithms that extend state-of-the-art local methods for Poisson CPD. Hybrid GCP-CPAPR combines Generalized Canonical Decomposition (GCP) with stochastic optimization and CP Alternating Poisson Regression (CPAPR), a deterministic algorithm, to increase the probability of converging to the MLE over either method used alone. Restarted CPAPR with SVDrop uses a heuristic based on the singular values of the CPD model unfoldings to identify convergence toward optimizers that are not the MLE and restarts within the feasible domain of the optimization problem, thus reducing overall computational cost when using a multi-start strategy. We provide empirical evidence that indicates our approaches outperform existing methods with respect to converging to the Poisson CPD MLE.

ARTICLE HISTORY
Received 6 December 2023
Accepted 30 August 2024

KEYWORDS
Tensor; canonical polyadic decomposition; GCP; CPAPR; count data; Poisson

2020 MATHEMATICS SUBJECT CLASSIFICATIONS
15A69; 65F55

1. Introduction

Low-rank tensor decompositions in general, and the canonical polyadic decomposition (CPD) specifically, are important for multi-way data analysis [50]. Fitting the parameters of a low-rank, nonnegative CPD model to count data is often formulated as a nonlinear, nonconvex global optimization problem. When the data are assumed to be Poisson-distributed, one approach is to determine the optimal Poisson parameters that maximize the likelihood of the data via tensor maximum likelihood estimation [22]. The global optimizer to the optimization problem is the maximum likelihood estimator (MLE). Since global optimization algorithms are often prohibitively expensive for tensor data, great emphasis has been placed on developing efficient local methods for finding the Poisson CPD parameters. In practice, local methods for solving global optimization problems are often orchestrated in a multi-start strategy—i.e. computing a set of approximations from many random starting points—to increase the probability that the model best approximating the MLE has been

CONTACT J. M. Myers jermyer@sandia.gov

found. However, this approach demands significant computational resources when high-confidence solutions are required and may lead to excessive computations even for small problems. To mitigate this issue, we examine the role of randomization and determinism in Poisson CPD solvers.

Our contributions are:

- two Poisson CPD methods that compute the MLE with higher probability than current effective local search methods, and
- validation of our methods with open-source software on synthetic data.

1.1. *Hybrid GCP-CPAPR* (HybridGC)

Our first method is HybridGC, which uses a two-stage hybrid strategy built from effective local methods. Local methods are typically chosen to compute the Poisson CPD because the associated nonconvex optimization problem can be solved efficiently as a sequence of convex subproblems. However, such methods are only guaranteed to converge to local optimizers, which motivates the use of multi-start. Generalized CP decomposition (GCP) [42,51] incorporates general loss functions into CPD models, including a Poisson likelihood-based loss, and stochastic optimization methods. The first stage of HybridGC uses GCP with stochastic optimization to form a quick approximation which helps the method avoid local minimizers that are not the MLE. CP Alternating Poisson Regression (CPAPR) [22] is a deterministic Poisson CPD method that alternates over a sequence of convex Poisson loss subproblems iteratively. Previously, in [61], we showed that CPAPR is performant and can compute accurate approximations to the MLE with higher probability than GCP. The second stage uses CPAPR to refine the approximation from GCP to higher accuracy.

Global methods are typically avoided for CPD due to their high, often prohibitive, cost resulting from slow convergence. Nonetheless, they have proven to be effective for many other global optimization problems. Simulated Annealing (SA) [46,48] is one such technique that can handle high-dimensional, nonlinear cost functions with arbitrary boundary conditions and constraints, where controlled, iterative improvements to the cost function are used in the search for a better model. SA effectively leverages stochastic search to avoid local minimizers and can be followed by deterministic search to refine approximate global solutions. Inspired by this combined approach, we propose HybridGC for computing the MLE of the Poisson CPD.

1.2. *Restarted CPAPR with* SVDrop

HybridGC is an improvement over existing methods for Poisson CPD but can still converge to local optimizers that are not the MLE, as we illustrate through experimental results presented in Section 4. We have identified a specific algebraic property of the approximate solutions computed during iterations of HybridGC that can lead to such local optimizers. Current methods for computing CPD rely on algebraic operations applied to tensors that are unfolded into one or more matrix representations. In all cases we have explored and are considered in this paper, one or more singular values of at least one of these matrix representations of the approximate CPD tensor solution computed using HybridGC or one of

the local methods drops to nearly zero (i.e. to zero within machine precision) and leads to convergence to one of these local optimizers.

We introduce the parameterized SVDrop heuristic to help identify drops of singular values of unfolded CPD solutions during iterations of CPAPR (i.e. the iterative refinement method used in the HybridGC method described above). Combining this heuristic with a multi-start strategy, we propose the Restarted CPAPR with SVDrop method for Poisson CPD to increase the chances of converging to the MLE while reducing the computational cost involved with converging to local optimizers across the multiple initializations.

1.3. Organization

In Section 2, we introduce notation, provide the necessary background, and discuss related work. In Section 2.6, we formalize several metrics to compare CPD methods, some of which we used previously in [61]. In Section 3, we describe the data used in numerical experiments. In Section 4, we introduce Hybrid GCP-CPAPR (HybridGC). In Section 5, we introduce Restarted CPAPR with SVDrop. In our experiments, we demonstrate that both methods often improve the likelihood of convergence to the MLE, thereby reducing excessive computations when compared to multi-start where the local methods are standalone solvers. In Section 6, we propose future work.

2. Background and related work

2.1. Notation and conventions

The set of real numbers and integers are denoted as $\mathbb{R}$ and $\mathbb{Z}$, respectively. The real numbers and integers restricted to nonnegative values are denoted as $\mathbb{R}_+$ and $\mathbb{Z}_+$, respectively. The *order* of a tensor is the number of *dimensions* or *ways*. Each tensor dimension is called a *mode*. A scalar (tensor of order zero) is represented by a lowercase letter, e.g. x. A bold lowercase letter denotes a vector (tensor of order one), e.g. $\vec{v}$. A matrix (tensor of order two) is denoted by a bold capital letter, e.g. $\mathbf{A} \in \mathbb{R}^{m\times n}$. Tensors of order three and higher are expressed with a bold capital script letter, e.g. $\mathcal{X} \in \mathbb{R}^{m\times n\times p}$. Values *computed*, *approximated*, or *estimated* are typically written with a hat—e.g. $\widehat{\mathcal{M}} \in \mathbb{R}^{m\times n\times p}$.

The ith entry of a vector $\vec{v}$ is denoted v_i, the (i,j) entry of a matrix $\mathbf{M}$ is denoted m_{ij}, and the (i,j,k) entry of a three-way tensor $\mathcal{X}$ is denoted x_{ijk}. *Fibers* are the higher-order analogue of matrix rows and columns. Indices are integer values that range from 1 to a value denoted by the capitalized version of the index variable, e.g. $i = 1,\dots,I$. We use MATLAB-style notation for subarrays formed from a subset of indices of a vector, matrix, or tensor mode. We use the shorthand $i_j\colon i_k$ when the subset of indices forming a subarray is the range $i_j,\dots,i_k$. The special case of a colon : by itself indicates all elements of a mode, e.g. the jth column or mode-1 fibre of the matrix $\mathbf{A}$ is $\mathbf{A}(:,j) = \mathbf{A}(i_1\colon i_I, j)$. We use the *multi-index*

$$\boldsymbol{i} := (i_1, i_2, \dots, i_d) \quad \text{with } i_j \in \{1,2,\dots,I_j\} \quad \text{for } j = 1,\dots,d, \tag{1}$$

as a convenient shorthand for the $(i_1, i_2, \dots, i_d)$ entry of a d-way tensor.

Superscript T denotes non-conjugate matrix transpose. We assume vectors $\vec{u}$ and $\vec{v}$ are column vectors so that $\vec{u}^T\vec{v}$ is an inner product of vectors and $\vec{u}\vec{v}^T$ is an outer product of

vectors. We also denote outer products of vectors as $\vec{u} \circ \vec{v} = \vec{u}\vec{v}^T$, which is especially useful when describing the d-way outer products of d vectors for $d \geq 2$. The number of matrix or tensor non-zero elements is denoted $\text{nnz}(\cdot)$; conversely, the number of zeros in a matrix or tensor is denoted $\text{nz}(\cdot)$.

2.2. Matricization: transforming a tensor into a matrix

Matricization, as defined in [50], also known as *unfolding* or *flattening*, is the process of reordering the elements of a d-way array into a matrix. The mode-n matricization of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, denoted $\mathbf{X}_{(n)}$, arranges the mode-n fibres to be the columns of the resulting matrix.

2.3. Canonical polyadic decomposition

The canonical polyadic decomposition (CPD) represents a tensor as a finite sum of rank-one outer products, a generalization of the matrix singular value decomposition (SVD) to tensors. One major distinction is that there are no orthogonality constraints on the vectors of the CPD model. Thus we treat the matrix SVD as a special case of the CPD. Nonetheless, low-rank CP decompositions are appealing for reasons similar to those of the low-rank SVD, including dimensionality reduction, compression, de-noising, and more. Interpretability of CP decompositions on real problems is well-documented, with applications including exploratory temporal data analysis and link prediction [25], chemometrics [60], neuroscience [5], and social network and web link analysis [50,52].

One particular application of interest is when the tensor data are counts. In this case, a common modelling choice is to assume that the data follow a Poisson distribution so that statistical methods, like maximum likelihood estimation, can be applied to the analysis. One key challenge for computing the Poisson CPD is that a low-rank CP tensor model of Poisson parameters must satisfy certain nonnegativity and stochasticity constraints. In the next few sections we cover the details of the low-rank CP tensor models of Poisson parameters and decompositions which are the focus of this work.

2.4. Low-rank CP tensor model

Assume $\mathcal{X}$ is a d-way tensor of size $I_1 \times \cdots \times I_d$. The tensor $\mathcal{X}$ is rank-one if it can be expressed as the outer product of d vectors, each corresponding to a mode in $\mathcal{X}$, i.e.

$$\mathcal{X} = \vec{a}_1 \circ \vec{a}_2 \circ \cdots \circ \vec{a}_d. \tag{2}$$

More broadly, the *rank* of a tensor $\mathcal{X}$ is the smallest number of rank-one tensors that generate $\mathcal{X}$ as their sum [50]. We concentrate on the problem of approximating a tensor of data with a low-rank CP tensor model, i.e. the sum of relatively few rank-one tensors.

Let $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \ldots, \lambda_d] \in \mathbb{R}^d$ be a vector of scalars and let $\mathbf{A}_1 \in \mathbb{R}^{I_1 \times R}$, $\mathbf{A}_2 \in \mathbb{R}^{I_2 \times R}, \ldots, \mathbf{A}_d \in \mathbb{R}^{I_d \times R}$ be matrices. The *rank-R canonical polyadic (CP) tensor model of* $\mathcal{X}$ [39] is:

$$\mathcal{X} \approx \mathcal{M} = [\![\boldsymbol{\lambda}; \mathbf{A}_1, \ldots, \mathbf{A}_d]\!] := \sum_{r=1}^{R} \lambda_r \mathbf{A}_1(:, r) \circ \cdots \circ \mathbf{A}_d(:, r). \tag{3}$$

Each $\mathbf{A}_k \in \mathbb{R}^{I_k \times R}$ is a *factor matrix* with I_k rows and R columns. The jth *component* of the mode-k factor matrix is the column vector $\mathbf{A}_k(:,j)$. We refer to the form $\mathcal{M} = [\![\boldsymbol{\lambda}; \mathbf{A}_1, \ldots, \mathbf{A}_d]\!]$ as a *Kruskal tensor*.

2.5. Computing the Poisson CPD for count data

We focus on an application where all of the entries in a data tensor are counts. For the remainder of this work, let $\mathcal{X} \in \mathbb{Z}_+^{I_1 \times \cdots \times I_d}$ be a d-way tensor of nonnegative integers, let $\mathcal{M}$ be a CP tensor model of the form Equation (3), and assume the following about $\mathcal{X}$:

- each $x_{\boldsymbol{i}} \in \mathcal{X}$ is sampled from a Poisson distribution with parameter $m_{\boldsymbol{i}} \in \mathcal{M}$, and
- the tensor $\mathcal{X}$ has low-rank structure (i.e. $R < \sqrt[d]{\prod_{k=1}^{d} I_k}$ [51]).

Chi and Kolda showed in [22] that under the assumptions above a *Poisson CP tensor model* is an effective low-rank approximation of a tensor of counts, $\mathcal{X}$, and presented an algorithm for computing $\mathcal{M}$. The Poisson CP tensor model has shown to be valuable in analyzing latent patterns and relationships in count data across many application areas, including food production [19], network analysis [11,24], term-document analysis [21,38], email analysis [15], link prediction [25], geospatial analysis [27,37], web page analysis [49], and phenotyping from electronic health records [36,40,41].

One numerical approach to fit a low-rank Poisson CP tensor model to data is *tensor maximum likelihood estimation*, which has proven to be successful in practice. Computing the Poisson CPD via tensor maximum likelihood estimation is equivalent to solving the following nonlinear, nonconvex optimization problem:

$$\min_{\mathcal{M}} f(\mathcal{X}, \mathcal{M}) = \min \sum_{\boldsymbol{i}} m_{\boldsymbol{i}} - x_{\boldsymbol{i}} \log m_{\boldsymbol{i}}, \tag{4}$$

where $\boldsymbol{i}$ is the multi-index Equation (1), $x_{\boldsymbol{i}} \geq 0$ is an entry in $\mathcal{X}$, and $m_{\boldsymbol{i}} > 0$ is a parameter in the Poisson CP tensor model $\mathcal{M}$. The function $f(\mathcal{X}, \mathcal{M})$ in Equation (4) is the negative of the log-likelihood of the Poisson distribution (omitting the constant $\sum_{\boldsymbol{i}} \log(x_{\boldsymbol{i}}!)$ term). We will refer to it simply as *Poisson loss*.

In contrast to linear maximum likelihood estimation [63], where a single parameter is estimated using multiple data instances, tensor maximum likelihood estimation fits a single parameter in an approximate low-rank model to a single data instance. Within the tensor context, low-rank structure means that multiple instances in the data are linked to a single model parameter, a type of multilinear maximum likelihood estimation. This distinction is not made anywhere else in the literature, to the best of our knowledge.

Much of the research associated with computing the low-rank Poisson CPD via tensor maximum likelihood estimation has focussed on local methods [22,34,42,51], particularly with respect to computational performance [9,10,12,56,62,66,71]. Many of the current local methods for Poisson CPD can be classified as either an *alternating* [20,35] or an *all-at-once* [1,2,65] optimization method.

Alternating local methods iteratively solve a series of subproblems by fitting each factor matrix sequentially while the remaining factor matrices are held fixed. These methods are a form of coordinate descent (CD) [75], where each factor matrix is a block of components

that is fit while the remaining component blocks (i.e. factor matrices) are left unchanged. Since each block corresponds to a lower-dimensional problem, alternating tensor methods employ block CD iteratively to solve a series of easier problems. CP Alternating Poisson Regression (CPAPR) was introduced by Chi and Kolda in [22] as a nonlinear Gauss-Seidel approach to block CD that uses a fixed-point majorization-minimization algorithm called *Multiplicative Updates (CPAPR-MU)*. At the highest level, the CPAPR algorithm performs an *outer iteration* where optimizations are applied on each mode in an alternating fashion. An *inner iteration* is an optimization using multiplicative updates applied to a subset of variables corresponding to an individual mode. Inner iterations are performed until the convergence criterion is satisfied for a mode or up to the maximum allowable number, l_{max}. Outer iterations are performed until the convergence criterion is satisfied for the whole model or up to the maximum allowable number, k_{max}. The convergence criterion is based on the *Karush-Kuhn-Tucker (KKT)* conditions, necessary conditions for convergence to a local minimizer in nonlinear optimization. A local minimizer that satisfies the KKT conditions is called a *KKT point*.

