2018

ESTIMATING THE RESPIRATORY LUNG MOTION MODEL USING TENSOR DECOMPOSITION ON DISPLACEMENT VECTOR FIELD

Kingston Kang
VCU

Estimating the Respiratory Lung Motion Model Using Tensor Decomposition on Displacement Vector Field

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University

by

Kingston Kang
Department of Biostatistics
School of Medicine
Virginia Commonwealth University
March 7, 2018

Advisor: Nitai Mukhopadhyay
Associate Professor, Department of Biostatistics

Virginia Commonwealth University
Richmond, Virginia
December, 2017

Contents

Abstract

ESTIMATING THE RESPIRATORY LUNG MOTION MODEL USING TENSOR DECOMPOSITION ON DISPLACEMENT VECTOR FIELD

By Kingston Kang

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2018.

Major Director: Nitai Mukhopadhyay, Associate Professor, Department of Biostatistics

Modern big data often emerge as tensors. Standard statistical methods are inadequate to deal with datasets of large volume, high dimensionality, and complex structure. Therefore, it is important to develop algorithms such as low-rank tensor decomposition for data compression, dimensionality reduction, and approximation.

With the advancement in technology, high-dimensional images are becoming ubiquitous in the medical field. In lung radiation therapy, the respiratory motion of the lung introduces variabilities during treatment as the tumor inside the lung is moving, which brings challenges to the precise delivery of radiation to the tumor. Several approaches to quantifying this uncertainty propose using a model to formulate the motion through a mathematical function over time. [Li et al., 2011] uses principal component analysis (PCA) to propose one such model using each image as a long vector. However, the images come in a multidimensional arrays, and vectorization breaks the spatial structure. Driven by the needs to develop low-rank tensor decomposition and provided the 4DCT and Displacement Vector Field (DVF), we introduce two tensor decompositions, Population Value

Decomposition (PVD) and Population Tucker Decomposition (PTD), to estimate the respiratory lung motion with high levels of accuracy and data compression. The first algorithm is a generalization of PVD [Crainiceanu et al., 2011] to higher order tensor. The second algorithm generalizes the concept of PVD using Tucker decomposition. Both algorithms are tested on clinical and phantom DVFs. New metrics for measuring the model performance are developed in our research. Results of the two new algorithms are compared to the result of the PCA algorithm.

Chapter 1

Introduction

1.1 Background Knowledge

Functional analysis methods and algorithms can be used to interpolate and extrapolate data. However, with the advent of the big data era, modern large datasets often exhibit features including large data volume, high data dimensionality, and complex data structure which render most traditional statistical methods inadequate, since most standard methods and algorithms scale exponentially with data volume (size). The phrase 'curse of the dimensionality' [Bellman, 2015] is frequently used to describe that the number of elements in a high-dimensional and large-scale data increases exponentially as the dimensionality of the data increases, which eventually leads to the inability or inadequacy of standard machine learning algorithms dealing with datasets with extremely high volume, dimensionality, and complex structures [Cichocki et al., 2016]. Modern big data sets often emerge and are more efficiently presented and stored as multi-dimensional arrays, which are often referred to as tensors. On top of the high data volume, the complex and rich structural information in the data adds an additional layer of difficulty in analyzing tensors, which are multidimensional array of structured data. Therefore, it is very important to develop machine learning algorithms such as low-rank tensor decomposition for dimensionality reduction, data compression, and approximation. In this dissertation, we focus on introducing two new low-rank tensor decomposition and approximation algorithms with a specific application on displacement vector field to build a respiratory lung motion model. We will be explaining the terminology displacement vector field in Chapter 2 and the two new algorithms in Chapters 4 and 5.

