Large–scale Sparse Tensor Decomposition Using a Damped Gauss–Newton Method

Teresa M. Ranadive
Laboratory for Physical Sciences
College Park, MD
tranadive@lps.umd.edu

Muthu M. Baskaran
Reservoir Labs, Inc.
New York, NY 10012
baskaran@reservoir.com

***Abstract*—CANDECOMP/PARAFAC (CP) tensor decomposition is a popular unsupervised machine learning method with numerous applications. This process involves modeling a high–dimensional, multi–modal array (a tensor) as the sum of several low–dimensional components. In order to decompose a tensor, one must solve an optimization problem, whose objective is often given by the sum of the squares of the tensor and decomposition model entry differences. One algorithm occasionally utilized to solve such problems is CP–OPT–DGN, a damped Gauss–Newton all–at–once optimization method for CP tensor decomposition. However, there are currently no published results that consider the decomposition of large–scale (with up to billions of non–zeros), sparse tensors using this algorithm. This work considers the decomposition of large–scale tensors using an efficiently implemented CP–OPT–DGN method. It is observed that CP–OPT–DGN significantly outperforms CP–ALS (CP–Alternating Least Squares) and CP–OPT–QNR (a quasi–Newton–Raphson all–at–once optimization method for CP tensor decomposition), two other widely used tensor decomposition algorithms, in terms of accuracy and latent behavior detection.**

***Index Terms*—Big data analytics, high performance computing, damped Gauss–Newton, sparse tensor decomposition**

I. Introduction

CANDECOMP/PARAFAC (CP) tensor decomposition, a popular unsupervised machine learning technique for data exploration and anomaly detection, involves approximating a high–dimensional, multi–modal array by the sum of several low–dimensional, multi–modal components. Roughly analogous to singular value decomposition or principle component analysis for matrices, this practice was first developed in [14] and [15], and found its first application in psychology [8]. Since then, CP tensor decomposition has been utilized in applications such as chemistry [3], criminology [12], medicine [32], text analysis [1], and cybersecurity [10], among many others. The process of decomposing a tensor involves solving a (typically large–scale) optimization problem, whose objective function is often given by the sum of the squares of the differences between corresponding entries of the tensor and the decomposition model. Although other, sometimes more appropriate, objective functions may be utilized in performing tensor decomposition [9], [16], our focus in this work is on minimizing the sum–of–squares, as (i) solving this problem often produces satisfying decomposition results for applications, and (ii) we can exploit the polynomial nature of this objective in order to reduce computational cost.

Two of the most popular algorithms for tensor decomposition with a sum–of–squares objective are CP–ALS [8] and CP–OPT–QNR [2]. CP–ALS, short for CP-Alternating Least Squares, is an algorithm in which we optimize over alternating sets of variables (each set corresponding to a certain tensor mode) by solving a least squares problem, while fixing the remaining sets of variables. CP–OPT–QNR is an all–at–once optimization approach that utilizes a quasi–Newton–Raphson method – the L–BFGS [21] algorithm. Less commonly used for minimizing the aforementioned objective is the tensor decomposition algorithm CP–OPT–DGN [28], [29], [36], which, like CP–OPT–QNR, is an all–at–once optimization algorithm, but, unlike CP–OPT–QNR, uses a damped Gauss–Newton / Levenberg–Marquardt [20], [23] approach.

In this work we implement and integrate CP–OPT–DGN into ENSIGN [31], a commercially available tensor decomposition software package. This package provides state–of–the–art high–performance C implementations of a number of tensor decomposition methods including CP–ALS and CP–OPT–QNR. ENSIGN tensor methods use optimized sparse tensor data structures, namely, mode–specific sparse (MSS) and mode–generic sparse (MGS) tensor data structures [4]. These methods are parallelized and optimized for computation and memory efficiencies in shared–memory and distributed–memory systems [5], [6]. We utilize the shared–memory implementations of all tensor decomposition algorithms studied in this paper, as we have yet to implement distributed–memory versions of CP–OPT–QNR and CP–OPT–DGN.

Enabled by ENSIGN, we then apply CP–OPT–DGN to decompose four large–scale (with up to billions of non–zeros) sparse tensors, each generated from one of four different data sets. We exploit the polynomial nature of the sum–of–squares objective in order to avoid the unnecessary computational expense required to compute the objective value and gradient at each trial step length, while performing a line search, and integrate similar techniques into ENSIGN's CP–OPT–QNR routine. (Note that related techniques have been used in the past for both sparse [34] and dense [30] tensors. We modify the strategy from [30] to be compatible with sparse tensors; the strategy from [34] is less efficient). It is observed that CP–OPT–DGN computes significantly more accurate decompositions than the other two algorithms, without requiring much additional computing time. We further note that CP–

OPT–DGN discovers latent behaviors that CP–ALS and CP–OPT–QNR do not. This corroborates the results observed for smaller dense tensors [28], [29], [36]. To the best of our knowledge, this work is the first to apply CP–OPT–DGN to sparse tensors of this size and highlights the utility in using CP–OPT–DGN to decompose such tensors: although previous implementations of CP–ALS and CP–OPT–QNR allow for the decomposition of sparse tensors with up to billions of non-zeros [13], [17], previous state–of–the–art implementations of CP–OPT–DGN are used only to decompose smaller (up to 100 million entries) dense tensors, even when sparsity is encouraged on the decomposition [28], [29], [36].

This paper is organized as follows. In Section II, we first describe the notation used in this paper and provide relevant background information on tensors, their decompositions, and the three aforementioned algorithms. In Section III, we discuss the improvements we have made to naive implementations of CP–OPT–QNR and CP–OPT–DGN. Next, in Section IV, we compare the performance of the three tensor decomposition algorithms, in terms of decomposition accuracy, while in Section V we compare the latent behaviors detected by each algorithm. Finally, a summary is given in Section VI.

II. Notation and Background Information

In this section, we describe the notation used in this paper, and provide background information on tensors, CP tensor decomposition, and related algorithms. For a more thorough introduction to tensors and tensor decomposition, see [18].