Hansen *et al.* [34] presented two Newton-based, active set gradient projection methods using up to second-order information, *Projected Damped Newton (CPAPR-PDN)* and *Projected Quasi-Newton (CPAPR-PQN)*. Moreover, they provided extensions to these methods where each component block of the CPAPR minimization can be further separated into independent, highly-parallelizable row-wise subproblems; these methods are *Projected Damped Newton for the Row subproblem (CPAPR-PDNR)* and *Projected Quasi-Newton for the Row subproblem (CPAPR-PQNR)*.

One outer iteration of all-at-once optimization methods updates all optimization variables simultaneously. The Generalized Canonical Polyadic decomposition algorithm (GCP) [42] is a gradient descent method based on a generic formulation of first derivative information for arbitrary loss functions to compute the CPD via tensor maximum likelihood estimation. The original GCP method has two variants: 1) deterministic, which uses limited-memory quasi-Newton optimization (L-BFGS) and 2) stochastic, which supports gradient descent (SGD), AdaGrad [23], and Adam [47] optimizations. The stochastic variants perform loss function and gradient computations on samples of the input data tensor so that the search path is computed from estimates of these values. We focus here on GCP-Adam [51], which applies Adam for scalability.

More generally, we focus on the GCP and CPAPR families of tensor maximum likelihood-based local methods for Poisson CPD for the following reasons:

(1) *Existing Theory*: Method convergence, computational costs, and memory demands are well-understood. See [62, Table 1] for a summary of convergence results and computational cost, [22, § 5.4] for CPAPR storage results, and [51] for GCP-Adam storage results.
(2) *Available Software*: High-level MATLAB implementing both families is available in Tensor Toolbox for MATLAB (TTB)[1] [6,7]. A Python version is available in pyttb[2]. High performance C++ code that leverages the Kokkos hardware abstraction library [26] to provide parallel computation on diverse computer architectures (e.g. x86-multicore, GPU, etc.) is available with SparTen[3] for CPAPR [71] and Genten[4] for GCP [66]. Additional open-source software for MATLAB includes N-Way Toolbox [4] and Tensorlab [74]. Commercial software includes ENSIGN Tensor Toolbox [13].

2.6. Error in computing the CPD using multi-start

Local methods seek local minimizes. We apply them to global optimization problems by using a multi-start strategy [32,58] where a set of approximations are computed from many random starting points in the feasible domain of the problem. Our methodology is to generate N random Poisson CP tensor models as initial guesses and compute N rank-R Poisson CP tensor approximations starting from each initial guess, which we refer to as *multi-start*. From this set, we choose the 'best' local minimizer—i.e. the approximation that minimizes Equation (4)—as the approximation to the global optimizer. In turn, the effectiveness of a given method is determined in part by the probability it will converge to a solution approximating the global optimizer over all N starting points.

We define several quantities that we will use to compare the effectiveness of a given method in computing a model that minimizes Equation (4). Let $\mathcal{X}$ be a d-way data tensor with dimensions $I_1, \ldots, I_d$. Let $\mathcal{S} = \{\widehat{\mathcal{M}}^{(1)}, \ldots, \widehat{\mathcal{M}}^{(N)}\}$ be a set of rank-R Poisson CP tensor approximations such that $|\mathcal{S}| = N$. Let $\mathcal{M}^*$ denote the *maximum likelihood estimator (MLE)*, i.e. the global minimizer of Equation (4). In general, the global minimizer is unknown; however, it has been shown to exist when $f(\mathcal{X}, \mathcal{M})$ is finite, which is the case when $m_i > 0$ [22]. As a result, we aim to recover the *empirical MLE*, $\widehat{\mathcal{M}}^*$: the rank-R Poisson CP tensor model that is the best approximation to $\mathcal{M}^*$. We specify the *empirical MLE restricted to* $\mathcal{S}$, i.e. $\widehat{\mathcal{M}}^*_{\mathcal{S}} \equiv \widehat{\mathcal{M}}^* \in \mathcal{S}$, as the best approximation to the MLE from $\mathcal{S}$:

$$\widehat{\mathcal{M}}^*_{\mathcal{S}} = \{\widehat{\mathcal{M}}^{(j)} \in \mathcal{S} \mid f(\mathcal{X}, \widehat{\mathcal{M}}^{(j)}) \leq f(\mathcal{X}, \widehat{\mathcal{M}}^{(k)}),\ k = 1, \ldots, |\mathcal{S}|,\ j < k\}. \tag{5}$$

The condition that $j < k$ guarantees that the set is nonempty in the case of a tie. We write $\mathcal{S}_A$ when every element in $\mathcal{S}$ was computed by algorithm A. This notation will be useful later on when analyzing results from different algorithms.

2.7. An error estimator on the loss function

The probability that algorithm A converges from any starting point in the feasible region of Equation (4) to some $\widehat{\mathcal{M}}^{(n)} \in \mathcal{S}_A$ such that $f(\mathcal{X}, \widehat{\mathcal{M}}^{(n)})$ is within a ball of radius $\epsilon > 0$ of $f(\mathcal{X}, \widehat{\mathcal{M}}^*)$ is defined as

$$P_A(\rho_n < \epsilon) \quad \text{as } n = 1, \ldots, |\mathcal{S}_A|, \tag{6}$$

where

$$\rho_n := \frac{|f(\mathcal{X}, \widehat{\mathcal{M}}^{(n)}) - f(\mathcal{X}, \widehat{\mathcal{M}}^*)|}{|f(\mathcal{X}, \widehat{\mathcal{M}}^*)|} \tag{7}$$

is the relative error in Poisson loss. We can only estimate P_A since $\mathcal{S}_A$ is finite in practice. A computable estimator to Equation (6) is

$$\widehat{P}(\mathcal{S}_A, \epsilon) = \frac{|\{\widehat{\mathcal{M}}^{(n)} \in \mathcal{S}_A \text{for which} \rho_n < \epsilon\}|}{|\mathcal{S}_A|}. \tag{8}$$

Equation (8) is a conservative estimator since it does not account for solutions which may be closer to $\mathcal{M}^*$ but are more than ϵ-distance from $\widehat{\mathcal{M}}^*$. We omit $\mathcal{S}_A$ and write only $\widehat{P}(\epsilon)$ when the method is clear from the context.

2.8. An error estimator on the algebraic structures

We define two measures of approximation error quantifying the structural similarity between two Kruskal tensors based on their algebraic properties called *factor match score* [22,51,53,54].

2.8.1. Factor match score (FMS)

FMS is the maximum sum of cosine similarities over all permutations of the column vectors of all the factor matrices between two Kruskal tensors, $\mathcal{M}_1 = [\![\lambda^{\mathbf{A}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]\!]$ and $\mathcal{M}_2 = [\![\lambda^{\mathbf{B}}; \mathbf{B}_1, \ldots, \mathbf{B}_d]\!]$:

$$\text{FMS}(\mathcal{M}_1, \mathcal{M}_2) = \max_{\pi(\cdot)} \frac{1}{R} \sum_{r=1}^{R} \left(1 - \frac{|\xi_r - \zeta_r|}{\max\{\xi_r, \zeta_r\}}\right) \prod_{n=1}^{d} \frac{\mathbf{A}_n(:,j)^T \mathbf{B}_n(:,\pi(j))}{\|\mathbf{A}_n(:,j)\| \|\mathbf{B}_n(:,\pi(j))\|},$$

$$\text{where } \xi_r = \lambda_r^{\mathbf{A}} \prod_{n=1}^{d} \|\mathbf{A}_n(:,r)\| \quad \text{and} \quad \zeta_r = \lambda_r^{\mathbf{B}} \prod_{n=1}^{d} \|\mathbf{B}_n(:,r)\|. \tag{9}$$

The permutation $\pi(\cdot)$ reorders the columns of the factor matrices of $\mathcal{M}_2$ to maximize the number of columns that are correctly identified.

An FMS of 1 indicates collinearity among the columns of all factor matrices and thus a perfect match between the two Kruskal tensors. As in [57], we say $\mathcal{M}_1$ and $\mathcal{M}_2$ are *similar* if $\text{FMS}(\mathcal{M}_1, \mathcal{M}_2) \geq 0.85$ and *equal* if $\text{FMS}(\mathcal{M}_1, \mathcal{M}_2) \geq 0.95$, which are common values used to define acceptable matches in recent work [22,34,51]. FMS is a particularly useful measure of the effectiveness of a method in relating the low-rank structure of an approximation to that of a known model. Using FMS, we estimate the probability that a method computes models with the same algebraic structure as the empirical MLE. We formalize this now.

2.8.2. Probability of similarity

For each computed solution $\widehat{\mathcal{M}}^{(n)} \in \mathcal{S}$, $n = 1, \ldots, |\mathcal{S}|$, define an indicator function $\psi_n(\mathcal{M}, \widehat{\mathcal{M}}^{(n)}, t)$ that is 1 when the nth model has $\text{FMS}(\mathcal{M}, \widehat{\mathcal{M}}^{(n)}) \geq t$ and 0 otherwise; i.e.

$$\psi_n(\mathcal{M}, \widehat{\mathcal{M}}^{(n)}, t) = \begin{cases} 1, & \text{if FMS}(\mathcal{M}, \widehat{\mathcal{M}}^{(n)}) \geq t \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

We use Equation (10) in our discussions below to quantify the *fraction over N solves with FMS greater than t*,

$$\Psi(\mathcal{M}, \mathcal{S}, t) = \frac{1}{N} \sum_{n=1}^{|\mathcal{S}|} \psi_n(\mathcal{M}, \widehat{\mathcal{M}}^{(n)}, t), \quad \text{where } \widehat{\mathcal{M}}^{(n)} \in \mathcal{S}, \forall n \in \{1, \ldots, |\mathcal{S}|\}. \tag{11}$$

2.9. Related work

There are other approaches in the literature that seek to fit models with other distributions in the exponential family or that use other algorithms to estimate parameters. Alternating least squares methods are relatively easy to implement and effective when used with

LASSO-type regularization [14,28]. The method of Ranadive *et al.* [68], CP-POPT-DGN, is an all-at-once active set trust-region gradient-projection method. CP-POPT-DGN is functionally very similar to CPAPR-PDN. Whereas CP-POPT-DGN computes the search direction via preconditioned conjugate gradient (PCG), CPAPR-PDNR computes the search direction via Cholesky factorization. The most significant differences are: 1) CP-POPT-DGN is all-at-once whereas all CPAPR methods are alternating, and 2) CPAPR can take advantage of the separable row subproblem formulation to achieve more fine-grained parallelism. The Generalized Gauss-Newton method of Vandecapelle *et al.* [72] follows the GCP framework to fit arbitrary non-least squares loss via an all-at-once optimization and trust-region-based Gauss-Newton approach. Hu *et al.* [43,44] re-parameterized the Poisson regression problem to leverage Gibbs sampling and variational Bayesian inference to account for the inability of CPAPR to handle missing data. Other problem transformations include probabilistic likelihood extensions via Expectation Maximization [45,67] and a Legendre decomposition [70] instead of a CP decomposition.

Section 5 presents the SVDrop heuristic that can identify approximate solutions that will converge to non-MLE solutions (i.e. local optimizers) for Poisson CPD early in the iterative process. This heuristic is based on spectral properties of unfoldings of the approximate solution, and previous work has also attempted to characterize non-optimal solutions for the CPD problem based on related properties. Early work by Kruskal *et al.* [55] studied two-factor degeneracies (2FD), where two components of a CPD model are highly correlated in all three modes. Mitchell and Burdick introduced a test for 2FD in [59], which uses an early definition of FMS to identify 2FD between successive iterates. More recent work [29] adds constraints (e.g. orthogonality constraints, ridge regression, SVD penalty) to the CPD problem to avoid 2FD.

Breiding and Vannieuwenhoven proposed a formulation of CPD as a Riemannian optimization problem, described spectral properties of approximate solutions to this reformulation that lead to ill-conditioning, and proposed a restart technique to escape from regions of ill-conditioning where iterations can stagnate [16–18]. Their work most closely resembles the ideas presented here, including identifying spectral properties of iterations that lead to non-optimal solutions and restarting sub-optimal iterations before full convergence is attained to reduce computational cost. However, their approach was developed for CPD with Gaussian data assumptions, including the Riemannian optimization objective function and derivatives (whose spectral properties are used for identifying ill-conditioning), a Riemannian optimization retraction function based on a low-rank Tucker decomposition, ST-HOSVD [73], that has not been extended directly to Poisson CPD, and a restart method based on local perturbations of iterates in regions of ill-conditioning. Our work here differs in that we focus exclusively on methods applicable to Poisson CPD, leverage spectral properties of the unfoldings of iterates (rather than of the Hessian of the objective function at iterates, thus not requiring second-order derivative information that may not be readily available), and use a restart technique that does not rely on the current iterates (as empirical evidence with Poisson CPD solvers indicated that such local perturbations did not consistently lead to convergence to MLE solutions). Moreover, the results presented here are empirical, whereas Breiding and Vannieuwenhoven focussed on both theoretical analysis and empirical evidence to identify ill-conditioning of CPD and propose remedies. Future work could consider extending their ideas to better understand the challenges associated with Poisson CPD that we identify and address in this paper.

3. Data examples

3.1. `LowRankSmall`

A synthetic count tensor with dimensions $4 \times 6 \times 8$, generated rank $R = 3$, and 17 nonzero entries (8.85% dense). The sparse tensor is fully provided in Appendix 1 as Listing 1. All experiments featuring this dataset were conducted with Tensor Toolbox for MATLAB v3.3. Additional implementation details are specified in Sections 4.2 and 5.4.

3.2. `MedRankLarge`

A synthetic count tensor with dimensions $1000 \times 1000 \times 1000$, generated rank $R = 20$, and 98026 nonzero entries (0.009% dense). All experiments featuring this dataset were conducted with SparTen for CPAPR and Genten for GCP-Adam on a dual-socket Intel Xeon Gold processor with 18 cores per socket. Both tools were compiled for single-node parallelism with GCC v8.3.1 and were run using all available OpenMP threads. Additional implementation details are specified in Section 4.2.

4. Hybrid GCP-CPAPR

We present Hybrid GCP-CPAPR (HybridGC), an algorithm for Poisson CPD that estimates the solution of a nonlinear, nonconvex optimization problem by *approximating a global optimization algorithm through the composition of local methods*. HybridGC first uses stochasticity to compute a coarse-grained estimate of the model and then refines the model with a deterministic method. Our numerical experiments demonstrate the synergy of this hybrid approach: HybridGC yields an effective algorithm that computes an approximation to the MLE for Poisson CPD with higher accuracy than the methods it leverages. The stochastic stage makes HybridGC scalable to very large problems and the deterministic stage allows the method to exploit convergence results in [22]. To the best of our knowledge, this is the first work that extends similar approaches in the matrix case (for example, [33]) to low-rank tensor decompositions in this way.