Tensor decomposition is able to decompose high-dimensional datasets with complex multi-dimensional structures (we will refer to such datasets as tensors) into their intrinsic factor tensors or matrices and core tensors or matrices (definitions are given in Chapter 5). In fact, the well-known statistical method principal component analysis, which can be accomplished through either Eigen decomposition (spectral decomposition) [Chatterjee et al., 1997, Abdi, 2007a] or singular value decomposition [De Lathauwer et al., 1994], is the most basic and special example of tensor decomposition in two-dimensional case [Vasilescu and Terzopoulos, 2003]. In fact, researchers have generalized these commonly known decompositions in two-dimensional case to higher dimensional tensors [Cardoso, 1991, Yang et al., 2006, Cardoso, 1990, Abdi, 2007b, De Lathauwer et al., 2000a, Grasedyck, 2010, Savas and Eldén, 2007]. Through tensor decomposition, we are able to store high volume data as highly compressed low-rank tensors [Kolda and Sun, 2008]. The original high volume tensor can be reconstructed or approximated through operations on those decomposed factor and core tensors and matrices. Tensor decomposition achieves high level data compression and dimensionality reduction through dissecting data into relevant and irrelevant parts [Verstraete et al., 2008, Fukuhara et al., 2013, Orús, 2014, Szalay et al., 2015]. The level of data compression can be performed on datasets with more than 10^{50} entries to a manageable size of 10^{7} or less [Oseledets and Tyrtyshnikov, 2009, Dolgov et al., 2014, Kazeev et al., 2013, Kressner et al., 2014, Vervliet et al., 2014, Dolgov and Khoromskij, 2015, Liao et al., 2015].

Due to the advancement in technology and apparatus, high volume large data are becoming ubiquitous in the medical field [Doi, 2007, Lindner, 2004, Marro et al., 2016, Chung and Kim, 2015, Rosenkrantz et al., 2016, Chen et al., 2014, Picano et al., 2014]. 3D+Time images (such as 4D CT scans) are commonly used in assisting medical procedures. Such large high-dimensional medical images can have billions of elements with rich and complex structures. Thus, it is natural to store high dimensional medical images as tensors (multi-dimensional arrays) to retain both the high volume and the complex data structures. However, due to the extremely high volumes, high-dimensional images are difficult to analyze.

In this dissertation, the specific type of tensor data we are dealing with is five-dimensional (5D) displacement vector field derived from 4D CT lung scans (4DCT) obtained to help with planning radiation therapy. We will present how these 4DCT images are obtained and how the 5D displacement vector field is derived. We believe that understanding the process of data collection is crucial for performing any type of analysis. The displacement vector field is used to build a respiratory motion model which will help with planing radiation treatment. Previous research in this area includes average motion model and PCA-based motion model. This paper [McClelland et al., 2013] gives a good review of the current respiratory motion models available. We will be explaining the PCA-based motion model in Chapter 3 and use that as the current standard, since PCA-based model achieved better performance over average motion model [Li et al., 2011].

1.2 Radiation Therapy for Lung Cancer

Cancer is the leading cause of death worldwide [Ries et al., 2006, Jemal et al., 2008] and lung cancer is the most common type of cancer. The three main types of lung cancer include non-small cell lung cancer which accounts for about 85% of lung cancer, small cell lung cancer, and lung carcinoid tumor. Depending on the stage of cancer and other factors, external beam radiation therapy (EBRT) can be used as a treatment or in combination with other treatments. In EBRT, the precise delivery of radiation to the tumor while reducing the radiation exposure to the surrounding healthy tissue is one of the primary concerns. Due to the respiratory motion of the lung, the tumor inside the lung is constantly moving, which makes it harder to achieve an accurate delivery of radiation to the tumor. The error in external radiation treatment introduced by the respiratory motion of the lung is rather significant [Keall et al., 2006]. Currently, there are two common techniques of reducing the errors of radiation delivery introduced by respiratory motion of the lung. The first method is called deep inspiration breath hold (DIBH), where patients take a deep breath and hold his or her breath during the radiation treatment period. By holding the breath, we can minimize the lung motion due to respiration [Rosenzweig et al., 2000]. However, DIBH has its drawbacks including complex setup, increased time for treatment delivery, high level of patient cooperation, and technical support [Hanley et al., 1999]. The other method is free breathing technique, where patients are allowed to breath freely throughout the entire treatment delivery. Regardless of the methods, they all have their own advantages and drawbacks. However, the comparison of the two methods is not the focus of this dissertation. We focus on developing statistical methods and algorithms to increase the accuracy of radiation delivery using the free breathing technique. Currently, research solves this problem by developing a respiratory motion model for each patient. This model will be developed prior to radiation treatment and used throughout the radiation treatment. Our primary goal is to improve the current respiratory motion model by developing algorithms with better prediction accuracy and a higher level of data compression.

We believe that having some background knowledge about the radiation planning stage is helpful for readers to understand the problem and the data that we will be using in this dissertation. After deciding to receive radiation as an option for treating lung cancer and before actually receiving the treatment, every patient must undergo a simulation stage during which a computed tomography (CT) scanner will take multiple transverse slice images of a patient's lung. These images will be used to help doctors and radiation physicists with planing the radiation treatment and dose re-evaluation.