A. Notation

For $N \in \mathbb{N}$, let $[N]$ and $[N]_0$ denote the sets $\{1, \ldots, N\}$ and $\{0, 1, \ldots, N\}$, respectively. Throughout this paper, vectors $\mathbf{v} \in \mathbb{R}^I$ are denoted by lowercase bold letters, with the i-th entry of $\mathbf{v}$ denoted as v_i. Similarly, matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ will be denoted by uppercase bold letters, with the (i,j)-th entry of $\mathbf{A}$ denoted as a_{ij} and the i-th row of $\mathbf{A}$ as $\mathbf{A}_{i\bullet}$. For $\mathbf{A} \in \mathbb{R}^{I \times R}$, the matrix $\mathbf{\Upsilon}_{\mathbf{A}} := \mathbf{A}^T\mathbf{A} \in \mathbb{R}^{R \times R}$ denotes the Gramian matrix formed by the columns of $\mathbf{A}$, and $\mathbf{A}^\dagger$ denotes Moore–Penrose pseudoinverse of $\mathbf{A}$. If also $\mathbf{B} \in \mathbb{R}^{J \times R}$, then $\mathbf{A} \odot \mathbf{B} \in \mathbb{R}^{IJ \times R}$ denotes the Khatri–Rao product of $\mathbf{A}$ and $\mathbf{B}$, and if $I = J$, then $\mathbf{A} \star \mathbf{B}$ denotes the Hadamard product of $\mathbf{A}$ and $\mathbf{B}$. Finally, let $\mathbf{A}^{(\mathbf{n})} \in \mathbb{R}^{I_n \times R}$ for all $n \in [N]$, so that $\{\mathbf{A}^{(\mathbf{n})}\}_{n \in [N]}$ is a collection of matrices, each with R columns. For this collection, we use the following notation for multiple Khatri–Rao and Hadamard products:

$$\bigodot_{m \in [N] \setminus \{n\}} \mathbf{A}^{(\mathbf{m})} = \mathbf{A}^{(\mathbf{N})} \odot \cdots \odot \mathbf{A}^{(\mathbf{n+1})} \odot \mathbf{A}^{(\mathbf{n-1})} \odot \cdots \odot \mathbf{A}^{(\mathbf{1})},$$

$$\begin{aligned} \mathbf{\Gamma}_{\mathbf{A}^{(\mathbf{n})}} &= \bigstar_{m \in [N] \setminus \{n\}} \mathbf{\Upsilon}_{\mathbf{A}^{(\mathbf{m})}} \\ &= \mathbf{\Upsilon}_{\mathbf{A}^{(\mathbf{N})}} \star \cdots \star \mathbf{\Upsilon}_{\mathbf{A}^{(\mathbf{n+1})}} \star \mathbf{\Upsilon}_{\mathbf{A}^{(\mathbf{n-1})}} \star \cdots \star \mathbf{\Upsilon}_{\mathbf{A}^{(\mathbf{1})}}. \end{aligned}$$

B. Tensors

Tensors are objects from multi–linear algebra that generalize the concepts of vectors and matrices. For $N \in \mathbb{N}$, an N**-mode tensor** $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is an N-dimensional array of numbers. Observe, therefore, that a vector is a 1-mode tensor, while a matrix is a 2-mode tensor. In this paper, we denote the N-mode tensor $\mathcal{X}$ (with $N \geq 3$) with an uppercase calligraphic letter. Additionally, $x_{i_1,\ldots,i_N}$ denotes the $(i_1, \ldots, i_N)$-th entry of $\mathcal{X}$, consistent with the notation for vectors and matrices.

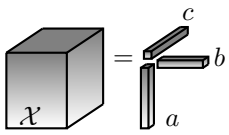

Fig. 1. A three mode, rank-1 tensor is the outer product of three vectors [18].

One may visualize a matrix in terms of its rows or columns. Similarly, one may visualize a tensor in terms of its **fibers**. Consider the N-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and fix $n \in [N]$. By fixing all of the indices of $\mathcal{X}$, except the index corresponding to the n-th mode, one obtains an n**-th mode fiber** of $\mathcal{X}$, e.g., $\mathbf{x}_{i_1 \ldots i_{n-1} \bullet i_{n+1} \ldots i_N} \in \mathbb{R}^{I_n}$. If one collects all of the n-th mode fibers of $\mathcal{X}$ and arranges them as the columns of a matrix, ordered by the indices corresponding to the remaining modes[1], one obtains $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times (I_1 \ldots I_{n-1} I_{n+1} \ldots I_N)}$, the matrix formed by **matricizing** $\mathcal{X}$ along its n-th mode. Matricizing tensors is an important concept in tensor decomposition, as the matrices $\mathbf{X}_{(1)}, \ldots, \mathbf{X}_{(N)}$ are required to write the gradient of the sum-of-squares objective function in a concise manner.

C. CP Tensor Decomposition

Just as a rank-1 matrix is any matrix that can be written as the outer product of two vectors, an N-mode, **rank-**1 **tensor** $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is any tensor that can be written as the outer product of N vectors $\mathbf{a}^{(\mathbf{1})} \in \mathbb{R}^{I_1}, \ldots, \mathbf{a}^{(\mathbf{N})} \in \mathbb{R}^{I_N}$ (cf. Fig. 1). Furthermore, for any $R \in \mathbb{N}$ with $R > 1$, a **rank-**R **tensor** $\mathcal{X}$ is any tensor that can be written as the sum of R rank-1 tensors, but not as the sum of $(R-1)$ rank-1 tensors (cf. Fig. 2). This sum of R outer products is termed a rank-R decomposition of $\mathcal{X}$. Note that the decomposition of a rank-R tensor is unique up to the scaling of the vectors forming the outer products (such that the outer products remain the same) and the reordering of the rank-1 summands. If $\mathcal{M} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is a rank-R tensor, such that vector along the n-th mode forming the r-th outer product summand of its decomposition is written as $\mathbf{a}_r^{(\mathbf{n})}$, for $r \in [R]$ and $n \in [N]$, we let $\mathbf{A}^{(\mathbf{n})}$ denote the n-th **factor matrix** of $\mathcal{M}$, whose columns are given by $\mathbf{a}_1^{(\mathbf{n})}, \ldots, \mathbf{a}_R^{(\mathbf{n})}$. We may then write the N-mode tensor $\mathcal{M}$ in terms of its factor matrices as

$$\mathcal{M} = [[\mathbf{A}^{(\mathbf{1})}, \ldots, \mathbf{A}^{(\mathbf{N})}]]. \tag{1}$$

By the Manifold Hypothesis [11], [26], high–dimensional data sets are often well approximated by low–dimensional models. Consequently, it has become a common practice to model data sets with multiple features as multi–modal tensors (each mode corresponding to one feature) and approximate such tensors by a rank-R tensor for some small $R \in \mathbb{N}$.

[1]Note that this column ordering is done first with respect to mode-N if $n \neq N$, then mode-(N-1) if $n \neq N-1$, and so on.