4.1. HybridGC method

Like SA, HybridGC leverages both stochastic and deterministic optimizations to start from an initial guess, iterate according to a schedule, and converge to a solution approximating the global optimizer. Specifically, HybridGC iterates from an initial guess $\mathcal{M}_0$ via a two-stage optimization between stochastic and deterministic search to return a Poisson CP tensor model $\widehat{\mathcal{M}}$ that is an estimate to $\mathcal{M}^*$. In the first stage, the stochastic search method starts from $\mathcal{M}_0$ and iterates for j iterations to return an intermediate solution, $\mathcal{M}_1$. In contrast to SA, which uses random perturbation for the 'heating' step, we use GCP-Adam [51] for structured stochastic optimization. In the second stage, deterministic search refines $\mathcal{M}_1$ for k iterations to return $\mathcal{M}_2$. We use CPAPR with Multiplicative Updates [22] as our 'cooling' step. HybridGC returns $\widehat{\mathcal{M}} = \mathcal{M}_2$ as an estimate to the global optimizer, $\mathcal{M}^*$. The details of HybridGC are given below in Algorithm 1.

Presently, our analogue of the temperature schedule is the combination of the number of GCP epochs and CPAPR outer iterations. A GCP *epoch* is comprised of one or more

iterations. The Adam step parameter and estimate of the objective function value both are updated once per epoch. The stochastic gradient is computed and a descent step is taken in each iteration. A CPAPR *outer iteration* corresponds to one pass over all of the tensor modes. For each mode, the tensor is matricized and one or more *inner iterations* are taken to optimize the factor matrix corresponding to that mode.

In further contrast to SA, we do not include a notion of acceptance-rejection with respect to newly obtained states; we leave this to future work. We only consider stochastic search followed by deterministic search and not the opposite. This is because stochastic search directions are found using estimates of the objective function from sample points. Thus it would be possible for the algorithm to converge to a minimizer yet remain marked as *not converged* if the objective function value were only coarsely estimated. Thus it is likely that stochastic search would move away from the optimizer.

Algorithm 1 Hybrid GCP-CPAPR

function HybridGC(tensor $\mathcal{X}$, rank R, initial guess $\mathcal{M}_0$)
  $\mathcal{M}_1 \leftarrow \text{GCP}(\mathcal{X}, R, \mathcal{M}_0)$
  $\mathcal{M}_2 \leftarrow \text{CPAPR}(\mathcal{X}, R, \mathcal{M}_1)$
  return model tensor $\widehat{\mathcal{M}} = \mathcal{M}_2$ as estimate to $\mathcal{M}^*$

4.2. Numerical experiments

This section presents experiments that were designed to evaluate the algorithm effectiveness of HybridGC compared to GCP-Adam and CPAPR-MU as standalone solvers. We demonstrate this by performing many independent trials on each of two synthetic low-rank Poisson multilinear datasets. For each trial on a dataset, we compute three rank-R Poisson CPD approximations with GCP-Adam, CPAPR-MU, and HybridGC; the specific parameterizations are detailed below in Section 4.2.1. For the remainder of this work, we refer to GCP and CPAPR without further specifying the optimization routine. We use $\mathcal{S}_G$, $\mathcal{S}_C$, and $\mathcal{S}_H$ to refer to the sets of approximations computed with GCP, CPAPR, and HybridGC, respectively.

We treat HybridGC decompositions as those computed with $j \geq 0$ GCP epochs followed by $k \geq 0$ CPAPR outer iterations. By default, which we use in our experiments, GCP sets $1\text{epoch} = 1000\text{iterations}$ and CPAPR sets a maximum of 10 inner iterations per mode per outer iteration.

Using the notation set above, we denote the solutions that were computed with GCP, CPAPR, and HybridGC as

$$\mathcal{S}_G = \{\widehat{\mathcal{M}}_j \mid \widehat{\mathcal{M}}_j \text{computedbyGCP}\};$$
$$\mathcal{S}_C = \{\widehat{\mathcal{M}}_j \mid \widehat{\mathcal{M}}_j \text{computedbyCPAPR}\}; \quad \text{and,}$$
$$\mathcal{S}_H = \{\widehat{\mathcal{M}}_j \mid \widehat{\mathcal{M}}_j \text{computedbyHybridGC}\}.$$

4.2.1. Procedure

Fix an input tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ and a rank R. Specify a maximum work budget $W = J_{max} + K_{max}$ for all methods where $J_{max} \geq 0$ and $K_{max} \geq 0$ are the maximum allowable

number of outer iterations for GCP and CPAPR, respectively. If $J_{max} = 0$ and $K_{max} = W$, then HybridGC is equivalent to CPAPR. Conversely, if $J_{max} = W$ and $K_{max} = 0$, then HybridGC is equivalent to GCP. In this way, HybridGC generalizes both methods by 'interpolating' GCP and CPAPR when both $J_{max} > 0$ and $K_{max} > 0$.

Starting from the same random initial guess, we computed $j \in (0, \ldots, W)$ rank-R decompositions with GCP iterating for at most j epochs. GCP is considered converged when the stochastic gradient learning rate α is decayed by a factor of 10^{-1} from 10^{-3} (the default) to 10^{-15}, which occurs if the solver fails to reduce the loss function value in an epoch. Next, starting from each of the $W + 1$ epoch iterates computed with GCP, we computed $k \in (W, W - 1, \ldots, 0)$ rank-R decompositions with CPAPR iterating for at most k outer iterations. CPAPR is considered converged when the KKT-based criterion is less than or equal to 10^{-15}. Since each HybridGC trial produced $W + 1$ decompositions, only the empirical MLE restricted to that trial was chosen for comparison.

In total, $N = 110,266$ rank $R = 3$ decompositions of `LowRankSmall` were computed with each algorithm. Two minimizers were found in this set of solutions and the minimizer with the absolute minimum loss function value was chosen as the empirical MLE, with $f(\mathcal{X}, \mathcal{M}) = -26.21406230880176$. Among the $N = 100$ rank $R = 20$ decompositions of `MedRankLarge` computed with each algorithm, many minimizers were found. The minimizer with the absolute minimum loss function value over all trials was chosen as the empirical MLE. In both cases, we refer to the set of solutions within a ball of radius ϵ around the empirical MLE as having converged to the MLE for some $\epsilon > 0$. In the case of `LowRankSmall`, since we have only one other minimizer, in our discussions we refer explicitly to solutions within a ball of ϵ around the second local minimizer, with $f(\mathcal{X}, \mathcal{M}) = -5.838518788084730$, as having converged to *the local minimizer* for some $\epsilon > 0$. We specify ϵ where it is germane to the discussion.

4.2.2. Comparison on the loss function

We now compare the effectiveness of HybridGC as an algorithm for solving a nonlinear, nonconvex optimization problem by considering the Poisson loss Equation (4) and the MLE-probability estimator based on it Equation (8). The empirical MLE is denoted $\mathcal{M}^*_{\mathcal{S}}$, with $\mathcal{S} = \mathcal{S}_G \cup \mathcal{S}_C \cup \mathcal{S}_H$.

Table 1 presents average behaviour about algorithm effectiveness as estimates of the probability that each method computed the empirical MLE with relative error Equation (8) less than ϵ for `LowRankSmall` and `MedRankLarge`. Viewing the average behaviour, we note the following:

- For the small dataset with low rank (`LowRankSmall`), HybridGC and CPAPR had comparable precision at all levels of accuracy. Additional work may be conducted to determine if the differences are statistically significant.
- HybridGC always converged to the MLE when CPAPR did. In our experiments with `LowRankSmall`, HybridGC converged to the MLE in the same trials as CPAPR did plus in an additional 0.42% of trials.
- HybridGC always converged to the MLE at the same or a higher rate than GCP. Although there were instances on `LowRankSmall` where GCP converged to the MLE and HybridGC did not, the opposite occurred in 13.7% more trials.

Table 1. Estimate of probability each method computes a solution within ϵ-radius of approximate global optimizer.

(a) `LowRankSmall` dataset (> 110 K trials).				(b) `MedRankLarge` dataset (100 trials).			
ϵ	CPAPR	GCP	HybridGC	ϵ	CPAPR	GCP	HybridGC
10^{-1}	0.963	0.963	**0.967**	10^{-1}	1.00	1.00	1.00
10^{-2}	0.963	0.963	**0.967**	10^{-2}	**0.46**	0.04	**0.46**
10^{-3}	0.963	0.879	**0.967**	10^{-3}	0.03	0.00	**0.17**
10^{-4}	0.963	0.003	**0.967**	10^{-4}	0.00	0.00	**0.01**

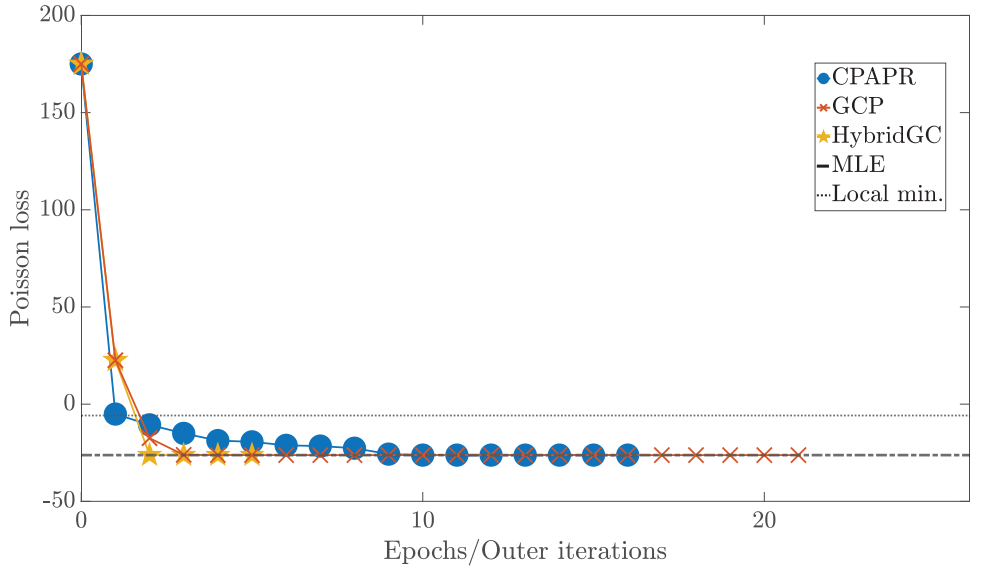
Table 2. Total time (sec.) for each trial in Figure 1.

Trial	CPAPR	GCP	HybridGC
Figure 1(a)	0.35	26.58	1.39
Figure 1(b)	2.34	21.97	1.51

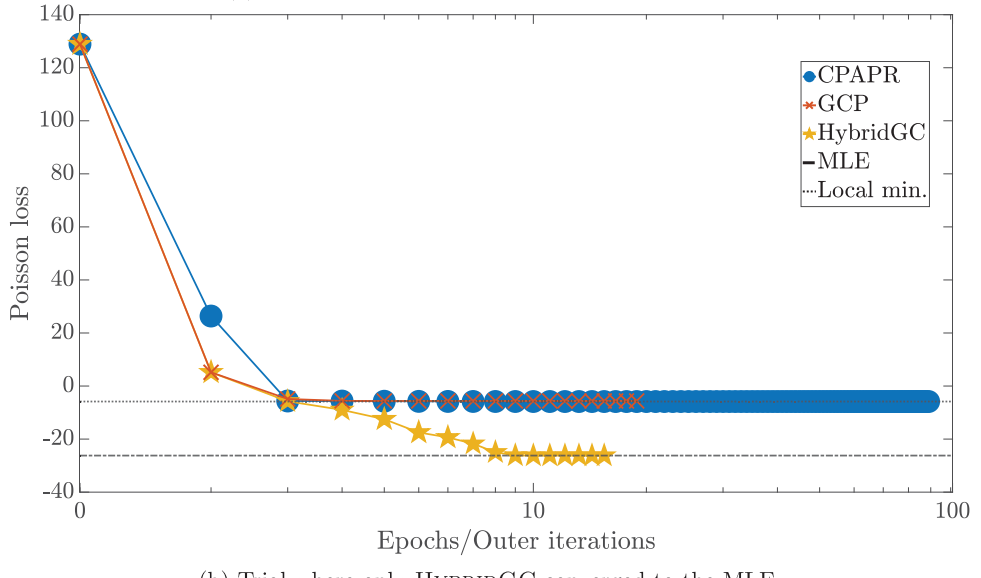
- On both datasets HybridGC had a higher probability of getting close to the empirical MLE at high accuracy.
- Even for small input tensors with low rank, GCP was virtually incapable of resolving the MLE beyond a coarse-grain approximation. The situation was worse for larger tensors with more components.

Figure 1 highlights two behaviours observed in two different trials among the $N = 110$, 266 trials we ran on `LowRankSmall`. Figure 1(a) presents the traces in Poisson loss function value from one trial where all methods computed approximations close to the empirical MLE when started from the same initial guess for `LowRankSmall`. Figure 1(b) presents similar traces except when only HybridGC converged to the MLE and the standalone solvers, GCP and CPAPR, converged to a different local minimizer. Of all $N = 110$, 266 trials, the empirical MLE was computed by HybridGC. The loss function values of the MLE and the second local minimizer are shown in both Figures 1(a and b) for direct comparison with the same axes.

Table 2 shows the wall clock times of the two trials presented in Figure 1. It is clear that GCP performs worse than CPAPR and HybridGC for this parameterization regardless of the minimizer it converges to. When CPAPR converged to the MLE, it converged approximately roughly 4× faster than HybridGC. Closer examination reveals that more than 97% of HybridGC runtime was spent in the GCP stage. We found the default parameterization (i.e. 1epoch = 1000iterations) to determine the poor performance of HybridGC compared to CPAPR. To see this, we studied the convergence behaviour and performance of running the CPAPR stage of HybridGC starting from the approximation computed after every iteration in the first epoch of the GCP stage. Not only did HybridGC converge to the MLE from every starting point but HybridGC outperformed CPAPR in 14.5% of starts, shown in Figure 2. In the best case, HybridGC was 29% faster than CPAPR. Since it was not possible to conduct a similar experiment for all trials due to storage and computational limitations, we acknowledge that this result may not hold in general. Nonetheless, this anecdotal evidence suggests, when considered with the overall convergence results described above, that HybridGC with very small epoch sizes in the GCP stage may be highly effective in computing the MLE efficiently with high probability.



(a) Trial where all methods converged to the MLE.



(b) Trial where only HybridGC converged to the MLE.

Figure 1. Examples of two types of behaviours of traces of loss function values for GCP, CPAPR, and HybridGC on `LowRankSmall`. The loss function values of the MLE and the second local minimizer are shown in both plots for direct comparison. (a) Trial where all methods converged to the MLE. (b) Trial where only HybridGC converged to the MLE.

A few additional remarks:

- In our experiments we observe the benefit of performing some amount of stochastic search followed by deterministic search. Owing to performing one epoch of stochastic search, HybridGC quickly identified the basin of attraction of the MLE and converged

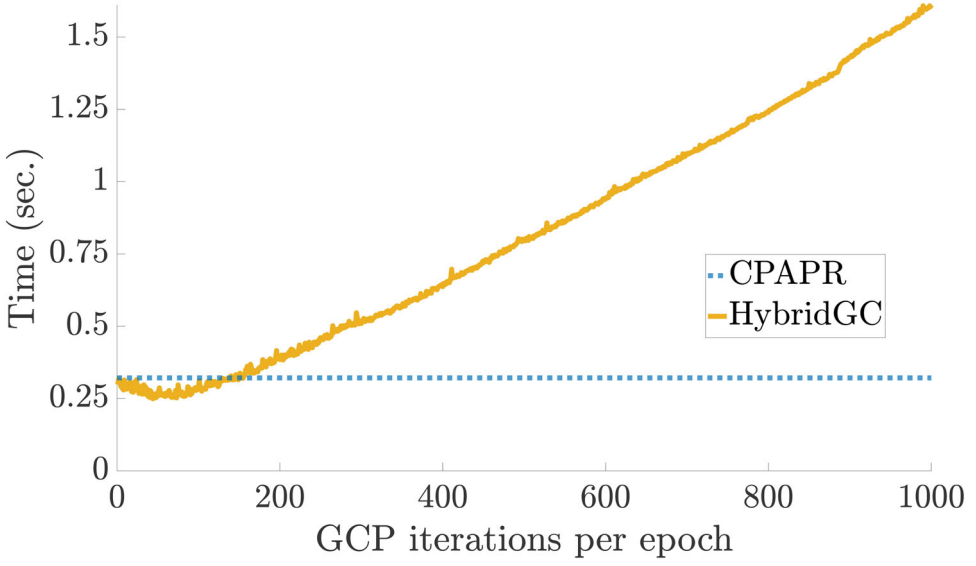


Figure 2. Performance of HybridGC versus CPAPR as a standalone solver where both methods converge to the MLE. The *x*-axis indicates how many iterations were run in the GCP stage before starting the CPAPR stage of HybridGC.

before GCP and CPAPR in both trials, even in the case where the standalone method converges to a local minimizer that is different from the MLE. The iteration histories of GCP and CPAPR are consistent with results from prior work on small tensors [61].