Chapter 2

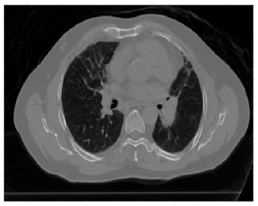
Displacement Vector Field and Respiratory Lung Motion

2.1 Data Collection Procedures and Data Structures

The data we will be using to estimate the respiratory lung motion model is called the displacement vector field (DVF), which is a five-dimensional array of structured data (fifth-order tensor). A good understanding of the data structure of the DVF data is crucial for performing any statistical analysis and modeling. The best way to reveal and understand DVF data structure is through looking at the data collection procedures.

Before receiving any radiation treatment, patients must undergo a simulation stage during which a CT scanner will take multiple images of a patient's lung. During a CT scanning, numerous x-ray beams and a set of electronic x-ray detectors rotate around the patient measuring the amount of radiation passing through the body, which in turn reveals the amount of radiation being absorbed by the scanned body parts. A computer will process the data and generate a 2D transverse slice plot of the scanned body parts in gray scales for each rotation. Different body parts absorb radiation by different degrees. The difference in the abilities to absorb radiation will be reflected by different intensities of gray in the 2D plot. The scanner will first take a 2D transverse slice at a starting location in one rotation, and it will move a couple millimeters and take another 2D transverse slice. It will repeat this process until the entire lung area of the patient is covered. Due to the advancement in technology, modern CT scanners are able to acquire multiple transverse slices in a single rotation, which also shortens the time of the entire CT scanning process.

The following is an example of a 2D transverse slice (Figure 2.1a) taken by a CT scanner plotted in gray scales. We can store this 2D image as a matrix (Figure 2.1b). The number of rows and columns of the matrix is directly determined by the total number of pixels in the original image. Typically, a transverse slice has a resolution of 512 by 512. After cropping the images close to the anatomy, the sizes of a cropped image will be reduced and will vary from patient to patient. The images we received are cropped and are about 300 by 450. When storing a 2D transverse slice as a matrix, there are about 300 rows (the size of r) and 450 columns (the size of c) in the matrix.


(a) A 2D Transverse Slice

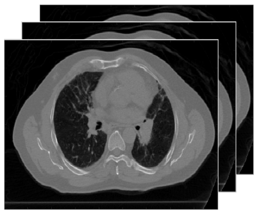
$$\begin{bmatrix} pixel_{11} & pixel_{12} & \dots & pixel_{1c} \\ pixel_{21} & pixel_{22} & \dots & pixel_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ pixel_{r1} & pixel_{r2} & \dots & pixel_{rc} \end{bmatrix}$$

(b) Saved as an r-by-c matrix

Figure 2.1: A 2D Transverse Slice Saved as an r-by-c Matrix

When we stack multiple 2D transverse slices together, we will have a 3D image representation (Figure 2.2a) of a patient's lung. The 3D object can be stored as a third order tensor (Figure 2.2b). The border of the tensor in Figure 2.2b is for the purpose of demonstration only. Here we will be explaining the concept of a tensor and its related elements in the context of DVF rather than designating a separate chapter for tensors alone.

A tensor is a multi-dimensional array of structured data. A basic concept in tensor is mode which is the same as the order of a tensor, and both mode and order quantify the dimensions of a tensor. For example, Figure 2.3 is a three-dimensional array, which is a third order tensor with three modes. Another basic concept in tensor is fiber, which is a vector in a given mode. The third order tensor shown below (Figure 2.3) has three modes. Each cube in the tensor corresponds to


(a) Multiple 2D transverse slices

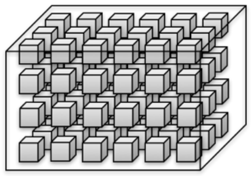

(b) Saved as a third order tensor

Figure 2.2: Multiple 2D Transverse Slices Saved as a Third-order Tensor

an element or a voxel in the 3D image obtained by stacking multiple 2D images. The green cubes together form a fiber in the first mode (called a mode-1 fiber); the blue cubes form a fiber in the second mode (mode-2 fiber); and the yellows cubes form a fiber in the third mode (mode-3 fiber). An element in the tensor can be represented using a lower case letter and its position can be referred using subscripts. For example, the red cube can be referred to as $x_{2,5,3}$. The concept of fiber is important because some techniques used in some algorithms introduced in Chapters 3 and 5 require different manipulations of fibers in tensor. We will explain the techniques in the chapters in which the algorithms are introduced and discussed.