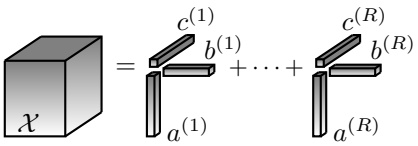

Fig. 2. A three mode, rank-R tensor is the sum of R rank-1 components [18].

The rank-1 tensors or **components** forming this decomposition often reveal latent behaviors possessed by the data set. In order to obtain this low–dimensional approximation, one must solve a (typically large–scale) optimization problem. For instance, one may minimize the sum–of–squares objective:

$$\underset{\text{rank}(\mathcal{M})\leq R}{\text{minimize}} \sum_{\mathbf{i}\in[I_1]\times\cdots\times[I_N]} (x_{\mathbf{i}} - m_{\mathbf{i}})^2/2, \text{ or equivalently,}$$
$$\underset{\text{rank}(\mathcal{M})\leq R}{\text{minimize}} \|\mathcal{X} - \mathcal{M}\|_F^2/2,$$

where $\| \star \|_F$ denotes the Frobenius norm. To significantly reduce the number of variables in this optimization problem and obtain a convex feasible region, one may reformulate this problem in terms of the factor matrices of $\mathcal{M}$ (cf. (1)):

$$\underset{\substack{\mathbf{A^{(n)}}\in\mathbb{R}^{I_n\times R}\\ \text{for } n\in[N]}}{\text{minimize}} 1/2 \times \Big\|\mathcal{X} - [[\mathbf{A^{(1)}},\ldots,\mathbf{A^{(N)}}]]\Big\|_F^2. \quad (2)$$

Penalty terms and/or constraints on the factor matrices (e.g., non–negativity constraints) may be added to improve decomposition interpretability and problem conditioning [13].

Related to the objective function in (2) is the notion of decomposition fit. Moreover, given a tensor $\mathcal{X}$ and a decomposition $\mathcal{M} := [[\mathbf{A^{(1)}},\ldots,\mathbf{A^{(N)}}]]$, the **fit** of the decomposition $\mathcal{M}$ to the tensor $\mathcal{X}$ is defined as

$$1 - \|\mathcal{X} - \mathcal{M}\|_F / \|\mathcal{X}\|_F.$$

This value describes how accurately the decomposition represents the tensor $\mathcal{X}$: the more accurate the decomposition, the lower the objective value is, and the closer the fit is to one.

D. Three Tensor Decomposition Algorithms

We now describe three different tensor decomposition algorithms that may be used to solve (2). We summarize the advantages and disadvantages of each algorithm in Section II-D4. The three algorithms, CP–ALS, CP–OPT–QNR, and CP–OPT–DGN are often abbreviated as ALS, QNR, and DGN, respectively, for the remainder of this paper.

1) ALS: CP–Alternating Least Squares is the oldest, most famous, tensor decomposition method [18]. ALS solves (2) by optimizing over one factor matrix (e.g., $\mathbf{A^{(1)}}$) via solving a least–squares problem, while fixing the remaining factor matrices (e.g., $\mathbf{A^{(2)}},\ldots,\mathbf{A^{(N)}}$); the algorithm loops over all factor matrices $\mathbf{A^{(n)}}, n \in [N]$ and alternatively optimizes (2) over each factor matrix $\mathbf{A^{(n)}}$ once during a single ALS iteration. Optimizing over $\mathbf{A^{(n)}}$ boils down to computing

$$\mathbf{A^{(n)}} = \mathbf{X_{(n)}}\Big(\bigodot_{m\in[N]\setminus\{n\}} \mathbf{A^{(m)}}\Big)\mathbf{\Gamma}_{\mathbf{A^{(n)}}}{}^{\dagger}$$

[18]. For computational efficiency, the product of $\mathbf{X_{(n)}}$ and $\bigodot_{m\in[N]\setminus\{n\}}\mathbf{A^{(m)}}$, commonly referred to as a MTTKRP (matricized tensor times Khatri–Rao product), is computed first, and then multiplied by $\mathbf{\Gamma}_{\mathbf{A^{(n)}}}{}^{\dagger}$. The most expensive step in obtaining the new value for $\mathbf{A^{(n)}}$ is typically computing this MTTKRP. When $\mathcal{X}$ is sparse with *nnz* non–zeros, the MTTKRP requires roughly $RN(nnz)$ flops. Note that N MTTKRPs are performed each iteration: each factor matrix, updated once per iteration, requires one MTTKRP.

2) QNR: Unlike ALS, in which (2) is optimized over alternating sets of variables, QNR attempts to optimize (2) over all variables at once, via the quasi–Newton–Raphson algorithm L–BFGS. L–BFGS is a line search algorithm that uses the objective value and gradient information from the current and previous M iterations, for some small fixed $M \in \mathbb{N}$, to obtain a quasi-Newton search direction. To find a step length, QNR uses the method from [25].

The partial derivative of $f : \mathbb{R}^{(I_1+\cdots+I_N)R} \to \mathbb{R}$, given by the objective function from (2), with respect to $\mathbf{A^{(n)}}$, is

$$\frac{\partial f}{\partial \mathbf{A^{(n)}}} = -\mathbf{X_{(n)}}\Big(\bigodot_{m\in[N]\setminus\{n\}} \mathbf{A^{(m)}}\Big) + \mathbf{A^{(n)}}\mathbf{\Gamma}_{\mathbf{A^{(n)}}}, \quad (3)$$

and requires performing one MTTKRP operation to compute. Therefore, if one were to naively implement the line search method from [25] as part of L–BFGS in QNR, it would be necessary to perform N (often expensive) MTTKRP operations for each trial step length. This is evident since (i) in order to determine whether or not a trial step length should be accepted during the line search method from [25], one must check whether or not the step length satisfies the strong Wolfe conditions, (ii) in a naive line search implementation, one would compute the gradient at the point corresponding to the current trial step length in order to check the strong Wolfe conditions, and (iii) each gradient computation requires N MTTKRP operations – one per factor matrix.

If many trial step lengths are tested, this process amasses a large computational expense. However, one may exploit the fact that the objective in (2) is a polynomial and acquire the same step length via the process described in Section III-A at a much lower computational cost. Moreover, once the search direction is obtained, the computational cost for determining a step length using this process is equivalent to about only $3N/2$ MTTKRP operations, regardless of the number of trial step lengths tested. Since obtaining the search direction requires one gradient evaluation, one may perform one iteration of QNR for a computational cost equivalent to roughly $5N/2$ MTTKRP operations; the remaining computational cost required for each QNR iteration is typically negligible, by comparison.