- The shared behaviour among all methods of making only incremental progress before finally converging is a feature of the theoretical convergence properties of each method. Mathematically, CPAPR (in the case of MU) and the deterministic stage of HybridGC converge sublinearly in the basin of attraction to the MLE; GCP (in the case of Adam) converges only linearly at best. See [61, Table 1] for details.

4.2.3. Comparison as algebraic structures

Next, we evaluate HybridGC as a method for computing an approximate low-rank basis to the global optimizer. We calculated the fraction of trials with FMS greater than t using Equation (11), with $t \in [0, 1]$, for GCP and CPAPR, i.e. $\Psi(\widehat{\mathcal{M}}^*_{\mathcal{S}}, \mathcal{S}_G, t)$ and $\Psi(\widehat{\mathcal{M}}^*_{\mathcal{S}}, \mathcal{S}_C, t)$, respectively. We repeated this calculation for HybridGC, i.e. $\Psi(\widehat{\mathcal{M}}^*_{\mathcal{S}}, \mathcal{S}_H, t)$, and grouped the results by the number of epochs taken by the first stage of HybridGC. Figure 3 presents these results for GCP, CPAPR, and HybridGC (up to 10 epochs of GCP). See Figure A1 in Appendix 4 for supplementary results. Since all curves showed the same behaviour for $t < 0.6$, we report values for $t \in [0.6, 1]$.

HybridGC tended to have a higher likelihood than GCP or CPAPR in finding a low-rank basis equal to the empirical MLE when FMS > 0.95, which is considered high accuracy. This figure provides numerical evidence that HybridGC—parameterized as a small amount of stochastic search ($\sim$10 GCP epochs) followed by deterministic search—was superior to GCP and CPAPR by themselves in computing high accuracy models (solutions with FMS greater than 0.95).

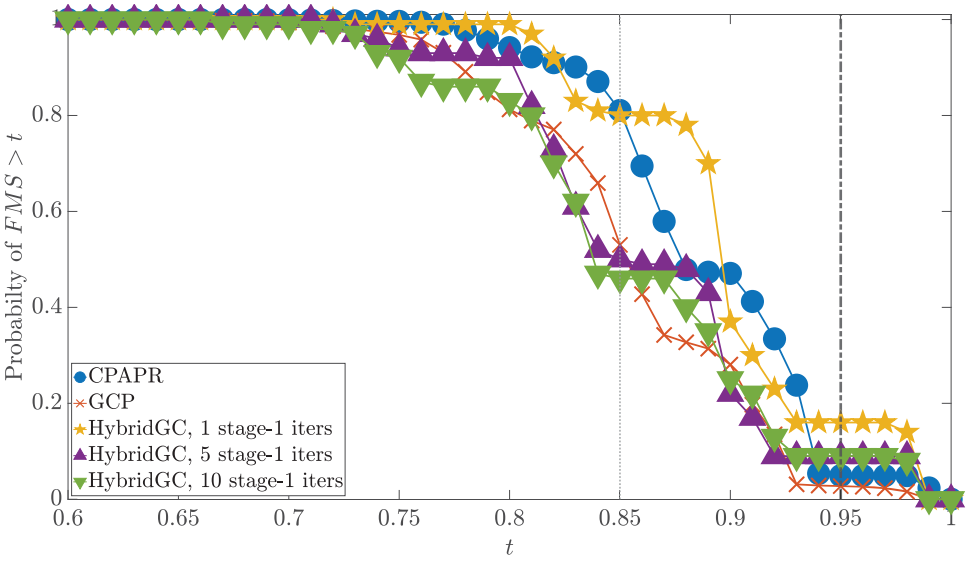


Figure 3. Factor match scores between CP models computed with HybridGC, CPAPR-MU, and GCP-Adam and the approximate global optimizer, $\widehat{\mathcal{M}}^*_S$. The dash-dot grey vertical lines and dotted black vertical lines denote the levels of 'similar' and 'equal' described in [57].

5. Restarted CPAPR with SVDrop

Now, we present our second algorithm, Restarted CPAPR with SVDrop.

We observed in Section 4.2.2 that there are situations where HybridGC converges to the MLE but a standalone method like CPAPR does not, and vice versa. The standalone method may converge to a minimizer far from the MLE despite having started from the same point. The standalone method may also converge to the MLE whereas HybridGC converges to some other minimizer. Thus it is necessary to characterize the situations that end in algorithm failure[5] so that we may explain these seemingly conflicted outcomes. Since it is easier to reason about a deterministic search path, we focus on sources of failure in CPAPR and leave a similar study of GCP for future work. CPAPR is also interesting because of its high success rate, meaning it is sometimes feasible to examine all failed trials exhaustively. Furthermore only a small number of parameters affect the search path of CPAPR in [22, Algorithm 3]. Another motivator for the work presented here is that CPAPR is used to refine the solution in HybridGC, thus understanding convergence properties is important. Taking these factors into account, CPAPR is a good candidate to analyze in order to understand why and when local methods fail for Poisson CPD.

In this section we develop a heuristic called SVDrop to identify iterates of CPAPR that will converge to local minimizers based on the singular values of the tensor unfoldings. This heuristic is combined with a restarting procedure to form a novel variant of the CPAPR algorithm called Restarted CPAPR with SVDrop (Algorithm 2). We first motivate our analysis by observing a previously unreported problem and reasoning as to its implications in Section 5.1. In Section 5.2, we characterize this problem and demonstrate empirically that: (1) the problem occurs frequently when computing the Poisson CPD for a small synthetic dataset and (2) it can be identified by tracking the Rth largest singular

values of the mode unfoldings at successive iterations. The method is fully described in Section 5.3. In Section 5.4, we present experimental evidence that Restarted CPAPR with SVDrop improves the probability of convergence to the MLE with an acceptable increase in computational cost.

5.1. The drawback of extra (or too few) inner iterations

Chi and Kolda's CPAPR paper [22] included a section titled 'the benefit of extra inner iterations'. Their conclusion was that although the maximum allowable number of inner iterations l_{max} 'does not significantly impact accuracy… increasing l_{max} can decrease the overall work and runtime'. They drew their conclusion from the mean and median factor match score (FMS) between the model and the 'true solution'. However, this definition of accuracy is incomplete since it ignores error estimators on the loss function. Instead, our definition of accuracy includes both the objective function value and expected convergence behaviour over many trials. Ultimately, we reach the opposite conclusion: the maximum allowable number of inner iterations can significantly impact algorithm accuracy. Subsequently, overall work and runtime are also affected. For instance, we observed situations where, from one fixed starting point, CPAPR would converge to different minimizers for increasing values of l_{max} in an alternating fashion: to the MLE for some value of l_{max}, then to a different minimizer for a larger value, and again to the MLE for an even larger value of l_{max}. In some cases, this alternating pattern repeated multiple times. This behaviour was not rare. In one trial from 3,677 starting points, we observed some type of alternating convergence pattern in 3,180 instances (86.4%). In general, characterizing the sensitivity of CPAPR to l_{max} is complicated.

Figure 4 demonstrates one example empirically where the effect of extra inner iterations counters Chi and Kolda's claim. Both plots show the trace of the objective function value for the first 8 outer iterations of CPAPR. The upper plot shows the trace of the objective function value over the total iteration history when $l_{max} = 4$. CPAPR converges to the empirical MLE (black dash-dot line) in 8 outer iterations. The lower plot is taken from the same initial point except the maximum number of inner iterations is one greater, i.e. $l_{max} = 5$. By the 8th outer iteration, CPAPR has settled in the basin of attraction of a KKT point far from the MLE (black dotted line). We will return to this case throughout the rest of this section, so we will refer to it as the *exemplar trial*.

Incremental changes to this parameter can result in drastically different outcomes. In experimentation, we frequently observe that, for the first several outer iterations, CPAPR tends to max out the number of inner iterations in each mode without converging. This led to the conclusion that early iterations are especially critical and sensitive to l_{max}. Figure 5 presents a conceptual model explaining the situation. The contour plot reflects the minima (in blue) and maxima (in brown) of a $d = 2$ problem. Darker shades reflect more extreme values. The x- and y-axes represent search paths in the directions of the second and first modes, respectively. The green, magenta, and red lines represent the search paths of CPAPR from the same initial starting point (yellow circle) but with different values for l_{max}. The magenta and red paths allow too many or too few inner iterations and converge to local minimizers. The green path allows the 'Goldilocks' amount—this choice leads to the MLE.[6] We will show that our novel analysis can differentiate between the green path and the magenta and red paths at runtime.

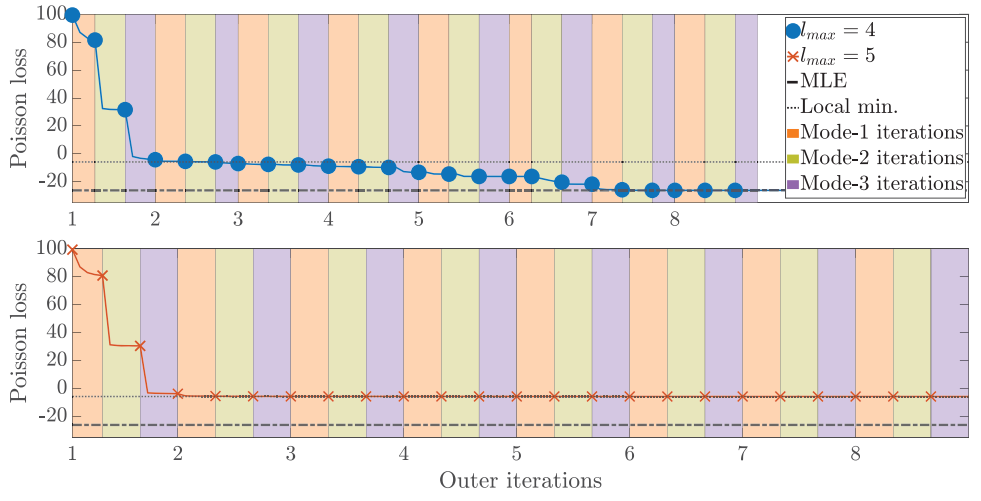


Figure 4. Traces of objective function values for the *exemplar trial*: two decompositions computed by CPAPR starting from the same initial guess but with different numbers of maximum allowable inner iterations per mode. The *x*-axis is given in terms of the number of outer iterations, so optimizations by mode are differentiated with vertical blocks of colour. Only the first 8 outer iterations are shown: CPAPR with $l_{max} = 4$ (top) converges to the MLE; CPAPR with $l_{max} = 5$ (bottom) has settled in the basin of attraction of a different minimizer. The first Poisson loss value in each mode is emphasized with a marker.

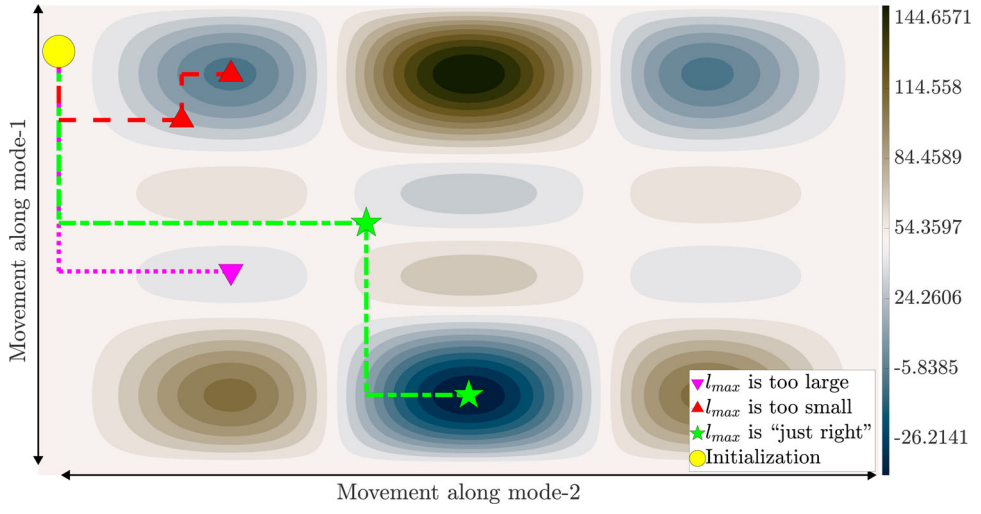


Figure 5. The contour plot illustrates how convergence depends on the number of inner iterations in the search direction for a 2D problem. Blue represents minima and brown represents maxima; darker shades are more extreme values than lighter shades. From the same starting initialization, CPAPR is run with three different values for inner iterations l_{max}: (1) the 'Goldilocks' amount that leads to the MLE; (2) too few or (3) too many inner iterations, which both lead to different minimizers.