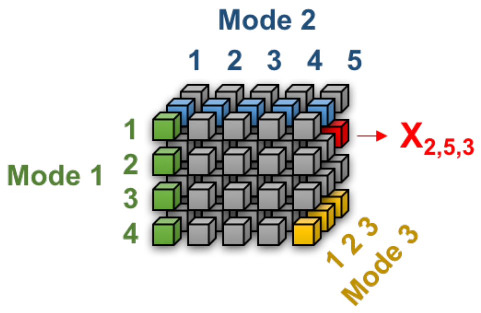


Figure 2.3: Fibers in a third order tensor

In order to capture the motion of the lung, we need to acquire multiple 3D images of the lung stored as tensors at different time points within a breathing cycle (Figure 2.4). An entire breathing cycle is evenly divided into ten segments with a total of ten time points, one time point at the beginning of each segment. These time points are referred to as phases. A 3D image of the lung is acquired at each phase as shown in Figure 2.4). Typically, we acquire multiple 3D images of the lung over multiple breathing cycles and then combine all the images to take a phase-wise average.

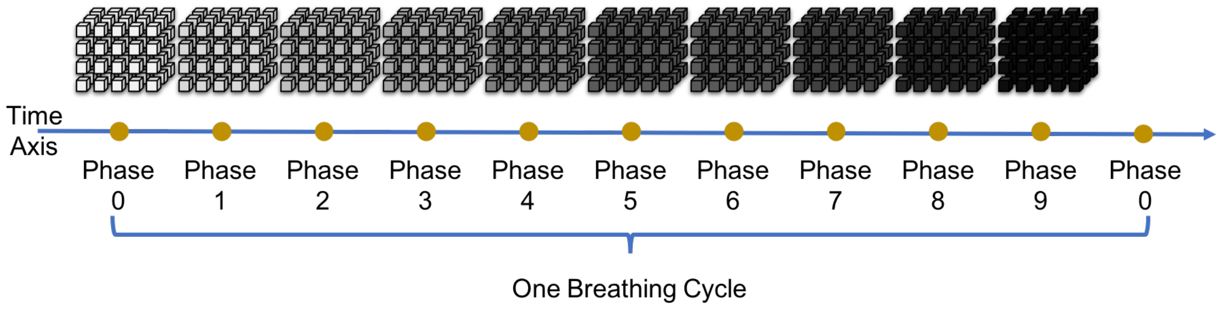


Figure 2.4: Tensors at Ten Phases Within One Breathing Cycle

Unlike 4DCT where each voxel has a value that represents intensity of its color, each cube in the DVF tensor data is not just one value but a vector of length three. This vector is referred to as the displacement vector, and this is where the DVF gets its name. We will first explain what displacement vector is and then how we obtain the displacement vector through applying image registration on 4DCT in the following paragraph. Displacement vector is a concept in physics and refers to a vector whose length is the shortest distance from the initial to the final position of a point in space. We will use an example (Figure 2.5) in 2D space to help familiarize readers with this concept. Imagine a point whose location is at the origin at time T_1. This point moves to a new location at time T_2. The vector $\vec{v} = (a, b)$, which is of length 2, pointing from the origin to the new location, is the displacement vector that both quantifies the magnitude and gives the direction of the motion of this point from the initial position to the final position within this 2D space. Simply stated, the displacement vector is the difference or the subtraction between the positions of this point at T_2 and T_1. In the case of DVF data, the displacement vector is in 3D space and is of length three. These three elements will be referred to as three different coordinates (cor_1, cor_2, cor_3).