3) DGN: DGN attempts to optimize (2) over all variables at once like QNR, but uses a damped Gauss–Newton search direction instead of the quasi-Newton search direction. Consequently, DGN also requires N MTTKRP operations to compute the gradient (required to obtain the search direction) at the current iterate, and the equivalent of $3N/2$ additional MTTKRP operations to determine a step length once the search direction is computed, if the process from

Section III-A is employed. To quickly obtain the damped Gauss–Newton search direction in the DGN algorithm, we use techniques from [36]. These techniques require multiplying together matrices and solving linear systems of order R^2. Even though R is typically relatively small, e.g., $R = 50$ in our experiments, multiplying together matrices and solving linear systems of order R^2, requires $O(R^6)$ flops. Hence, obtaining the search direction may become an algorithm bottleneck if these subroutines are implemented inefficiently. We provide more details on quickly computing the damped Gauss–Newton search direction and our implementation of DGN in Sections III-B–III-C.

4) Algorithm Advantages and Disadvantages: In what follows, we summarize the advantages (denoted with a '+') and disadvantages (denoted with a '-') of the three tensor decomposition algorithms described above, as observed in the literature [2], [28], [29], [36]. Our aim is to examine whether or not these observations hold for large–scale sparse tensors.

ALS:

+ Easiest to implement. Fastest to converge.
- Obtains a worse fit than QNR and DGN. Can fail to detect latent behaviors detected by QNR. This is particularly true if the decomposition rank R is chosen too large.

QNR

+ Easier to implement than DGN. May result in a better fit than ALS and can detect behaviors that ALS does not.
- More difficult to implement/slower to converge than ALS.

DGN

+ Obtains a better fit than ALS.
- Most difficult of the three algorithms to implement. Slower to converge than ALS.

III. Efficient QNR and DGN Implementations

We now discuss our efficient QNR and DGN implementations in detail. These implementations offer several improvements over more naive implementions of these algorithms, including (i) a fast method for obtaining a suitable step length once a search direction is determined, for both QNR and DGN (cf. Section III-A), and (ii) enhancements for quickly obtaining the search direction for DGN by incorporating parallelized versions of LAPACKE subroutines (cf. Section III-B). We also discuss our choice of damping factor for DGN in Section III-C.

A. Efficiently Obtaining a Step Length

As discussed in Sections II-D2 and II-D3, once a search direction is obtained, one may compute a suitable step length for either QNR or DGN at a total cost roughly equivalent to $3N/2$ MTTKRP operations. This is in contrast to performing kN MTTKRP operations where $k \in \mathbb{N}$ is the number of trial step lengths, ranging from 2 to 17 in our experiments. We now discuss how to obtain the step length in this efficient way.

To check the strong Wolfe conditions and determine whether or not to accept a step length during a line search, one may consider the function $\varphi : \mathbb{R} \to \mathbb{R}$ of the step length given by

$$\varphi(\tau) := f\big([[\mathbf{A^{(1)}}, \ldots, \mathbf{A^{(N)}}]] + \tau[[\mathbf{D^{(1)}}, \ldots, \mathbf{D^{(N)}}]]\big),$$

where $f : \mathbb{R}^{(I_1+\cdots+I_N)R} \to \mathbb{R}$ is the objective function from (2), $\{\mathbf{A^{(n)}}\}_{n\in[N]}$ is the current iterate of factor matrices, and $\{\mathbf{D^{(n)}}\}_{n\in[N]}$ is the set of factor matrices determining the search direction. Note that since f is a polynomial, φ is a polynomial as well. Therefore, if one knows the polynomial coefficients of φ, one may quickly compute the value and derivative of this univariate function at each trial step length in place of computing the gradient of f; this allows us to check the Wolfe conditions at a notably reduced computational cost.

Calculating the coefficients of the aforementioned polynomial involves approximately the same computational expense as performing $3N/2$ MTTKRP operations, once the search direction is computed. To see why this is, note that

$$\varphi(\tau) := \varphi_1(\tau)/2 - \varphi_2(\tau) + \varphi_3(\tau)/2, \tag{4}$$

where $\varphi_1, \varphi_2, \varphi_3 : \mathbb{R} \to \mathbb{R}$, such that $\varphi_1(\tau) := \|\mathcal{X}\|_F^2$,

$$\varphi_2(\tau) := \langle \mathcal{X}, [[\mathbf{A^{(1)}} + \tau\,\mathbf{D^{(1)}}, \ldots, \mathbf{A^{(N)}} + \tau\,\mathbf{D^{(N)}}]]\rangle, \text{ and} \tag{5}$$

$$\varphi_3(\tau) := \|[[\mathbf{A^{(1)}} + \tau\,\mathbf{D^{(1)}}, \ldots, \mathbf{A^{(N)}} + \tau\,\mathbf{D^{(N)}}]]\|_F^2. \tag{6}$$

The value of the constant function φ_1 is computed at the beginning of QNR and DGN. It need not be recomputed here. To obtain the coefficients for φ_2, a degree N polynomial, one may use Algorithm 1, which requires roughly $3N^2R(\text{nnz})/2$ flops, the computational equivalent of $3N/2$ MTTKRP operations. Finally, to compute the coefficients of φ_3, a polynomial of degree $2N$, one may use Algorithm 2, which requires about $(N3^NR^2 + 4R^2\sum_{n=1}^{N} I_n)$ flops, a cost typically negligible compared to that required to obtain the coefficients of φ_2, as $3 \leq N \leq 5$, $R = 50$, $nnz \geq 4e7$, and $4R\sum_{n=1}^{N} I_n \ll N^2 \times nnz$ in most of of our experiments. Hence, after adding the coefficients of φ_1, φ_2, and φ_3 together, one obtains the coefficients of φ with a total cost roughly equivalent to $3N/2$ MTTKRP operations. Theorem 1 (resp. 2) in the Appendix ensures that Algorithm 1 (resp. 2) produces the correct coefficients for φ_2 (resp. φ_3).

B. Fast Computation of the Damped Gauss-Newton Direction

As discussed in Section II-D3, we must multiply matrices together and solve linear systems of order R^2, to obtain the damped Gauss–Newton direction. Since these matrix multiplication and linear system solver subroutines require on the order of R^6 flops, they must be implemented efficiently, as otherwise they may become a bottleneck for DGN, even with only moderately sized R. Therefore, in our DGN implementation, we utilize an OpenBLAS implementation of the LAPACKE functions `dgemm_` (for matrix multiplication) and `dgesv_` (for solving linear systems) to enable a fast, parallel computation of the damped Gauss–Newton direction. This allows us to obtain that direction in a negligible amount of time, when compared to computing the gradient (cf. (3)) and the polynomial coefficients of $\varphi(\tau)$ (cf. (4)–(6)).