5.2. Spectral properties identify rank-deficient solutions

By analyzing the singular values of each mode unfolding, we can see critical changes to the model tensor that are otherwise hidden. In particular, we consider the R-th largest

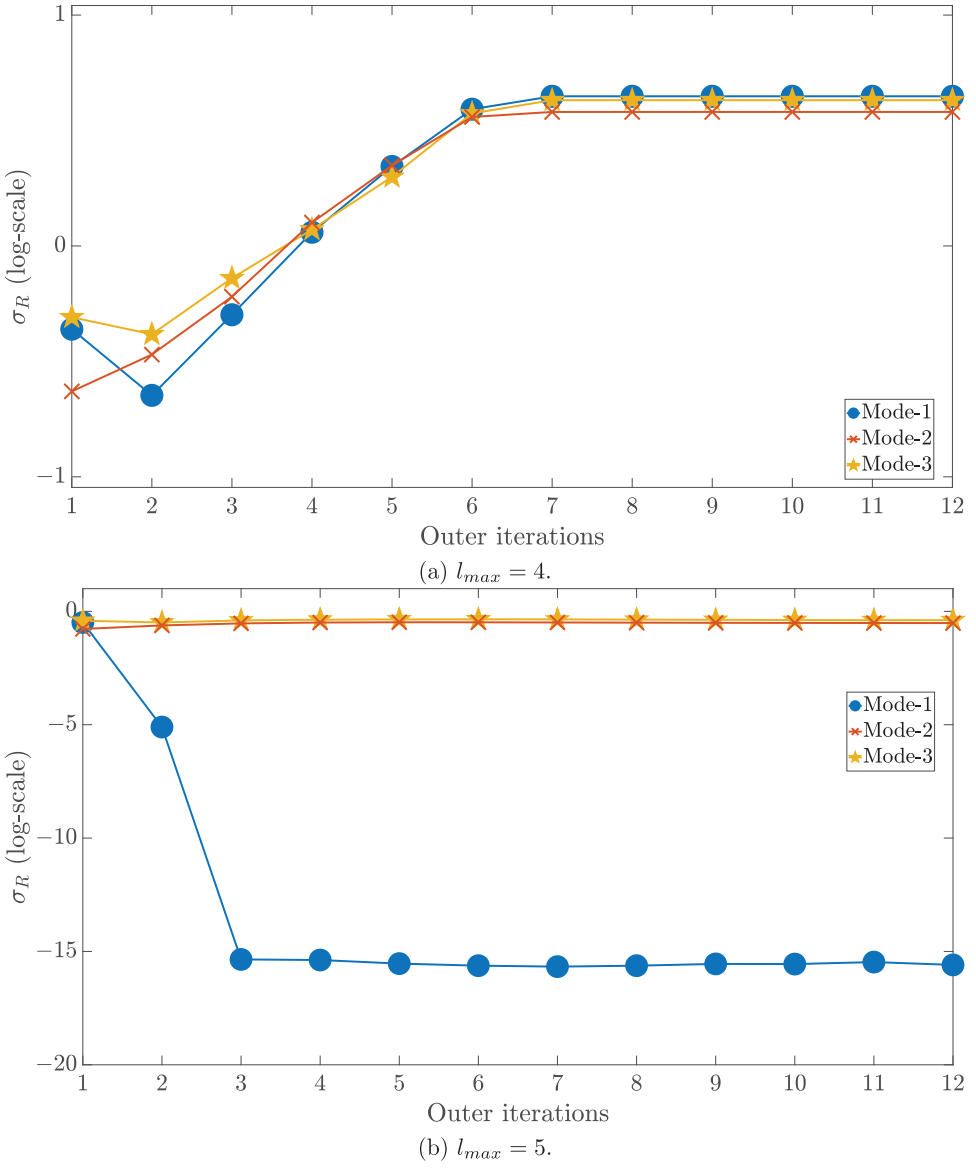


Figure 6. Traces of the $R = 3$-rd largest singular value of each mode unfolding after every update for the *exemplar trial* on `LowRankSmall`. Analyzing the Rth largest singular value of each mode yields an explicit signal indicating convergence to a rank-deficient solution. The x-axis in both plots has been truncated to the total number of outer iterations needed to converge to the MLE; the traces shown in the lower plot continue until convergence to the local minimizer without change. (a) $l_{max} = 4$. (b) $l_{max} = 5$.

singular value when the requested decomposition rank is R. Figure 6 shows the values of the third largest singular value (since $R = 3$) of each of the mode unfoldings over the iteration history: Figure 6(a) plots when $l_{max} = 4$, the case when CPAPR converges to the MLE; Figure 6(b) plots when $l_{max} = 5$, the case when CPAPR converges to a different local minimizer. Both trials iterated from the same initial guess. The critical observation is

that the Rth singular value in mode-1 when $l_{max} = 5$ inner iterations are taken per outer iteration (blue line with circle markers in Figure 6(b)) is driven below machine precision. We define a *rank-deficient solution* as an approximate solution (i.e. iterate of CPAPR) where one or more of the Rth largest singular values of the tensor unfoldings across all modes is zero. A rank-deficient solution is a KKT point, but it is not the MLE. *Ceteris paribus*, the search path that leads to it is determined by l_{max}.

At present this analysis of success versus failure is only possible by examining the singular values of the mode unfoldings. The λ values of the CP model, which are freely available, do not provide the same information as the singular values of the mode unfoldings, which must be computed at additional cost. For example, Figure 7 displays the λ values of the CP model at each iteration for both values of l_{max} in the *exemplar trial*. The λ_1 and λ_3 values (blue and yellow, respectively) became nearly identical when the number of inner iterations taken per outer iteration was $l_{max} = 5$. This occurred at nearly the same time the mode-1 Rth singular value started to drop drastically. It remains unclear whether this behaviour may indicate convergence to a rank-deficient solution or if it is just a coincidence. We did not observe a similar pattern in the λ weights in a sample of other trials with similar behaviours in the singular values. The behaviour also is not explained by standard techniques in optimization, e.g. unsatisfied convergence criteria, or machine learning, e.g. large gradient norms. The Rth largest singular value of the mode-1 unfolding is a pronounced indicator of convergence to a rank-deficient solution.

We observed this behaviour in most cases. Recall from Table 1 that CPAPR converged to the MLE in 106,215 of 110,266 trials ($\widehat{P}(\epsilon = 10^{-4}) = 0.963$). For a random sample of 10,000 of these trials,[7] the Poisson CP models were not rank-deficient: the Rth largest singular value when CPAPR terminated was typically far from machine precision (4.449 on average). By contrast, in 4,020 of the 4,051 trials (99.235%) where CPAPR converged to a different KKT point, we found that the Rth largest singular value in mode-1 when CPAPR terminated was on the order of double precision machine epsilon (i.e. $\approx 2.2204 \times 10^{-16}$). Thus we describe these models as rank-deficient: when the Rth largest singular value in one or more modes is numerically close to 0 (i.e. near or below machine precision) so that the column rank of these mode unfoldings is less than R. We can reasonably conclude that there is a strong connection between rank-deficient solutions and KKT points that are not the MLE.

5.3. *Restarted CPAPR with the* SVDROP heuristic

Upon closer examination of the traces from the rank-deficient search path in Figure 6, we notice that the Rth largest singular value in mode-1 drops dramatically between the 29th and 30th iteration. (In this case, $l_{max} = 5$, so the 30th total iteration is the first inner iteration on mode-1 of the 3rd outer iteration.) To be precise, the *gap ratio* of the Rth largest singular value from the mode-1 unfolding between iterations 29 and 30 is $\sigma_{(1)}[R]^{(29)}/\sigma_{(1)}[R]^{(30)} \approx 3.6 \times 10^{10}$. Considering all of the failed trials, the maximum gap ratio was 1.55×10^{12} on average, the median of the maximum gap ratios was 2.95×10^{6}, and the median iterate where it was observed was the 30th iteration. Analogous to the indication of numerical instability by large condition number, we interpret a large gap ratio between successive iterates as indicative of a search path that will converge to a rank-deficient solution. Therefore, large gap ratio may serve as a reliable heuristic to determine

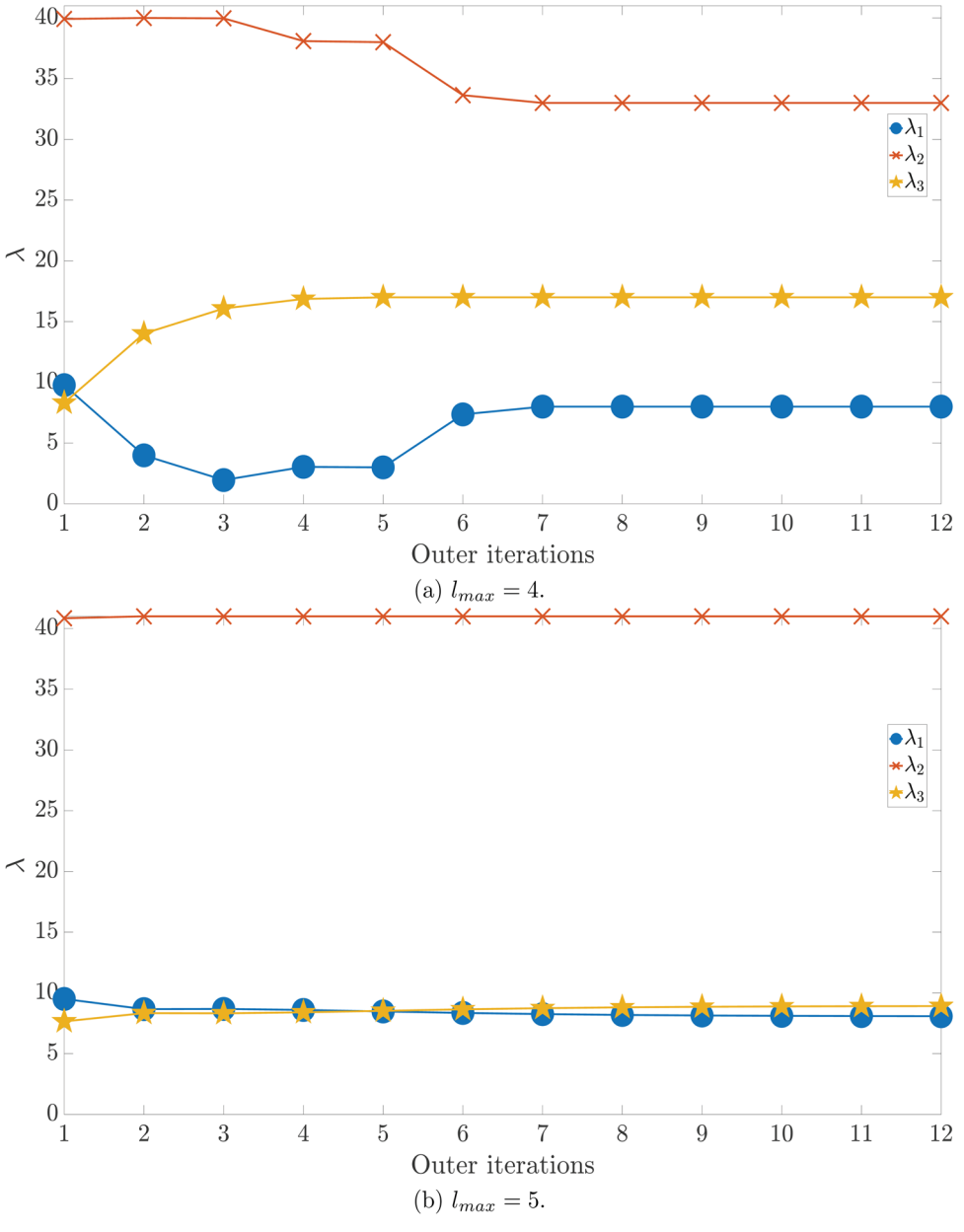


Figure 7. Traces of the R λ weights maintained by CPAPR in each iteration for the *exemplar trial* on `LowRankSmall`. It is unclear whether the λ-weights provide a useful heuristic for determining convergence to the MLE versus a rank-deficient solution. The x-axis in both plots has been truncated to the total number of outer iterations needed to converge to the MLE; the traces shown in the lower plot continue until convergence to the local minimizer without change. (a) $l_{max} = 4$. (b) $l_{max} = 5$.

whether the current search path should be accepted or rejected. This observation informs our solution in Algorithm 2. Before we present the method in full, two core concepts remain to be discussed: (1) the SVDrop heuristic and (2) Restarted CPAPR.

In our experiments, we use the MATLAB dense `svd` solver to compute gap ratios. Regardless of the SVD solver chosen, calculating the gap ratio between successive iterates will add non-trivial costs due to the SVD computation[8]. Furthermore, it is not even clear whether the gap ratio needs to be computed after every update. To mitigate incurring these excessive computational costs, we propose the SVDrop heuristic in a new subproblem for an extended CPAPR algorithm in Section 5.3.1. Additionally, in Section 5.5.3 we discuss how iterative SVD solvers (e.g. Krylov methods or randomized SVD) may be used to further reduce costs.

5.3.1. Procedure

The inputs to the subproblem algorithm are:

(1) the maximum number of inner iterations, $l_{max} \in \mathbb{N}_{>0}$;
(2) the number of SVDrop inner iterations between successive models for computation of their spectral properties, $\tau \in \mathbb{Z}_{+}$; and,
(3) a maximum threshold for the gap ratio indicating an acceptable search path, $\gamma \in \mathbb{R}_{+}$.

While the model is not converged in Algorithm 2, we compute a rank-R decomposition with CPAPR and improve the model fit with multiplicative updates and calls to Algorithm 3. The CPAPR subproblem loop begins at Line 5 in Algorithm 3. Every τ inner iterations, we compute the gap ratio between the current model and the checkpointed model from the singular values of the mode unfoldings at Lines 11–12. If the gap ratio computed is smaller than γ, then we accept the search path, checkpoint the model, and continue iterating. When the gap ratio is greater than γ, the SVDrop heuristic indicates that the search path will converge to a rank-deficient solution and we reject it, since to otherwise continue likely will waste a great deal of computation, as seen in Figures 1 and 6. A simple option is to restart: discard the work done up to now, randomly choose a new starting point in the feasible domain of the optimization problem, and recompute. Restarting, taken together with the SVDrop heuristic, is the idea behind our new method, Restarted CPAPR with SVDrop, presented in Algorithms 2 and 3. The exposition follows the template of the ideal version of CPAPR (see [22, Algorithms 1–2]). The symbol $\odot$ denotes the *matricized tensor times Khatri-Rao product (MTTKRP)* [69], an important computational kernel in many tensor decomposition algorithms. The symbol $\oslash$ denotes elementwise division. The symbol $*$ denotes elementwise matrix multiplication, i.e. the Hadamard product. We denote $\mathbf{e}$ as a vector of all ones with the appropriate dimension corresponding to mode n in Algorithms 2 and 3, $\lambda \in \mathbb{R}^d$ as the vector storing the weights of the Kruskal tensor $\mathcal{M}$ in Equation (3), and $\text{diag}(\lambda) \in \mathbb{R}^{d \times d}$ is a square matrix with the elements of λ along the diagonal and zeros everywhere else.

5.3.2. Additional considerations

First, we reset the counter of SVDrop inner iterations before the first inner iteration of a mode, since we observed that the singular values of the unfoldings of the modes that are held fixed change very little between iterations[9]. A consequence is that the number of SVDrop inner iterations should always be less than or equal to the maximum number of inner iterations, i.e. $\tau \leq l_{max}$. Second, the gap ratio tolerance γ should be sufficiently large. We used $\gamma = 10^6$ in our numerical experiments but it is unknown if this value generalizes

to other problems. Third, the update step Line 10 in Algorithm 3 is for exposition; it can be performed implicitly on $\mathcal{M}$ efficiently in software.

Algorithm 2 Restarted CPAPR algorithm with SVDropSubproblem.

Let $\mathcal{X}$ be a tensor of size $I_1 \times \cdots \times I_d$.
User input:
- R: Number of components in low-rank approximation
- l_{max}: Maximum number of inner iterations per outer iteration
- τ: Number of SVDrop inner iterations to perform
- γ : Tolerance for identifying rank-deficiencies (e.g. 10^6)

1: `restart` $\leftarrow$ true
2: **repeat** {SVDrop}
3: **if** (`restart`) **then**
4: $\mathcal{M} \leftarrow$ GenerateRandomGuess($[I_1, \ldots, I_d], R$)
5: **repeat** {OuterIteration}
6: **for** $n = 1, \ldots, d$ **do**
7: $\mathbf{\Pi} \leftarrow (\mathbf{A}^{(d)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)})^T$
8: [$\mathbf{B}$, `restart`] $\leftarrow$ SVDropSubproblem($\mathcal{M}, \mathbf{\Pi}, R, n, l_{max}, \tau, \gamma$)
9: **if** (`restart`) **then**
10: **break** ▷ Break out of {OuterIteration}.
11: $\boldsymbol{\lambda} \leftarrow \mathbf{e}^T\mathbf{B}$, $\mathbf{A}^{(n)} \leftarrow \mathbf{B}\,\mathrm{diag}(\boldsymbol{\lambda})^{-1}$
12: **until** convergence {OuterIteration}
13: **until** convergence {SVDrop}

5.4. Numerical experiments

To evaluate Restarted CPAPR with SVDrop, we selected 14,051 random initializations for `LowRankSmall` from the experiments in Section 4.2: the random sample of 10,000 starts where CPAPR converged to the MLE that we mentioned previously plus the 4,051 starts where CPAPR converged to a different KKT point. We computed rank $R = 3$ CP decompositions using CPAPR with Multiplicative Updates starting from each point. We increased the number of SVDrop inner iterations as $\tau \in \{0, \ldots, 10\}$ but kept all other parameters identical to the experiments in Section 4.2. A value $\tau = 0$ indicates that SVDrop was not used and that CPAPR was not restarted at any iteration.