The displacement vector in the DVF data is obtained by registering all images at all phases onto the same reference image through image registration. This registration is done patient-wise. Different patients are most likely to have different reference images. The displacement vector for each voxel (defined to be any of the discrete elements comprising a three-dimensional entity

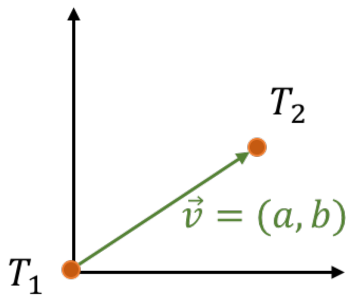


Figure 2.5: Displacement Vector in 2D Space

[Merriam-Webster.com. Merriam-Webster, 2018. Web. 16 Feb 2018.]) at any given phase provides the displacement of the corresponding anatomical locations from the reference image to the observed CT images at the given phase. For example, the difference between the image at phase 1 and the reference image is obtained by registering the image at phase 1 onto the reference image. The difference at each voxel or element (we will use the two words interchangeably) between the reference image and the image at phase 1 is a displacement vector of length three, which describes the element's motion and its new position at phase 1 from the reference image. Image registration is a topic that contains a lot of interesting study and research. The images we used have been pre-registered (thanks to Matthew Riblett). Because image registration is not the focus of our research, we will not be discussing the techniques and processes of image registration in greater detail. If readers are interested, we refer them to these articles [Lucas et al., 1981, Mattes et al., 2003, Collignon et al., 1995, Maes et al., 1997, Maintz and Viergever, 1998, Studholme et al., 1996, Pietrzyk et al., 1994, Zitova and Flusser, 2003, Dzingwa and Hayes, 2017, Oliveira and Tavares, 2014].

To summarize the DVF structure, it is a fifth order tensor with 5 modes (Figure 2.6). The first mode of DVF is the total number of rows in a 2D cropped transverse slice saved as a matrix, which is about 300 and can vary from patient to patient. The second mode is the total number of columns in a 2D cropped transverse slice saved as a matrix, which is about 400 and can also vary from patient to patient. The third mode is the total number of 2D transverse slices taken, which is about

100 and can also vary from patient to patient. The fourth mode is the total number of phases, which is a fixed number of 10. The fifth mode is the displacement vector in a 3D space derived from CT scans by registering images at each phase onto a reference image, which is of length 3. In general, one set of DVF for one patient contains about 360 millions ($300 \times 400 \times 100 \times 10 \times 3$) elements. To put that in the unit of computer memory, it is about 2 gigabytes worth of data per patient.

$$\begin{bmatrix} mode_1: & mode_2: & mode_3: & mode_4: & mode_5: \\ \#\,of\,rows\,in & \#\,of\,cols\,in & \#\,of & \# & displacement \\ a\,transverse & a\,transverse & transverse & of & vector \\ slice & slice & slices & phases & (cor_1, cor_2, cor_3) \\ 300ish & 400ish & 100ish & 10 & 3 \end{bmatrix}$$

Figure 2.6: Size of each mode is a displacement vector field (DVF)

The large size and high dimensionality of DVF data renders most traditional statistical and functional modeling inadequate. DVF is associated with a subject's lung anatomy, and therefore the distances between two elements in DVF are associated with actual physical meanings. If two elements in DVF tensor are next to each other, they represent tissues that are also close to each other in the subject's lung. Images are acquired over a period of time as represented by the measure of phase. Therefore, the structure of the DVF contains both spatial and temporal components and is multi-relational, which is another obstacle in analyzing the data.

2.2 Challenges Related to Analyzing Large-scale, High-dimensional, and Multi-relational Data

The challenges related to analyzing DVF are the massive size, high dimensionality, and multi-relational (both temporal and spatial relations) structure. In later chapters, we will be discussing the current algorithm, a principal component analysis (PCA) based algorithm, used to estimate the respiratory lung motion from DVF. We will explain how the algorithm works together with its advantages and drawbacks. Following this, we will introduce the two new algorithms and eventually compare them to the current PCA algorithm.

2.3 Clinical Displacement Vector Field Data

In this section, we will present the detailed structure and size of the ten sets of clinical DVF data that we will be using for analysis. These DVF are all obtained from Massey Cancer Center, School of Medicine, Virginia Commonwealth University.

The ten sets of DVF data belong to ten different patients. Each patient has their own set of DVF. To prepare DVF data for analysis, the data has to be registered. Image registration takes some time and has its own field of important and interesting study and research. Here we present the size of each mode in each set of DVF data in Table 2.1 and Table 2.2 .