C. Damping Factor for DGN

In our DGN implementation, we begin by utilizing the damped Gauss–Newton search direction with a damping factor

Algorithm 1 Computing the coefficients of φ_2 (cf. (5)).

Input: $\mathcal{X} \in [I_1] \times \cdots \times [I_N]$ is a sparse tensor; $\{\mathbf{A}^{(\mathbf{n})}\}_{n\in[N]}$ and $\{\mathbf{D}^{(\mathbf{n})}\}_{n\in[N]}$ are sets of suitably sized factor matrices.

Initialize $p_{2,n}$, the degree n coefficient of φ_2 to zero, for $n \in [N]_0$.
for each $\mathbf{i} = (i_1, \ldots, i_N)$ such that $x_{\mathbf{i}} \neq 0$ **do**
  Set $\pi^{(\mathbf{0})} = \mathbf{A}^{(\mathbf{1})}_{i_1\bullet}$.
  Set $\pi^{(\mathbf{1})} = \mathbf{D}^{(\mathbf{1})}_{i_1\bullet}$.
  for $m = 2, \ldots, N$ **do**
    Set $\pi^{(\mathbf{m})} = \pi^{(\mathbf{m-1})} \star \mathbf{D}^{(\mathbf{m})}_{i_m\bullet}$ ▷ R flops
    for $q = m-1, \ldots, 1$ **do**
      $\pi^{(\mathbf{q})} \leftarrow \pi^{(\mathbf{q-1})} \star \mathbf{D}^{(\mathbf{m})}_{i_m\bullet} + \pi^{(\mathbf{q})} \star \mathbf{A}^{(\mathbf{m})}_{i_m\bullet}$ ▷ $3R$ flops
    end for
    $\pi^{(\mathbf{0})} \leftarrow \pi^{(\mathbf{0})} \star \mathbf{A}^{(\mathbf{m})}_{i_m\bullet}$ ▷ R flops
  end for
  for $n = 0, 1, \ldots, N$ **do**
    $p_{2,n} \leftarrow p_{2,n} + x_{\mathbf{i}} \sum_{r=1}^{R} \pi^{(\mathbf{n})}_r$ ▷ $(R+1)$ flops
  end for
end for

Algorithm 2 Computing the coefficients of φ_3 (cf. (6)).

Input: $\{\mathbf{A}^{(\mathbf{n})}\}_{n\in[N]}$ and $\{\mathbf{D}^{(\mathbf{n})}\}_{n\in[N]}$ are sets of suitably sized factor matrices.

Initialize $p_{3,n}$, the n-th degree coefficient of φ_3 to zero, for $n \in [2N]_0$.
for $n = 1, \ldots, N$ **do**
  Set $\mathbf{M}^{(\mathbf{n})}_{\mathbf{0}} = \boldsymbol{\Upsilon}_{\mathbf{A}^{(\mathbf{n})}}$ ▷ $I_n R^2$ flops
  Set $\mathbf{M}^{(\mathbf{n})}_{\mathbf{1}} = (\mathbf{A}^{(\mathbf{n})})^T \mathbf{D}^{(\mathbf{n})} + (\mathbf{D}^{(\mathbf{n})})^T \mathbf{A}^{(\mathbf{n})}$ ▷ $2I_n R^2$ flops
  Set $\mathbf{M}^{(\mathbf{n})}_{\mathbf{2}} = \boldsymbol{\Upsilon}_{\mathbf{D}^{(\mathbf{n})}}$ ▷ $I_n R^2$ flops
end for
for $(j_1, \ldots, j_N) \in [2]_0 \times \cdots \times [2]_0$ **do**
  $\mathbf{M} = \bigstar_{n=1}^{N} \mathbf{M}^{(\mathbf{n})}_{j_n}$ ▷ NR^2 flops
  $J \rightarrow \sum_{n=1}^{N} j_n$ ▷ N flops
  $p_{3,J} \rightarrow p_{3,J} + \sum_{i,j=1}^{R} m_{ij}$ ▷ R^2 flops
end for

of one. Subsequent search directions use the same damping factor if the objective experienced a relative decrease of a certain threshold or more during the previous iteration. Otherwise, we use the negative gradient as the search direction, which is equivalent to letting the damping factor $\lambda \rightarrow \infty$. Although different versions of Levenberg–Marquardt offer more sophisticated strategies for choosing the damping factor, our strategy is easy to implement and produced excellent experimental results (cf. Sections IV and V).

IV. Decomposition Fit Over Time

We now study the decomposition of four different tensors, each of which is constructed from a real world data set. These data sets, described below, relate to a variety of applications, i.e., geospatial analysis, text analysis, and cybersecurity, so that we may demonstrate the versatility of DGN's effectiveness when applied to large–scale sparse tensors. A summary of the tensors formed from these data sets is given in Table I.

Data set	Dimensions	No. of Non-zeros
AIS	17,431 x 445 x 7,592,678	49,210,786
Amazon	4,821,207 x 1,774,269 x 1,805,187	1,741,809,018
Enron	6,066 x 5,699 x 244,268 x 1,176	54,202,099
LANL	1,433 x 22,077 x 534,687 x 58,389 x 11	40,266,345

TABLE I
TENSORS GENERATED FROM FOUR DIFFERENT DATA SETS.

The Automatic Identification System (AIS) data set contains vessel traffic data collected by the U.S. Coast Guard for monitoring the location and characteristics of large vessels in U.S. and international waters [22]. Our AIS data set is a subset of the AIS records recorded from UTM zones 14-19 from the years 2015-2017. The tensor was constructed using the ENSIGN tensor construction utility (`csv2tensor`). The tensor modes correspond to (i) time binned to the hour, (ii) latitude and longitude fused and binned to three decimal places, and (iii) Maritime Mobile Service Identity (MMSI).

The Amazon data set [24] was used to construct a tensor downloaded directly from [35]. The data set was formed from a corpus of reviews of products sold on `amazon.com`. After preprocessing the reviews to remove stop words, the number of times an Amazon user utilized a given word in review of a specific product was recorded. The constructed tensor therefore has modes corresponding to user, product, and word.

The Enron data set [33] was also used to construct a tensor downloaded directly from [35]. This data set was formed from a corpus of emails released by Enron Corporation during an investigation by the Federal Energy Regulatory Commission. Emails with a sender or recipient outside of the `@enron.com` domain were removed and the remaining emails were preprocessed by removing stopwords and using Porter stemming. The tensor has modes akin to sender, recipient, word, and date.