5.4.1. Probability of convergence

In the previous experiments using CPAPR without any restarts, the total number of trials that did not converge to the MLE at the level of $\epsilon = 10^{-4}$ was 4,051 of 110,266 ($1 - \widehat{P}_{MLE}(10^{-4}) = 0.037$; see Table 1). Table 3 reports the total number of trials that: (1) converged to the MLE, (2) converged to some other KKT point, or (3) did not converge to any KKT point using this set of 4,051 initial starts. Convergence was calculated as in Equation (8) at the level of $\epsilon = 10^{-4}$ for each value of τ[10]. Of these, 3,905 converged to some other KKT point and 146 did not converge to any KKT point when $\tau = 0$. Our method improved on this in all cases. It is interesting to note that the probabilities

Algorithm 3 Subproblem solver for Algorithm 2 with SVDROP heuristic.

function SVDROPSUBPROBLEM(Kruskal tensor $\mathcal{M} = [\![\boldsymbol{\lambda}; \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}]\!]$, a sequence of Khatri-Rao products $\boldsymbol{\Pi}$, desired rank R, mode n, maximum number of inner iterations l_{max}, number of SVDROP inner iterations τ, maximum threshold for gap ratio γ)
  $t \leftarrow 0$
  $\sigma_R \leftarrow \text{svd}(\mathbf{M}_{(n)})[R]$ ▷ Rth largest singular value of $\mathbf{M}_{(n)}$.
  $\mathbf{B} \leftarrow \mathbf{A}^{(n)} \text{diag}(\boldsymbol{\lambda})$
  for $l = 1, \ldots, l_{max}$ **do**
    $t \leftarrow t + 1$
    $\boldsymbol{\Phi} \leftarrow (\mathbf{X}_{(n)} \oslash (\mathbf{B}\boldsymbol{\Pi}))\boldsymbol{\Pi}^T$
    $\mathbf{B} \leftarrow \mathbf{B} * \boldsymbol{\Phi}$
    if $(t = \tau)$ **then**
      $\boldsymbol{\lambda} \leftarrow \mathbf{e}^T\mathbf{B}$, $\mathbf{A}^{(n)} \leftarrow \mathbf{B}\,\text{diag}(\boldsymbol{\lambda})^{-1}$ ▷ Update $\mathbf{A}^{(n)}$, which updates $\mathcal{M}$.
      $\sigma_R^{(l)} \leftarrow \text{svd}(\mathbf{M}_{(n)})[R]$
      if $(\sigma_R/\sigma_R^{(l)} > \gamma)$ **then**
        return $[\mathbf{B}, \texttt{true}]$ ▷ $\text{rank}(\mathbf{M}_{(k)}) < R$; forces restart in Algorithm 2.
      $\sigma_R \leftarrow \sigma_R^{(l)}$
      $t \leftarrow 0$
  return $[\mathbf{B}, \texttt{false}]$ ▷ $\text{rank}(\mathbf{M}_{(k)}) = R$; does not force restart in Algorithm 2.

Table 3. Convergence results of Restarted CPAPR with SVDROP: the number of trials that converged to the MLE, converged to some other KKT point, or did not converge to a KKT point. The initial guesses were the set of starts from previous experiments where CPAPR without restarting (i.e. $\tau = 0$) did not converge to the MLE (number of starts $N = 4{,}051$). *Converged* means the solution satisfied the KKT-based CPAPR convergence criterion to tolerance 10^{-15}.

		SVDROPinner iterations τ										
Converged	Minimizer	0	1	2	3	4	5	6	7	8	9	10
Yes	MLE	0	4024	**4049**	4035	4028	4029	3906	3970	3983	3990	3998
Yes	Other KKT point	3905	0	0	0	0	0	102	43	31	24	20
No	–	146	27	**2**	16	23	22	43	38	37	37	33

of convergence to a different KKT point and failure to converge to any KKT point were higher when $\tau \geq 6$. We defer further discussion on this point to Section 5.5 since we will provide additional results that will help us reason about this behaviour and allow us to make suggestions with more context.

In the best case, when the number of SVDROP inner iterations was $\tau = 2$, Restarted CPAPR with SVDROP recovered the MLE in all but two trials ($\widehat{P}_{MLE}(10^{-4}) = 0.9995$). In these cases, SVDROP did not converge to a KKT point. Instead CPAPR oscillated near some other local minimizer—perhaps a saddle point. Figure 8 demonstrates that the failure of SVDROP in these two instances was due to setting the gap ratio tolerance γ too large (red $\times$ marker). In both plots, the gap ratio was always less than the choice of the tolerance in our experiments ($\gamma = 10^6$). When set appropriately, e.g. $\gamma = 8 \times 10^3$ (blue circle marker), Restarted CPAPR with SVDROP converged. Note that the traces were identical until the

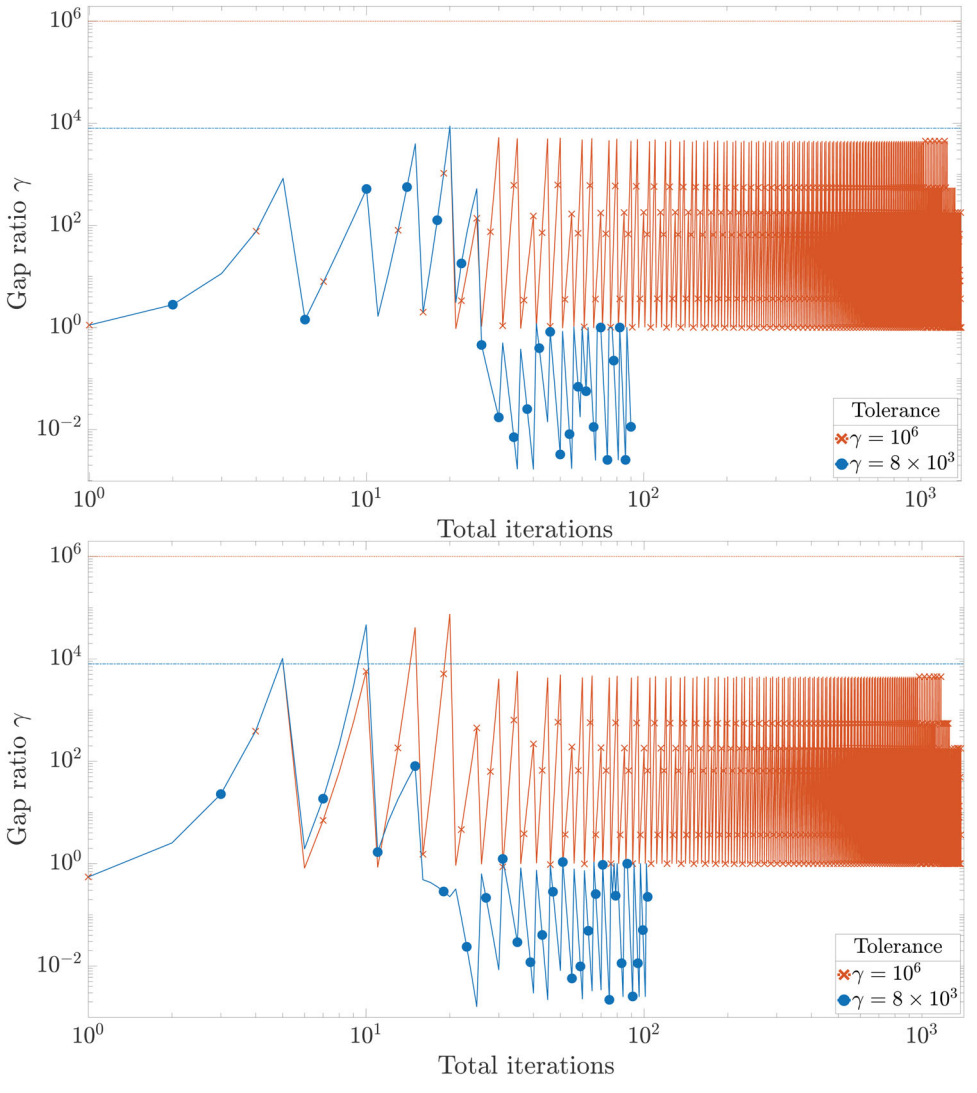


Figure 8. Two trials demonstrating the sensitivity of SVDROP to γ. Starting from points where CPAPR without restarting (i.e. $\tau = 0$) was known to converge to the MLE, SVDROP failed to converge to any KKT point when γ was set too large (red $\times$ marker). When γ is too large, SVDROP may fail to recognize rank deficiency and CPAPR may stagnate near a local minimizer that is not a KKT point. When set appropriately (blue circle marker), SVDROP converged to the MLE.

gap tolerance was exceeded in the $\gamma = 8 \times 10^3$ case. This triggered a restart, so that the iteration histories diverged. Observe that the trial in the bottom plot restarted twice when $\gamma = 8 \times 10^3$. Algorithm 2 will always restart until convergence to a KKT point that is not rank-deficient.

5.4.2. Computational cost

The formulae to compute computational costs in FLOPS are provided in Appendix 2. In the best case ($\tau = 2$), the cost of Restarted CPAPR with SVDROP was 7.04× higher than

CPAPR without restarting ($\tau = 0$). Although the cost of converging to the MLE using Restarted CPAPR with SVDrop was more expensive than CPAPR without restarting, it may be possible to converge to the MLE with higher probability—$\widehat{P}_{MLE}(10^{-4}) = 0.9995$ versus $\widehat{P}_{MLE}(10^{-4}) = 0.963$—with the extra work.

5.4.3. Caveats

Although our results were promising, we caution that our approach may not always improve on CPAPR without restarting. There appears to be a 'Goldilocks' range for the gap ratio: too large γ may decrease the probability that rank-deficient solutions are identified and too small γ may trigger unwanted restarts.

5.4.3.1. Choosing γ too large. Starting from the random sample of 10,000 points where CPAPR without restarting converged to the MLE in previous experiments, SVDrop occasionally performed worse. When the number of SVDrop inner iterations was $\tau = 1$, 34 trials did not converge to a KKT point. When the number of SVDrop inner iterations was $\tau = 8$, one trial converged to a KKT point that was not the MLE. In that trial, a restart was triggered when the gap ratio was greater than 10^6. However, two gap ratios, which were large ($>2 \times 10^4$) but below the threshold, were missed due to the tolerance having been set too large.

5.4.3.2. Choosing γ too small. When starting from points known to converge to the MLE without restarting (i.e. $\tau = 0$), it is possible that SVDrop could misclassify a solution as rank-deficient and restart from a new initial guess, which might needlessly increase the work expended. We consider this unwanted behaviour since minimal computational cost, in addition to accuracy, is a desired characteristic of our algorithm. Figure 9 shows estimated probabilities that SVDrop would trigger an unwanted restart for a range of gap ratios γ. The implication is that choosing γ to be too small may hurt performance.

5.5. Discussion

The results for SVDrop presented here are limited to a small exemplar. We discuss below several questions that should be addressed before conclusions about the general efficacy of SVDrop can be considered.

5.5.1. Connections between τ, γ, and l_{max}

It is possible that the number of SVDrop steps should be bounded by half the number of inner iterations, $\tau \leq \lfloor l_{max}/2 \rfloor$. The reason being that there is a correspondence between τ, l_{max}, and the number of times the gap ratio is computed per set of inner iterations. To see this, suppose $l_{max} = 10$ and $\tau = 6$. If not converged when $l = 6$, then the gap ratio would be computed only once for that mode; any additional changes to the model variables beyond that inner iteration would not be captured by SVDrop. Since rank-deficiency might only be indicated by the singular values of only one mode, as we observed in Figure 6(a), it may not be apparent that the model had been driven even closer to a rank-deficient solution while the remaining modes were being optimized. If $\tau = 5$, then the gap ratio would be computed twice: first when $l = 5$ and again when $l = 10$. In this case, we would not miss critical changes. Thus it is possible that the gap ratio should be computed

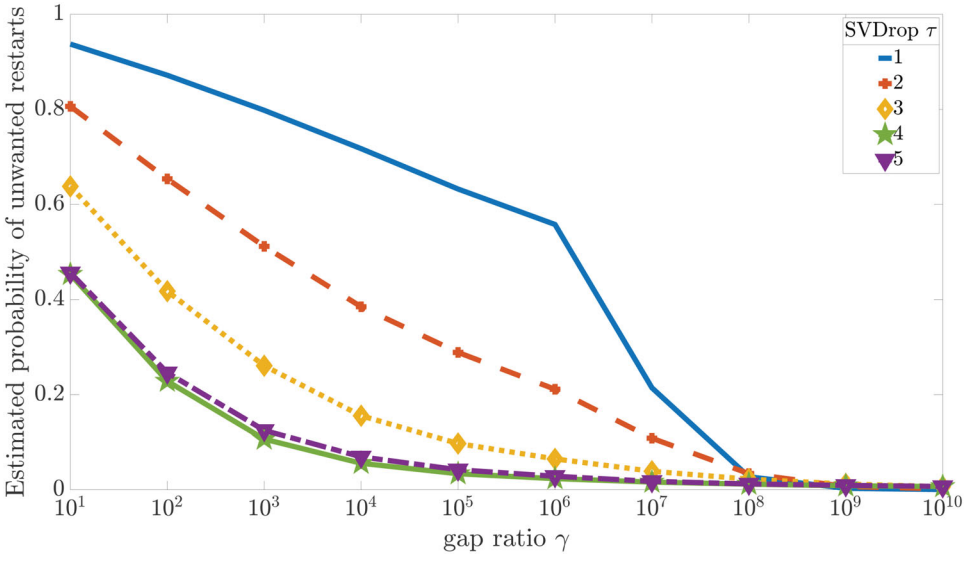


Figure 9. For trials that were known to converge to the MLE without using SVDROP, this figure reflects the estimated probabilities that SVDROP would trigger an unwanted restart for a range of gap ratios γ.

at least twice per set of inner iterations. Two possible options are to set $\tau = \lfloor l_{max}/2 \rfloor$ or to compute γ when $l = l_{max}$. We speculate that the choices of τ and γ are highly-coupled when considering both the probability of convergence to the MLE and the probability of unwanted restarting. We leave this investigation to future work.

5.5.2. Swamps

In future work, we will explore rank-deficiency more systematically for local minimizers of Poisson CPD problems to better understand the general applicability of our SVDROP approach. For example, in Figure 4, the CP model appears to be in a swamp[11] close to some local minimizer before emerging to converge to the MLE. One direction open to future work is to compare SVDROP with methods for avoiding 2FD such as those in [29].

5.5.3. Efficient computation of gap ratios

Black-box solvers, e.g. those built on LAPACK `xGES*D` [3], compute *all* of the singular values of the matrix. *Iterative methods*, such as subspace iteration or Krylov methods, may be a more practical choice for computing the gap ratio due to the (relatively) small number of nonzero entries in each unfolding (This is a consequence of the CPAPR algorithm design that drives elements toward zero.) Iterative methods are also useful when only a small number of singular values are needed and are amenable to sparse data structures. While it is less straightforward to characterize the computational costs and other trade-offs of iterative methods, it is worth investigating their role in future work.

6. Conclusions

We presented two new methods for Poisson canonical polyadic decomposition: HYBRIDGC and Restarted CPAPR with SVDROP.