Set	1^{st} mode	2^{nd} mode	3^{rd} mode	4^{th} mode	5^{th} mode
1	300	450	110	10	3

Table 2.1: Size of One Set of Clinical Full Lung DVF

Set	1^{st} mode	2^{nd} mode	3^{rd} mode	4^{th} mode	5^{th} mode
1	210	143	94	10	3
2	230	143	112	10	3
3	222	130	104	10	3
4	246	128	106	10	3
5	326	238	128	10	3
6	280	180	116	10	3
7	300	170	128	10	3
8	278	164	80	10	3
9	304	148	108	10	3

Table 2.2: Sizes of Nine Sets of Clinical Half Lung DVF

Table 2.1 shows the size of a set of DVF that contains a patient's both lungs, whereas the second table summarizes the sizes of 9 sets of one-lung (including a patient's either left or right lung in each set) DVFs, which is why the size of 2^{nd} mode in the second table is almost half of the size of the 2^{nd} mode in Table 2.1. We will refer to the DVF that contains both lungs as clinical full lung DVF and the one-lung DVF as clinical half lung DVF. We first obtained a set of full lung DVF. After applying our models on the full lung DVF, it showed promising results. We then requested more clinical DVF data. Because of the time consumption of registering images, we only use half lung DVF. Using half lung DVF not only saves us time on image registration but also shortens the time it takes to run our motion model algorithms. Most importantly, we have no reasons to doubt that the result of the algorithm performance will be affected by whether it is a set of DVF of both lungs or one lung. The size of the full lung DVF data is about 2.2 gigabytes, and the remaining nine sets are about half the size of the full lung DVF. All ten sets of DVF have 10 phases as described earlier in this chapter.

The size of each DVF varies realistically based on the size of each patient. This is not an accident but a natural occurrence as the medical facility, imaging instrument, and other setting are always changing from patient to patient. This is the reason why we cannot pool the information across patients even though the breathing motion of different patients are expected to share some similarities. All the clinical DVF are de-identifed.

2.4 Phantom Displacement Vector Field Data

With the help from Matthew Riblett who generated the simulated DVF data from a phantom lung, we are able to test our algorithms not only on clinical DVF but also on simulated DVF.

Because simulating DVF is quite time consuming, we acquired four sets of simulated DVF with varying levels of noise added. Each set of simulated DVF also has its noiseless version of DVF which can serve as ground truth. Before presenting the size of the simulated DVF, we will first include a brief description of the four sets of simulated DVF, which is provided by Matthew.

Each of the included models shares a common base model (M1), but have their own distinct modifications.

M1: Base Model (Spine, Ribs, Lungs, Carina)

The M1 model serves as the base model for approximating the size, location, and scope of thoracic anatomy and physiology as observed by a simple CT scan.

M2: Modified Base Model (Spine, Ribs, Lungs, Carina)

The M2 model modifies the base model introducing a smaller subject and, correspondingly, adjusting the size and location of subject anatomy (most notably the lungs and ribs). The respiration excursion for this model is also greater than that of the base model. The vessel trees within the lungs have also been ascribed a modest motion trajectory, unlike those of the base model.

M3: Modified Base Model (Spine, Ribs, Lungs, Carina)

The M3 model modifies the base model by adjusting the magnitude of patient respiration along the superior-inferior axis for each lung as well as slight variations in anatomical structure size (specifically, lungs and ribs). Instead of perfectly symmetrical motion, this model introduces a difference in excursion between the right and left lungs. (We eventually only used half lung in our analysis so the asymmetry in motion did not play any role.)

M4: Modified Base Model (Spine, Ribs, Lungs, Carina)

The M4 model modifies the base model by reducing the magnitude of patient respiration along the superior-inferior axis as well as slight variations in anatomical structure size (specifically, lungs and ribs). The magnitude of motion is significantly less than that of the base model and is intended to approximate shallow respiration.

Due to the large size of simulated DVF, we took the same practice and eventually only used half lung simulated DVF. Below we present the sizes of each simulated half lung DVF. The simulated sets of DVF have the same dimension, but the anatomy in the simulated sets of DVF are different.

Set	1st mode	2nd mode	3rd mode	4th mode	5th mode
1	512	256	100	10	3
2	512	256	100	10	3
3	512	256	100	10	3
4	512	256	100	10	3

Table 2.3: Sizes of Four Sets of Simulated Half Lung DVF

Chapter 3

Current Statistical Method of Estimating the Respiratory Lung Motion Model

3.1 Principal Component Analysis Based Respiratory Lung Motion Model

The current method that is being widely adopted and used to estimate the respiratory lung motion model is a principal component based (PCA) algorithm ([Li et al., 2011]). We will present and discuss this PCA method in terms of the algorithm and its advantages and drawbacks. Readers can refer to [Li et al., 2011] for more detail regarding this method.