The LANL data set is a public cyber data set published by Los Alamos National Lab [37]. It contains 90 days of anonymized NetFlow and host events data. We used the NetFlow data from a single day to form the tensor for our analysis. The tensor, constructed using the ENSIGN `csv2tensor` utility, has modes corresponding to timestamp (binned by minute), source device, destination device, destination port, and number of bytes transferred (binned in logarithmic scale).

We decompose each of the four tensors ten different times (each time using a different initial guess) for each of the three different tensor decomposition algorithms, with nonnegativity constraints on all factor matrices. Each decomposition was run on a single node with 512GB of memory and 20 cores. For each tensor, we plot the objective value over time obtained by each of the decomposition algorithms during the ten trials, along with the average objective value over the ten trials, in Figure 3. Also, we record the the average time for each algorithm to converge and the best fit obtained over the ten trials for the four data sets in Table II. For all four tensors, we see that DGN reduces the average objective value from (2) the most out of all three algorithms and achieves the best fit over all ten trials, demonstrating that DGN outperforms the other two algorithms in terms of accuracy; furthermore, DGN converges on average in about the same time as QNR and in less than twice the time as ALS, for all four tensors (cf. Fig. 3 and Table II). Note that we declare that an algorithm

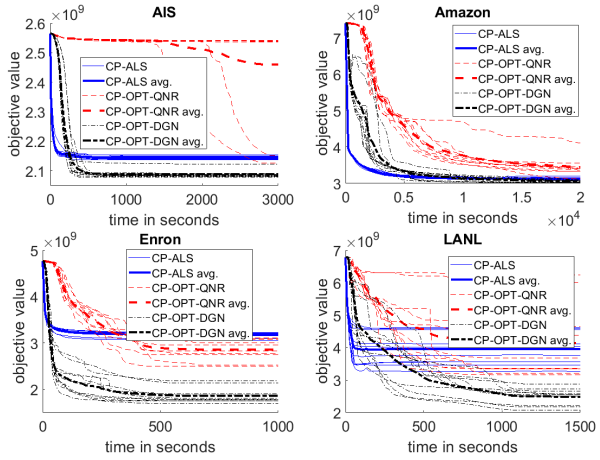

Fig. 3. Objective value (cf. (2)) vs. time in seconds for each tensor from Table I obtained by ALS, QNR, and DGN. We add nonnegativity constraints to (2). For each data set and algorithm, we plot the results for ten different initial guesses (thin lines), as well as the average of those results (thick lines). Convergence to a lower objective value indicates better decomposition accuracy. Hence, DGN demonstrates better accuracy than ALS and QNR.

	CP–ALS		CP–OPT–QNR		CP–OPT–DGN	
	time	fit	time	fit	time	fit
AIS	1.26 E 3	0.085	2.86 E 3	0.089	2.14 E 3	0.100
Amazon	1.72 E 4	0.343	2.13 E 4	0.352	2.14 E 4	0.361
Enron	9.34 E 2	0.207	5.34 E 2	0.275	5.62 E 2	0.405
LANL	6.77 E 2	0.417	8.78 E 2	0.317	1.18 E 3	0.447

TABLE II
AVERAGE TIME TAKEN TO CONVERGE IN SECONDS AND BEST FIT OBTAINED OVER TEN TRIALS

has converged if the objective value is not relatively reduced by at least a certain threshold (here 1e-10) or a maximum number of iterations is reached (here 100). In nearly all trials, the maximum number of iterations was reached, implying that the average time per iteration for each algorithm and data set is given roughly by the value in Table II divided by one hundred. Hence, in all of our experiments, DGN produces the most accurate tensor decompositions out of all three algorithms, without much additional computing time (both overall and per iteration), for large–scale sparse tensors. This corroborates the earlier results on smaller tensors (cf. Section II-D4).

V. Latent Behavior Detection

Finally, we see how DGN detects behaviors that ALS and QNR do not, while ALS and QNR often fail to detect any behaviors other than those detected by DGN. Due to space limitations, only latent behaviors found in the Enron data set will be discussed in detail. We note however, that DGN discovered three behaviors in the LANL data set not found by ALS, while ALS failed to detect any behaviors in addition to those detected by DGN. In all of our experiments, there were no instances in which (i) ALS (resp. QNR) detected behaviors undetected by DGN, and (ii) DGN failed to detect behaviors in addition to those discovered by ALS (resp. QNR). For information on latent behaviors detected in the AIS data set, see [19].

After computing ten different decompositions of the Enron tensor (cf. Section IV) with each of the three tensor

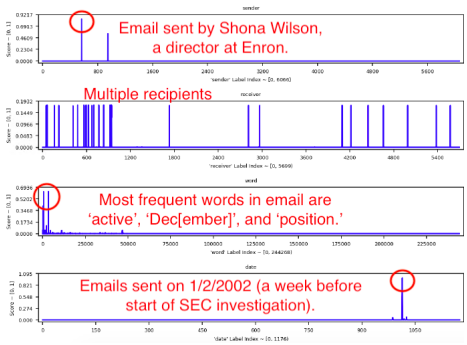

Fig. 4. Behavior detected by DGN, undetected by QNR and ALS.

decomposition algorithms, we select the decomposition that obtains the highest fit out of the ten, for each algorithm. Then, for each component in each decomposition, we measure the cosine similarity between that component, and the other two decompositions, noting the following: (i) There are three components from the DGN decomposition that have a cosine similarity of less than 1e-12 with the ALS decomposition and less than 1e-6 with the QNR decomposition, indicating that the behaviors described by these three components were not detected by either ALS or QNR. (ii) Each component from both the ALS and QNR decompositions has a cosine similarity of at least 1e-4 with the DGN decomposition (and in nearly all cases at least 1e-3), indicating that DGN detected all behaviors detected by the other two algorithms. The three behaviors detected only by DGN were determined to correspond to [27]: (i) an email accidentally sent by one employee to multiple coworkers containing many repetitions of the same line, (ii) a chain of emails on Enron buying Columbia Energy Services Corporation and having certain ISDA master agreements transferred to Enron during April–July of 2000, and (iii) emails related to calculating the value at risk to be reported to the U.S. Securities and Exchange Commission (SEC) sent out in early January 2002; this is particularly interesting as these emails were sent days before the SEC began its investigation of Enron. These emails were sent by a single person, a director at the company, and contained the words 'active', 'Dec[ember]', and 'position' (cf. Fig. 4). Hence, DGN outperforms ALS and QNR, not only in terms of accuracy, but in regards to latent behavior detection.