6.1. HybridGC

Our method can minimize low-rank approximation error with high accuracy relative to GCP-Adam and CPAPR-MU while reducing computational costs. Since HybridGC was run in our experiments with a far stricter computational budget than GCP-Adam and CPAPR-MU, we argue that HybridGC can be more computationally efficient. The implication is that the performance gain allows even more multi-starts, and subsequently, a greater number of high accuracy approximations. Furthermore, our contribution is a new method that interpolates two very different algorithms.

One direction for future work is to study the impact on computation and accuracy of HybridGC when solver tolerances are relaxed. A second direction is related to the stagnation of HybridGC in Figure 1(b) near the local minimizer, which is indicative of a swamp [59]. We leave it to future work to compare HybridGC with methods designed to avoid swamps.

6.2. Restarted CPAPR with SVDrop

Our method identifies rank-deficient solutions with near-perfect accuracy and has the highest likelihood of finding the MLE in our experiments. A corollary is that our algorithm almost always avoids an entire class of minimizers that are different from the MLE. Our experimental results demonstrate this empirically with conservative budgets for both restarting and total iteration, alongside other untuned parameters. Provided more generous allotments and proper tuning, we expect SVDrop to always identify rank-deficient solutions in the limit of multi-starts.

One direction for future work is to extend the CPD ill-conditioning analysis by Breiding and Vannieuwenhoven [16–18] to the Poisson CPD problem addressed here. Such analysis could prove useful in providing a theoretical understanding of the rank-deficient solutions explored empirically in this paper.

6.3. Parameter tuning

Unlike SVDrop, HybridGC lacks a mechanism for changing its search path. Assuming a pattern behaviour exists, it is essential that further algorithm development uncovers a diagnostic to identify it. Otherwise, HybridGC will remain reliant on costly ad hoc parameter tuning by the user. Convergence and the computational cost of Restarted CPAPR with SVDrop both depend on the complex interplay of search parameters, which is not well-understood. Although we provided sensible values and rationalized upper bounds on some parameters, it remains an open question as to how sensitive SVDrop is to parameter variability. It is essential to better understand this interplay since SVDrop can be prohibitively expensive when it does fail. Fortunately, this is rare.

Notes

1. https://gitlab.com/tensors/tensor_toolbox.
2. https://github.com/sandialabs/pyttb.
3. https://github.com/sandialabs/sparten.
4. https://gitlab.com/tensors/genten.
5. By 'success', we mean convergence to the MLE. By 'failure' we mean convergence to any other KKT point or failure to converge to a KKT point.

6. The fairy tale of Goldilocks is about a girl named Goldilocks who enters a house belonging to three bears and tries out their belongings to find one that suits her best. Each item belonging to one of the three bears that she samples is either 'too many', 'too few', or 'just right'. Allusions to the Goldilocks fairy tale are also used by astronomers to describe the zone of habitable exoplanets around a star.
7. Computing this statistic for all random starts was prohibitively expensive.
8. When **A** is dense, the fastest *direct* or *transformation method* for serial or shared-memory parallel execution to compute all singular triplets of the matrix incurs a cost of $4mn^2 - \frac{4}{3}n^3$ floating point operations (FLOPS) or better [8,30,31,64] (assuming $m \geq n$). It is accessible through LAPACK's `xGESVD` driver [3].
9. It is possible that the singular values of the mode unfoldings held fixed may change. However, this was beyond the scope of this work.
10. Our results hold to the level of $\epsilon = 10^{-8}$ but we report $\epsilon = 10^{-4}$ to make comparison with the previous experiments with HybridGC without any restarts.
11. The term *swamp* was introduced by Mitchell and Burdick in [59]. A swamp is *a phenomenon… in which a [CP] sequence spends a long time in the vicinity of an inferior resolution before emerging and converging to an acceptable resolution*. Here we take 'acceptable resolution' to be the MLE and an 'inferior resolution' to be some other local minimizer.

Acknowledgments

We thank Eric Phipps of Sandia National Laboratories for assistance with Genten, a high-performance GCP solver. We thank the anonymous referees for their clarifications and helpful suggestions. This article has been authored by an employee of National Technology & Engineering Solutions of Sandia, LLC under Contract No. DE-NA0003525 with the U.S. Department of Energy (DOE). The employee owns all right, title and interest in and to the article and is solely responsible for its contents. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article or allow others to do so, for United States Government purposes. The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan https://www.energy.gov/downloads/doe-public-access-plan.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Notes on contributors

Jeremy M. Myers, Ph.D., is a Research Staff Member in the Scalable Modeling and Analysis Department of the Center for Computation and Analysis for National Security at Sandia National Laboratories in Livermore, CA, USA. His research interests include numerical linear algebra, machine learning, matrix and tensor decompositions, and high-performance computing.

Daniel M. Dunlavy, Ph.D., is a Distinguished Member of Technical Staff in the Machine Intelligence and Vis Department of the Center for Computing Research at Sandia National Laboratories in Albuquerque, NM, USA. He leads several research and development groups in the areas of machine learning, tensor decompositions and graph algorithms. In addition to his research, he also leads the development of several open-source software projects in these areas as well.

References

[1] E. Acar, D.M. Dunlavy, and T.G. Kolda, *A scalable optimization approach for fitting canonical tensor decompositions*, J. Chemom. 25(2) (2011), pp. 67–86. https://doi.org/10.1002/cem.1335.

[2] E. Acar, T.G. Kolda, and D.M. Dunlavy, *All-at-once optimization for coupled matrix and tensor factorizations*, arXiv. (2011). https://doi.org/10.48550/arXiv.1105.3422.
[3] E. Anderson, Z. Bai, C. Bischof, L.S. Blackford, J. Demmel, J.J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*. 3Society for Industrial and Applied Mathematics, Philadelphia, PA, United States, 1999. ISBN: 0-89871-447-8.
[4] C. Andersen and R. Bro, *The N-way toolbox for MATLAB*, Chemometr. Intell. Lab. Syst. 52(1) (2000), pp. 1–4. https://doi.org/10.1016/S0169-7439(00)00071-X.
[5] C. Andersen and R. Bro, *Practical aspects of PARAFAC modeling of fluorescence excitation-emission data*, J. Chemom. 17(4) (2003), pp. 200–215. https://doi.org/10.1002/cem.790.
[6] B. Bader and T. Kolda, *Efficient MATLAB computations with sparse and factored tensors*, SIAM J. Sci. Comput. 30(1) (2008), pp. 205–231. https://doi.org/10.1137/060676489.
[7] B.W. Bader, T.G. Kolda, and others, Tensor Toolbox for MATLAB, Version 3.6, 2023. www.tensortoolbox.org.
[8] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst (eds.), *Templates for the Solution of Algebraic Eigenvalue Problems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.
[9] M. Baskaran, B. Meister, and R. Lethin, *Low-overhead load-balanced scheduling for sparse tensor computations*, in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, Waltham, MA, USA, 2014, pp. 1–6.
[10] M. Baskaran, T. Henretty, B. Pradelle, M.H. Langston, D. Bruns-Smith, J. Ezick, and R. Lethin, *Memory-efficient parallel tensor decompositions*.2017 IEEE High Performance Extreme Computing Conference (HPEC). Waltham, MA, USA, 2017.
[11] M.M. Baskaran, T. Henretty, J. Ezick, R. Lethin, and D. Bruns-Smith, *Enhancing network visibility and security through tensor analysis*, Future Gener. Comput. Syst. 96 (2019), pp. 207–215.
[12] M. Baskaran, T. Henretty, and J. Ezick, *Fast and scalable distributed tensor decompositions*, in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, Waltham, MA, USA, 2019, pp. 1–7.
[13] M. Baskaran, D. Leggas, B. von Hofe, M.H. Langston, J. Ezick, and P.D. Letourneau, ENSIGN, [Computer Software] (2022). https://doi.org/10.11578/dc.20220120.1.
[14] J.A. Bazerque, G. Mateos, and G.B. Giannakis, *Inference of Poisson count processes using low-rank tensor data*.2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 2013.
[15] M.W. Berry, M. Browne, and B.W. Bader, *Discussion tracking in enron email using PARAFAC*, in *Survey of Text Mining II*, Springer, London, 2008, pp. 147–163.
[16] P. Breiding and N. Vannieuwenhoven, *A riemannian trust region method for the canonical tensor rank approximation problem*, SIAM J. Optim. 28 (2018), pp. 2435–2465.
[17] P. Breiding and N. Vannieuwenhoven, *Convergence analysis of riemannian Gauss–Newton methods and its connection with the geometric condition number*, Appl. Math. Lett. 78 (2018), pp. 42–50. Available at https://www.sciencedirect.com/science/article/pii/S089396591730318X
[18] P. Breiding and N. Vannieuwenhoven, *The condition number of join decompositions*, SIAM J. Matrix Anal. Appl. 39 (2018), pp. 287–309.
[19] R. Bro, *Multiway analysis in the food industry. Models, algorithms and applications*, Ph.D. diss., University of Amsterdam, 1998.
[20] J.D. Carroll and J.J. Chang, *Analysis of individual differences in multidimensional scaling via an N-way generalization of 'Eckart-Young' decomposition*, Psychometrika 35 (1970), pp. 283–319.
[21] P.A. Chew, B.W. Bader, T.G. Kolda, and A. Abdelali, *Cross-language information retrieval using PARAFAC2*.13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, CA, USA, 2007.
[22] E.C. Chi and T.G. Kolda, *On tensors, sparsity, and nonnegative factorizations*, SIAM J. Matrix Anal. Appl. 33 (2012), pp. 1272–1299.
[23] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, J. Mach. Learn. Res. 12 (2011), pp. 2121–2159.

[24] D.M. Dunlavy, T.G. Kolda, and W.P. Kegelmeyer, *Multilinear algebra for analyzing data with multiple linkages*, in *Graph Algorithms in the Language of Linear Algebra*, Kepner J. and Gilbert J., ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2011.
[25] D.M. Dunlavy, T.G. Kolda, and E. Acar, *Temporal link prediction using matrix and tensor factorizations*, ACM Trans. Knowl. Discov. Data 5 (2011), p. 1–27.(27 pages).
[26] H.C. Edwards, C.R. Trott, and D. Sunderland, *Kokkos: enabling manycore performance portability through polymorphic memory access patterns*, J. Parallel Distrib. Comput. 74 (2014), pp. 3202–3216.
[27] J. Ezick, T. Henretty, M. Baskaran, R. Lethin, J. Feo, T.C. Tuan, C. Coley, L. Leonard, R. Agrawal, B. Parsons, and W. Glodek, *Combining tensor decompositions and graph analytics to provide cyber situational awareness at HPC scale*. 2019 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 2019.
[28] M.P. Friedlander and K. Hatz, *Computing non-negative tensor factorizations*, Optim. Methods Softw.23 (2008), pp. 631–647.
[29] P. Giordani and R. Rocci, *Remedies for degeneracy in candecomp/parafac*, in *Quantitative Psychology Research*, van der Ark L.A., Bolt D.M., Wang W.C., Douglas J.A., and Wiberg M., ed., Springer International Publishing, Cham, Switzerland, 2016. pp. 213–227.
[30] G. Golub and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, J. Soc. Ind. Appl. Math. Ser. B: Numer. Anal. 2 (1965), pp. 205–224.
[31] G.H. Golub and C.F. Van Loan, *Matrix Computations*, 4th ed., Johns Hopkins Studies in the Mathematical Sciences, The Johns Hopkins University Press, Baltimore, 2013.
[32] A. Gyorgy and L. Kocsis, *Efficient multi-start strategies for local search algorithms*, J. Artif. Intell. Res. 41 (2011), pp. 705–720.
[33] N. Halko, P.G. Martinsson, and J.A. Tropp, *Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions*, Siam Rev. 53 (2011), pp. 217–288.
[34] S. Hansen, T. Plantenga, and T.G. Kolda, *Newton-based optimization for Kullback–Leibler nonnegative tensor factorizations*, Optim. Methods Softw. 30 (2015), pp. 1002–1029.
[35] R.A. Harshman, *Foundations of the PARAFAC procedure: Models and conditions for an 'explanatory' multi-modal factor analysis*, in *UCLA Working Papers in Phonetics* 16. Department of Linguistics, Phonetics Laboratory, UCLA, Los Angeles, CA, USA, 1970. pp. 1–84.
[36] J. Henderson, J.C. Ho, A.N. Kho, J.C. Denny, B.A. Malin, J. Sun, and J. Ghosh, *Granite: diversified, sparse tensor factorization for electronic health record-based phenotyping*. 2017 IEEE International Conference on Healthcare Informatics (ICHI), Park City, UT, USA, 2017.
[37] T. Henretty, M. Baskaran, J. Ezick, D. Bruns-Smith, and T.A. Simon, *A quantitative and qualitative analysis of tensor decompositions on spatiotemporal data*. 2017 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 2017.
[38] T.S. Henretty, M.H. Langston, M. Baskaran, J. Ezick, and R. Lethin, *Topic modeling for analysis of big data tensor decompositions*. Disruptive Technologies in Information Sciences, Orlando, FL, USA, 2018.
[39] F.L. Hitchcock, *The expression of a tensor or a polyadic as a sum of products*, J. Math. Phys. 6 (1927), pp. 164–189. Available at https://onlinelibrary.wiley.com/doi/abs/10.1002/sapm192761164.
[40] J.C. Ho, J. Ghosh, S.R. Steinhubl, W.F. Stewart, J.C. Denny, B.A. Malin, and J. Sun, *Limestone: high-throughput candidate phenotype generation via tensor factorization*, J. Biomed. Inform. 52 (2014), pp. 199–211.
[41] J.C. Ho, J. Ghosh, and J. Sun, *Marble: High-throughput phenotyping from electronic health records via sparse nonnegative tensor factorization*, in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14*, Association for Computing Machinery, New York, NY, USA, 2014, pp. 115–124.
[42] D. Hong, T.G. Kolda, and J.A. Duersch, *Generalized canonical polyadic tensor decomposition*, SIAM Rev. 62 (2020), pp. 133–163.
[43] C. Hu, P. Rai, C. Chen, M. Harding, and L. Carin, *Scalable Bayesian non-negative tensor factorization for massive count data*, in *Machine Learning and Knowledge Discovery in Databases*,