3.2 Algorithm

The PCA-based method utilizes a common technique in dealing with high-order tensor data called vectorization. In the DVF data, when fixing the phase and coordinate in the displacement vector (of length 3), we are left with a third-order tensor. The PCA-based algorithm can be broken down into three steps. For each third order tensor for a given coordinate at a given phase within a breathing cycle, the PCA based algorithm first unfolds a third-order tensor data into a long vector by stacking the mode-1 fibers into a long vector (Step 1). The algorithm repeats this procedure for all 10 phases

(Step 2) and combines all long vectors into a matrix (Step 3). Each long vector obtained from step 1 corresponds to a column in the newly-formed matrix in step 3. The number of columns in the matrix corresponds to the number of phases in a breathing cycle. These three steps are repeated for each coordinate in the displacement vector in the DVF data and the newly-formed matrix for each coordinate are then stacked by row into a larger matrix, which is a second-order tensor that contains all the data from the fifth-order DVF tensor data. These procedures are summarized visually in the following figure (Figure 3.1) .

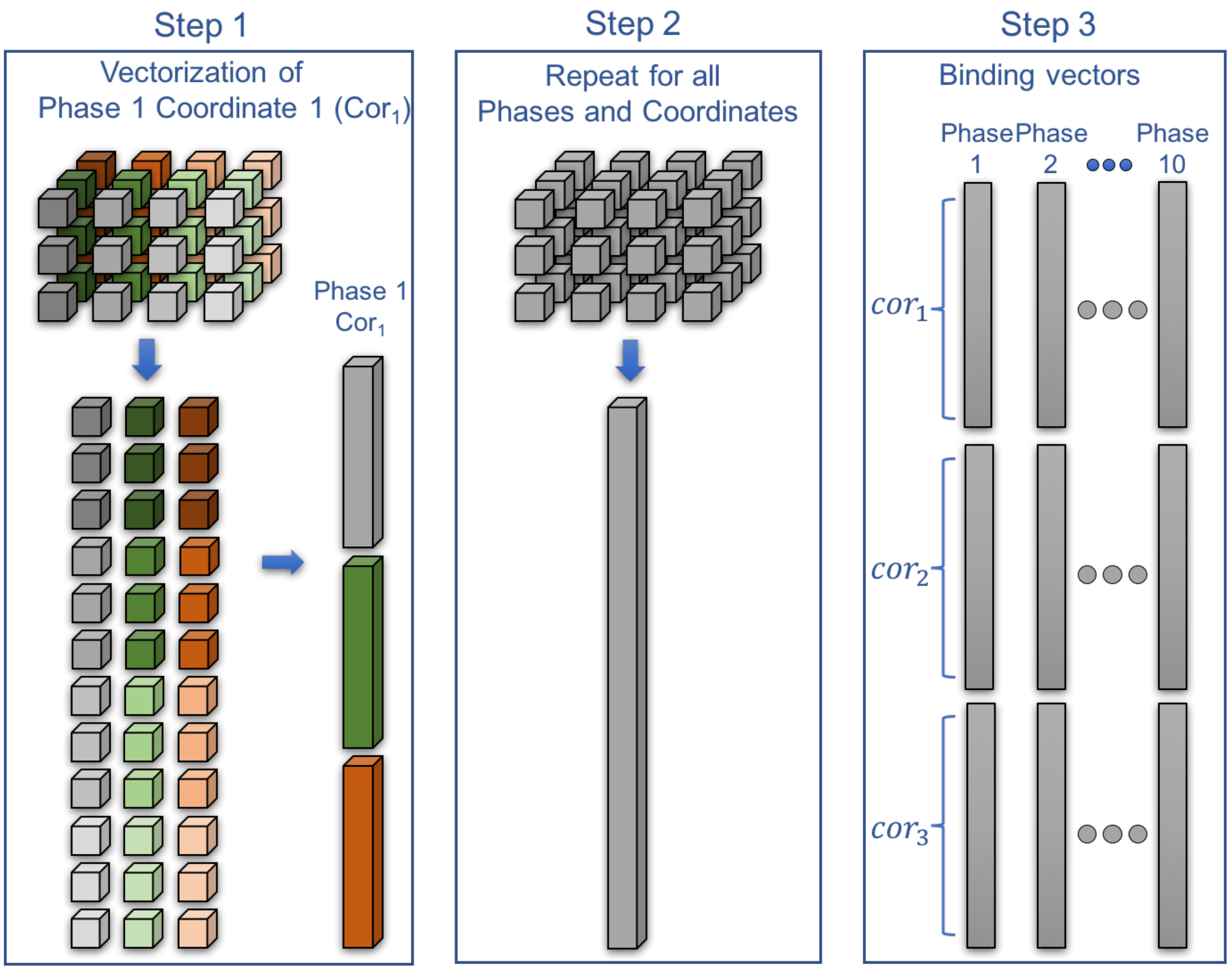


Figure 3.1: PCA based motion model

Once we form the new matrix in step 3, we will apply a standard PCA analysis on the matrix.

The PCA paper recommended keeping two eigenvectors and eigenvalues. The paper analyzes a total of eight patients and, for the majority of the DVF data analyzed, keeping two eigenvectors explains more than 80% of the total variance. We will be following this tradition when applying this PCA algorithm to our DVF data. However, we do want to point out at this point that the percentage of variance explained can vary from patient to patient and it is not uncommon to encounter a circumstance where keeping 2 eigenvectors and eigenvalues will not explain more than 80% of the total variance. In the paper, the performance of the PCA algorithm is accessed by MSE. We argue that MSE is not the best model performance indicator in this case. We will be explaining our choice and introducing new metrics for model performance assessment in chapter 6.

The performance of the PCA model will be assessed from multiple facets and be compared to our algorithms. The rationale of choosing metrics for performance measurement and comparisons will be discussed in the Chapter 6 as well.