VI. Summary and Concluding Remarks

In this work, we implement a highly efficient, parallelized, shared–memory damped Gauss–Newton tensor decomposition algorithm, CP–OPT–DGN, into the tensor software package ENSIGN. We decompose several large–scale sparse tensors using this algorithm, and observe that this algorithm outperforms the decomposition algorithms CP–ALS and CP–OPT–QNR in terms of decomposition accuracy and latent behavior detection. Future research directions include implementing a distributed–memory version of CP–OPT–DGN and developing efficient damped Gauss–Newton all–at–once tensor decomposition algorithms that minimize other appropriate objective functions.

ACKNOWLEDGMENTS

The authors would like to thank Thomas Rolinger for his help with compiling the OpenBLAS library, as well as the anonymous referees for their efforts in reviewing this paper.

REFERENCES

[1] E. Acar, S. A. Camtepe, M. S. Krishnamoorthy, and B. Yener, "Modeling and multiway analysis of chatroom tensors," in *Proc. IEEE Int. Conf. Intell. and Secur. Inform.*, Atlanta, GA, USA, May 19–20, 2005, pp. 256–268.

[2] E. Acar, D. M. Dunlavy, and T. G. Kolda, "A scalable optimization approach for fitting canonical tensor decompositions," *J. Chemometrics*, vol. 25, no. 2, pp. 67–86, Feb. 2011.

[3] C. J. Appellof and E. R. Davidson, "Strategies for analyzing data from video fluorometric monitoring of liquid chromatographic effluents," *Anal. Chem.*, vol. 53, no. 13, pp. 2053–2056, Nov. 1981.

[4] M. Baskaran, B. Meister, N. Vasilache, and R. Lethin, "Efficient and scalable computations with sparse tensors," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, Waltham, MA, USA, Sept. 10–12, 2012.

[5] M. Baskaran et al., "Memory-efficient parallel tensor decompositions," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, Waltham, MA, USA, Sept. 12–14, 2017.

[6] M. Baskaran, T. Henretty, and J. Ezick, "Fast and scalable distributed tensor decompositions," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, Waltham, MA, USA, Sept. 24–26, 2019.

[7] R. Bro and S. De Jong, "A fast non-negativity constrained least squares algorithm," *J. Chemometrics*, vol. 11, pp. 393–401, 1997.

[8] J. D. Carroll and J.–J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[9] E. C. Chi and T. G. Kolda, "On tensors, sparsity, and nonnegative factorizations." *SIAM J. Matrix Anal. and Appl.*, vol. 33, pp. 1272–1299, 2013.

[10] J. Ezick et. al., "Combining tensor decompositions and graph analytics to provide cyber situational awareness at HPC scale," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, Waltham, MA, USA, Sept. 24–26, 2019.

[11] C. Fefferman, S. Mitter, and H. Narayanan, "Testing the manifold hypothesis," *J. Amer. Math. Soc.*, vol. 29, no. 2, pp. 983–1049, 2016.

[12] L. Ge, J. Liu, A. Zhou, and H. Li, "Crime rate inference using tensor decomposition," *IEEE SmartWorld, Ubiquitous Intell. & Comput., Adv. & Trusted Comput., Scalable Comput. & Commun. Cloud & Big Data Comput., Internet of People and Smart City Innov.*, Guangzhou, China, 2018, pp. 713–717.

[13] A. Gudibanda, T. Henretty, M. Baskaran, J. Ezick, and R. Lethin, "All-at-once decomposition of coupled billion-scale tensors in Apache Spark," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, Waltham, MA, USA, Sept. 25–27, 2018.

[14] F. L. Hitchcock, "The expression of a tensor or polyadic as a sum of products," *J. Math. Phys.*, vol. 6, pp. 164–189, 1927.

[15] F. L. Hitchcock, "Multilple invariants and generalized rank of a p–way matrix or tensor," *J. Math. Phys.*, vol. 7, pp. 39–79, 1927.

[16] D. Hong, T. G. Kolda, and J. A. Duersch, "Generalized canonical polyadic tensor decomposition," *SIAM Rev.*, vol. 62, no. 1, pp. 133–163, 2020.

[17] I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos, "HaTen2: Billion-scale Tensor Decompositions," *2015 IEEE 31st Int. Conf. on Data Eng.*, Seoul, 2015, pp. 1047–1058

[18] T. G. Kolda and B. W. Bader, "Tensor decomposition and applications," *SIAM Rev.*, vol. 51, no. 3, pp. 455-500, 2009.

[19] D. Leggas et. al., "Multiscale Data Analysis Using Binning, Tensor Decompositions, and Backtracking," *Proc. IEEE High Perform. Extreme Comput. Virtual Conf.*, Sept. 24–26, 2020.

[20] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quart. Appl. Math.*, vol. 2, pp. 164–168, 1944.

[21] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Math. Program.*, vol. 45, pp. 503–528, 1989.

[22] *Vessel traffic data*, MarineCadastre, 2015–2017. [Online]. Available: https://marinecadastre.gov/AIS/

[23] D. W. Marquardt, " An algorithm for least–squares estimation of non–linear parameters," *J. Soc. Indust. Appl. Math.*, vol. 11, no. 2, pp. 431-441, 1963.

[24] J. McAuley and J. Leskovec, "Hidden factors and hidden topics: understanding rating dimensions with review text," in *Proc. 7th ACM Conf. on Recommender Syst.*, 2013, pp. 165–172.

[25] J. J. More and D. J. Thuente, "Line search algorithms with guaranteed sufficient decrease," *ACM Trans. on Math. Software*, vol. 20, no. 3, pp. 286–307, 1994.

[26] H. Narayanan, "Sample complexity in manifold learning," in *Manifold Learning Theory and Applications*, Y. Ma and Y. Fu, Ed., Boca Raton, FL, USA: CRC Press, 2011, ch. 4.

[27] *ENRON Emails*, OCCRP Aleph, 2018. Accessed: Jun. 24, 2020. [Online]. https://aleph.occrp.org/datasets/1021

[28] A.-H. Phan, P. Tichavsky, and A. Cichocki, "Fast damped Gauss-Newton method for sparse and nonnegative tensor factorization," in *Proc. IEEE Int. Conf. on Acoust., Speech and Signal Process. (ICASSP)*, Prague, Czech Republic, 2011, pp. 1988–1991.