A. Appice, P.P. Rodrigues, V. Santos Costa, J. Gama, A. Jorge, and C. Soares, eds., Springer International Publishing, Cham, 2015, pp. 53–70.
[44] C. Hu, P. Rai, and L. Carin, *Zero-truncated poisson tensor factorization for massive binary tensors*, in *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*. AUAI Press, Arlington, Virginia, USA, 2015, pp. 375–384.
[45] K. Huang and N.D. Sidiropoulos, *Kullback-Leibler principal component for tensors is not NP-hard*. 2017 51st Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 2017.
[46] L. Ingber, *Very fast simulated re-annealing*, Math. Comput. Model. 12 (1989), pp. 967–973.
[47] D.P. Kingma and J. Ba, *Adam: A method for stochastic optimization*. 3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA, 2015.
[48] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, *Optimization by simulated annealing*, Science 220 (1983), pp. 671–680.
[49] T. Kolda and B. Bader, *The TOPHITS model for higher-order web link analysis*.Workshop onLink Analysis, Counterterrorism and Security, Bethesda, MD, USA, 2006.
[50] T.G. Kolda and B.W. Bader, *Tensor decompositions and applications*, SIAM Rev. 51 (2009), pp. 455–500.
[51] T.G. Kolda and D. Hong, *Stochastic gradients for large-scale tensor decomposition*, SIAM J. Math. Data Sci. 2 (2020), pp. 1066–1095.
[52] T. Kolda, B. Bader, and J. Kenny, *Higher-order web link analysis using multilinear algebra*. Fifth IEEE International Conference on Data Mining (ICDM), Houston, TX, USA, 2005.
[53] B. Korth and L.R. Tucker, *The distribution of chance congruence coefficients from simulated data*, Psychometrika 40 (1975), pp. 361–372.
[54] B. Korth and L.R. Tucker, *Procrustes matching by congruence coefficients*, Psychometrika 41 (1976), pp. 531–535.
[55] J.B. Kruskal, R.A. Harshman, and M.E. Lundy, *How 3-MFA data can cause degenerate parafac solutions, among other relationships*, in *Multiway Data Analysis*, North-Holland Publishing Co., NLD, 1989, pp. 115–122.
[56] P.D. Letourneau, M. Baskaran, T. Henretty, J. Ezick, and R. Lethin, *Computationally efficient CP tensor decomposition update framework for emerging component discovery in streaming data*. 2018 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 2018.
[57] U. Lorenzo-Seva and J. Ten Berge, *Tucker's congruence coefficient as a meaningful index of factor similarity*, Methodology 2 (2006), pp. 57–64.
[58] R. Martí, P. Pardalos, and M. Resende, eds. *Handbook of Heuristics*, Springer International Publishing, Cham, Switzerland, 2018.
[59] B.C. Mitchell and D.S. Burdick, *Slowly converging parafac sequences: swamps and two-factor degeneracies*, J. Chemom. 8 (1994), pp. 155–168.
[60] J. Mocks, *Topographic components model for event-related potentials and some biophysical considerations*, IEEE Trans. Biomed. Eng. 35 (1988), pp. 482–484.
[61] J.M. Myers and D.M. Dunlavy, *Using computation effectively for scalable Poisson tensor factorization: Comparing methods beyond computational efficiency*. 2021 IEEE High Performance Extreme Computing Conference (HPEC), Virtual, 2021.
[62] J.M. Myers, D.M. Dunlavy, K. Teranishi, and D.S. Hollman, *Parameter sensitivity analysis of the sparten high performance sparse tensor decomposition software*. 2020 IEEE High Performance Extreme Computing Conference (HPEC), Virtual, 2020.
[63] I.J. Myung, *Tutorial on maximum likelihood estimation*, J. Math. Psychol. 47 (2003), pp. 90–100.
[64] B. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.
[65] A.H. Phan, P. Tichavský, and A. Cichocki, *Low complexity damped Gauss–Newton algorithms for CANDECOMP/PARAFAC*, SIAM J. Matrix. Anal. Appl. 34 (2013), pp. 126–147.
[66] E.T. Phipps and T.G. Kolda, *Software for sparse tensor decomposition on emerging computing architectures*, SIAM J. Sci. Comput. 41 (2019), pp. C269–C290.
[67] P. Rai, C. Hu, M. Harding, and L. Carin, *Scalable probabilistic tensor factorization for binary and count data*. 24th International Conference on Artificial Intelligence (ICJAI), Tokyo, Japan, 2015.

[68] T.M. Ranadive and M.M. Baskaran, *An All-at-Once CP decomposition method for count tensors*. 2021 IEEE High Performance Extreme Computing Conference (HPEC), Virtual, 2021.
[69] A.K. Smilde, R. Bro, and P. Geladi, *Multi-Way Analysis with Applications in the Chemical Sciences*, J. Wiley, Chichester, West Sussex, England; Hoboken, NJ, 2004.
[70] M. Sugiyama, H. Nakahara, and K. Tsuda, *Legendre decomposition for tensors*. Thirty-second Annual Conference on Neural Information Processing Systems (NeurIPS), Montreal, QC, Canada, 2018.
[71] K. Teranishi, D.M. Dunlavy, J.M. Myers, and R.F. Barrett, *SparTen: Leveraging kokkos for on-node parallelism in a second-order method for fitting canonical polyadic tensor models to Poisson data*. 2020 IEEE High Performance Extreme Computing Conference (HPEC), Virtual, 2020.
[72] M. Vandecappelle, N. Vervliet, and L.D. Lathauwer, *A second-order method for fitting the canonical polyadic decomposition with non-least-squares cost*, IEEE Trans. Signal Process. 68 (2020), pp. 4454–4465.
[73] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, *A new truncation strategy for the higher-order singular value decomposition*, SIAM J. Sci. Comput. 34 (2012), pp. A1027–A1052.
[74] N. Vervliet, O. Debals, and L. De Lathauwer, *Tensorlab 3.0–numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization*. 50th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, 2016.
[75] S.J. Wright, *Coordinate descent algorithms*, Math. Program. 151 (2015), pp. 3–34.

Appendices

Appendix 1. `LowRankSmall` synthetic data tensor and empirical maximum likelihood estimator from numerical experiments

Sparse Tensor 1. Synthetic data tensor `LowRankSmall` used in experiments in Section 4.2.

```
sptensor  %% tensor type
3       % number of dimensions
4 6 8     % sizes of dimensions
17      % number of nonzeros
1 4 1 1   % start of sparse tensor data: <mode-1-index> <mode-2-index> <mode
    -3-index> <value>
1 4 6 5
1 4 7 9
1 6 6 1
1 6 7 1
2 1 2 6
2 1 4 5
2 1 8 3
2 4 4 1
2 5 2 1
2 6 2 6
2 6 4 8
2 6 8 3
4 1 8 4
4 2 1 1
4 2 8 2
4 5 8 1
```

MLE Kruskal Tensor 2. Kruskal tensor empirical maximum likelihood estimator (MLE) of Listing 1 computed in experiments in Section 4.2. The empirical MLE is the solution with the smallest Poisson loss from among 110,226 random starts. The λ values in the Kruskal tensor below are unnormalized in rational form for simplicity of presentation; in practice, the λ values should be converted to the appropriate system-dependent floating point format.

```
ktensor      %% tensor type
3          % number of dimensions
4 6 8        % sizes of dimensions
3          % number of components
1/33 1/17 1/8   % lambda values
matrix       %% 1st factor matrix
2          % number of dimensions
4 3         % sizes of dimensions
0 1 0        % start of data
1 0 0
0 0 0
0 0 1
matrix       %% 2nd factor matrix
2          % number of dimensions
6 3         % sizes of dimensions
14 0 4       % start of data
0 0 3
0 0 0
1 15 0
1 0 1
17 2 0
matrix       %% 3rd factor matrix
2          % number of dimensions
8 3         % sizes of dimensions
0 1 1        % start of data
13 0 0
0 0 0
14 0 0
0 0 0
0 6 0
0 10 0
6 0 7
```

Appendix 2. Computational cost derivations

A.1 CPAPR-MU operation count

CPAPR-MU was the first algorithm developed for Poisson CPD and it is important to assess its cost in terms of the number of floating point operations (FLOPS) required. We are only interested in sparse tensor computations, so we do not consider the costs of operations for dense tensors.

The work in an iteration of CPAPR-MU is dominated by the following operations:

(1) Sequence of Khatri-Rao products: $\Pi \leftarrow (\mathbf{A}^{(d)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)})^T$
(2) Implicit MTTKRP: $\Phi \leftarrow (\mathbf{X}_{(k)} \oslash (\mathbf{B}\Pi))\Pi^T$
(3) Multiplicative update: $\mathbf{B} \leftarrow \mathbf{B} * \Phi$

The first line is a sequence of Khatri-Rao products, where the binary operator $\odot$ denotes the Khatri-Rao product between matrices. The Khatri-Rao product is a primary component of the *matricized tensor times Khatri-Rao product* (MTTKRP), a key computational kernel in many tensor algorithms, not just CPD. Improving performance of the MTTKRP is a very active research area. For our purposes, we treat Khatri-Rao product as a black box and do not count its costs.

The second line computes the Φ matrix used in the multiplicative update. Note that $\mathbf{B}$ is simply the nth factor matrix $\mathbf{A}^{(n)}$ scaled by the λ weights, i.e. $\mathbf{B} = \mathbf{A}^{(n)}\,\mathrm{diag}(\lambda)$. We write that it is an implicit MTTKRP since (1) and (2) together are mathematically equivalent to MTTKRP. The MTTKRP is efficient because it can be done without forming dense arrays and the implicit MTTKRP can be performed even more efficiently due to the special structure of the matricized tensor in minimizing the Poisson loss for sparse tensors. The matrix multiplication $\mathbf{B}\Pi$ requires $\mathcal{O}(R\prod_{k=1}^{d} I_k)$ arithmetic.

The elementwise division $\mathbf{X}_{(k)} \oslash (\mathbf{B}\Pi)$ depends on the number of nonzeros in $\mathcal{X}$; thus it requires $\text{nnz}(\mathcal{X})$ operations. Lastly, the product $(\mathbf{X}_{(k)} \oslash (\mathbf{B}\Pi))\Pi^T$ requires $\mathcal{O}(R\prod_{k=1}^{d} I_k)$ arithmetic.

The elementwise multiplication in the third line, $\mathbf{B} * \Phi$, requires RI_n multiplications in the nth mode. All together, the number of FLOPS required per inner iteration for the nth mode is proportional to

$$\text{nnz}(\mathcal{X}) + rI_n + 2R\prod_{k=1}^{d} I_k \text{FLOPS}. \tag{A1}$$

Note: we ignored the following computations with negligible costs (with corresponding line numbers from [22, Algorithm 3]):

(1) inadmissible zero avoidance (Line 4);
(2) the shift of weights from Λ to mode-n and vice versa (Lines 5, 15, 16); and,
(3) the check for convergence (Line 9).

Appendix 3. Cyclic GCP-CPAPR

We develop Cyclic GCP-CPAPR (CyclicGC), a generalized form of HybridGC that *cycles* between a stochastic method to compute a model approximation and a deterministic method to resolve the model to the best accuracy possible at scale. In our formulation, HybridGC is CyclicGC with a single cycle. We define parameterizations and cycle *strategies*, which prescribe how CyclicGC iterates in each cycle.

Let $L \in \mathbb{N}$ be a number of *cycles*. Define *strategy* to be the L-length array of structures, `strat`, specifying the following for each cycle $l \in \{1, \ldots, L\}$:

- `S_opts`: stochastic search parameterization, including solver and search budget, j, measured in outer iterations.
- `D_opts`: deterministic search parameterization, including solver and search budget, k, measured in outer iterations.

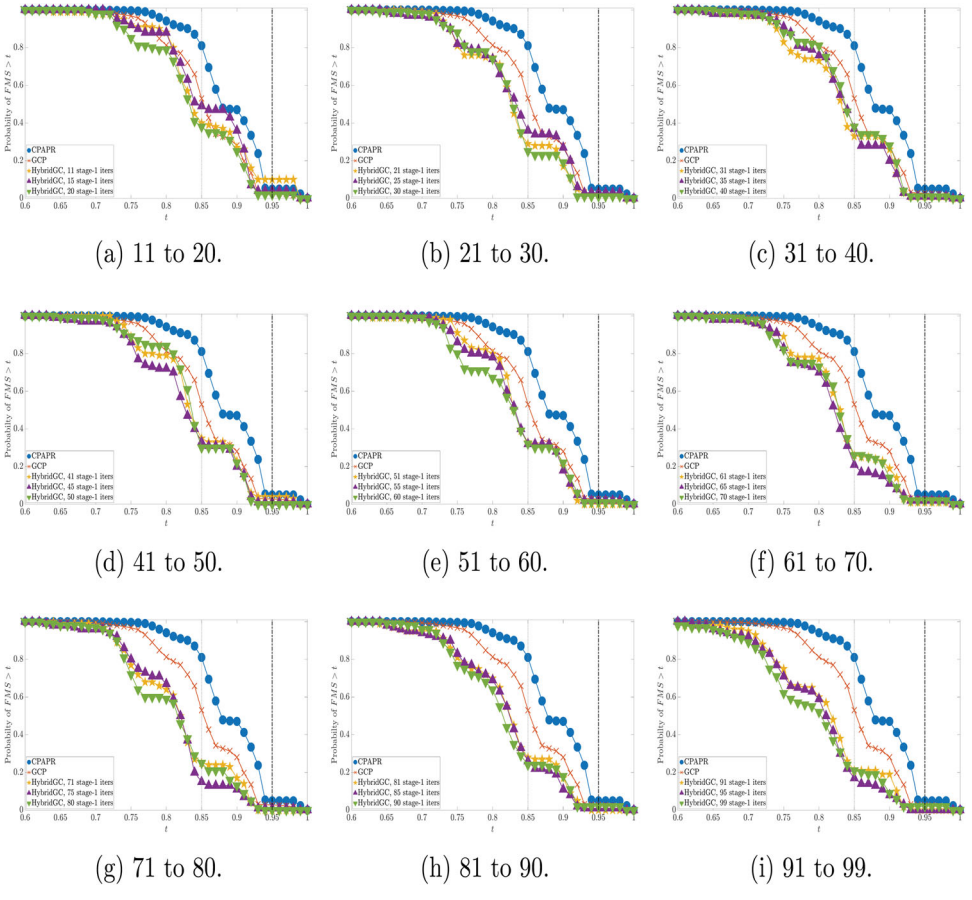
CyclicGC iterates from an initial guess $\mathcal{M}^{(0)}$ via a two-stage alternation between stochastic and deterministic search for L cycles to return a Poisson CP tensor approximation $\widehat{\mathcal{M}}$ as an estimate to $\mathcal{M}^*$. In the first stage of the lth cycle, the stochastic solver iterates from $\mathcal{M}^{(l-1)}$ for j outer iterations, parameterized by `strat(l).S_opts` to return an intermediate solution, $\mathcal{M}^{(l)}$. In the second stage, the deterministic solver refines $\mathcal{M}^{(l)}$ for k outer iterations, parameterized by `strat(l).D_opts`, to return the lth iterate, $\mathcal{M}^{(l)}$, overwriting the output from the previous stage.

Algorithm 4 Cyclic GCP-CPAPR

1: **function** CyclicGC(tensor $\mathcal{X}$, rank R, initial guess $\mathcal{M}^{(0)}$, number of cycles L, L-array of structures `strat` defining L strategies.)
2: **for** $l = 1, \ldots, L$ **do**
3: $\mathcal{M}^{(l)} \leftarrow$ GCP($\mathcal{X}$, R, $\mathcal{M}^{(l-1)}$ `strat(l).S_opts`)
4: $\mathcal{M}^{(l)} \leftarrow$ CPAPR($\mathcal{X}$, R, $\mathcal{M}^{(l)}$, `strat(l).D_opts`)
5: **return** model tensor $\widehat{\mathcal{M}} = \mathcal{M}^{(L)}$ as estimate to $\mathcal{M}^*$

Appendix 4. Supplemental numerical results

This section presents additional numerical results that are supplementary to those in Section 4.2.



(a) 11 to 20. (b) 21 to 30. (c) 31 to 40.

(d) 41 to 50. (e) 51 to 60. (f) 61 to 70.

(g) 71 to 80. (h) 81 to 90. (i) 91 to 99.

Figure A1. Factor match scores between CP models computed with HybridGC, CPAPR-MU, and GCP-Adam and the approximate global optimizer, $\widehat{\mathcal{M}}^*_{\mathcal{S}}$. The dash-dot grey vertical lines and dotted black vertical lines denote the levels of 'similar' and 'equal' described in [57]. Colormaps scaled for clarity. (a) 11 to 20. (b) 21 to 30. (c) 31 to 40. (d) 41 to 50. (e) 51 to 60. (f) 61 to 70. (g) 71 to 80. (h) 81 to 90. (i) 91 to 99.