[29] A.-H. Phan, P. Tichavsky, and A. Cichocki, "Low complexity damped Gauss–Newton algorithms for CANDECOMP/PARAFAC," *SIAM J. Matrix Anal. and Appl.*, vol. 34, no. 1, pp. 126–147, 2013.

[30] M. Rajih, P. Comon, and R. A. Harshman, "Enhanced line search: a novel method to accelerate PARAFAC," *SIAM J. Matrix Anal. and Appl.*, vol. 30, no. 3, pp. 1148–1171, 2008.

[31] *ENSIGN Tensor Toolbox*, Reservoir Labs. Accessed: Jun. 24, 2020. [Online]. Available: https://www.reservoir.com/ensign-cyber/

[32] T. Schultz and H.-P. Seidel, "Estimating crossing fibers: a tensor decomposition approach," *IEEE Trans. on Visualization and Comput. Graph.*, vol. 14, no. 6, pp. 1635–1642, 2008.

[33] J. Shetty and J. Adibi, "The Enron email dataset database schema and brief statistical report," Univ. Southern California Inf. Sci. Inst., Los Angeles, CA, USA, 2004, vol. 4.

[34] N. D. Sidiropoulos et. al., "Tensor decomposition for signal processing and machine learning," *IEEE Trans. on Signal Process.*, vol. 65, no. 13, pp. 3551–3582, 2017.

[35] S. Smith et. al., *FROSTT: The formidable repository of open sparse tensors and tools*, 2017. [Online]. Available: http://frostt.io/

[36] P. Tichavsky, A.-H. Phan and A. Cichocki, "A further improvement of a fast damped Gauss-Newton algorithm for CANDECOMP-PARAFAC tensor decomposition," in *Proc. IEEE Int. Conf. on Acoust., Speech and Signal Process. (ICASSP)*, Vancouver, Canada, 2013, pp. 5964–5968.

[37] M. Turcotte, A. Kent and C. Hash, "Unified host and network data set," in *Data Science for Cyber-Security*, World Scientific, Nov. 2018, ch. 1, pp. 1-22.

APPENDIX

Theorem 1. *At the conclusion of Algorithm 1, $p_{2,n}$ contains the n-th degree polynomial coefficient of φ_2, for $n \in [N]_0$.*

Proof. All line numbers in this proof refer to those in Algorithm 1. By (5), we have that

$$\varphi_2(\tau) = \sum_{x_{i_1 \dots i_N} \neq 0} x_{i_1 \dots i_N} \sum_{r=1}^{R} \Big[\prod_{\ell=1}^{N} (a_{i_\ell r} + \tau d_{i_\ell r}) \Big].$$

Fix $\mathbf{i} = (i_1, \dots, i_N)$ such that $x_{\mathbf{i}} \neq 0$. In view of lines 1-2 and lines 12-15, it is sufficient to show that the quantity stored in $\pi_r^{(n)}$ in line 13 is given by the n-th degree polynomial coefficient of $\prod_{\ell=1}^{N}(a_{i_\ell r} + \tau d_{i_\ell r})$. Define

$$\psi_{m,r}(\tau) := \prod_{\ell=1}^{m} (a_{i_\ell r} + \tau d_{i_\ell r}) \text{ for } r \in [R],\, m \in [N],$$

and let $s_{m,r,q}$ denote the q-th degree coefficient of $\psi_{m,r}$ for $q \in [m]_0$. We claim that after reaching line 5 and upon completing $(m-1)$ iterations of the loop beginning in that line, $\pi_r^{(q)}$ is equal to $s_{m,r,q}$, $q \in [m]_0$, for $m \in [N]$, and prove this claim by induction on m.

If $m = 1$, then no iterations of the loop beginning in line 5 have been completed and the claim holds, as $\pi_r^{(0)} \leftarrow a_{i_1 r} =$

$s_{1,r,0}$ and $\pi_r^{(1)} \leftarrow d_{i_1 r} = s_{1,r,1}$ are the degree zero and one coefficients of $\psi_{1,r}(\tau) = a_{i_1 r} + \tau d_{i_1 r}$, respectively. Assume the claim holds for $(m-1)$ when $1 < m \leq N$. Then after completing $(m-2)$ iterations of the aforementioned loop, we have that $\pi_r^{(q)}$ contains $s_{m-1,r,q}$ for $q \in [m-1]_0$. Note that

$$\psi_{m,r}(\tau) = (a_{i_m r} + \tau\, d_{i_m r})\, \psi_{m-1,r}(\tau). \tag{7}$$

Suppose $q = m$. By line 6, the induction hypothesis, and (7),

$$\pi_r^{(m)} \leftarrow d_{i_m r}\, \pi_r^{(m-1)} = d_{i_m r}\, s_{m-1,r,m-1} = s_{m,r,m}.$$

Fix $q \in [m-1]$. Then similarly, by line 8, the induction hypothesis, and (7), the degree q coefficient of $\psi_{m,r}(\tau)$ is

$$\begin{aligned}\pi_r^{(q)} &\leftarrow d_{i_m r}\, \pi_r^{(q-1)} + a_{i_m r}\, \pi_r^{(q)} \\ &= d_{i_m r} s_{m-1,r,q-1} + a_{i_m r} s_{m-1,r,q} = s_{m,r,q}.\end{aligned}$$

Finally suppose $q = 0$. By line 10, the induction hypothesis, and (7), the degree zero coefficient of $\psi_{m,r}(\tau)$ is

$$\pi_r^{(0)} \leftarrow a_{i_m r} \pi_r^{(0)} = a_{i_m r} s_{m-1,r,0} = s_{m,r,0}.$$

By induction, the claim holds; taking $m = N$ yields the result. □

Theorem 2. *At the conclusion of Algorithm 2, $p_{3,n}$ contains the n-th degree polynomial coefficient of φ_3, for $n \in [2N]_0$.*

Proof. By (6) and lines 2–6 of Algorithm 2, we have that

$$\begin{aligned}\varphi_3(\tau) &= \mathbf{1}_R^T \left[\bigstar_{n=1}^N \Upsilon_{\mathbf{A}^{(\mathbf{n})} + \tau \mathbf{D}^{(\mathbf{n})}}\right] \mathbf{1}_R \\ &= \mathbf{1}_R^T \left[\bigstar_{n=1}^N \left(\mathbf{M_0^{(n)}} + \tau \mathbf{M_1^{(n)}} + \tau^2 \mathbf{M_2^{(n)}}\right)\right] \mathbf{1}_R.\end{aligned}$$

The conclusion follows from lines 7–11 of Algorithm 2. □