3.3 Advantages, Drawbacks, and Limitations

The vectorization of a tensor in the first step of the PCA-based algorithm combined with column and row bindings of the vectorized tensor makes it possible for us to apply a standard PCA analysis on the DVF data. The advantages of this PCA-based algorithm include its simplicity, time efficiency, and relatively good accuracy with small MSE. However, the vectorization of the tensor data is also the drawback of the PCA-based algorithm, because the algorithm is unable to fully exploit the structures after vectorizing a high-order tensor. In Chapter 2, we discussed that the spatial structure of DVF has actual physical meanings. And this spatial structural information is damaged by the vectorization process in the PCA-based analysis. We propose new algorithms that both keep the data structure and meanwhile trying to maintain high efficiency in terms of processing and modeling time and accuracy.

Chapter 4

Estimating the Respiratory Lung Motion Model Using Population Value Decomposition on DVF

4.1 Introduction and Chapter Layout

The default population value decomposition (PVD) [Crainiceanu et al., 2011] is a newly-proposed algorithm to analyze a large population of two-dimensional images. The primary application of PVD is to reduce the size of a population of large two-dimensional images (can be seen as a second-order tensor) to a manageable size. The detailed descriptions of PVD can be found in this paper [Crainiceanu et al., 2011]. Because DVF is a fifth-order tensor, PVD cannot be directly applied due to the high-dimensionality of DVF. We will briefly present PVD to familiarize readers with algorithm and from there we will introduce the revised PVD and give a detailed description of the steps taken to apply the revised PVD to our fifth-order tensor DVF data.

4.2 Default Population Value Decomposition

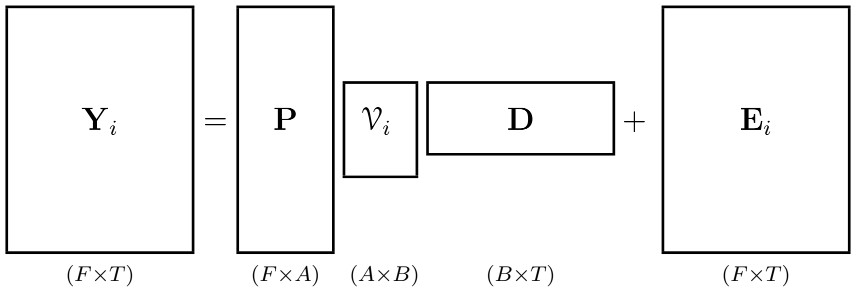
We first give the definition of default PVD and then we will give a pictorial representation to explain and help readers understand the algorithm.

Definition 4.2.1. If Y_i, $i = 1.2...., n$, is a sample of two-dimensional images of size F by T, then a population values decomposition (PVD) is

$$Y_i = P\,\mathcal{V}_i\,D + E_i,$$

where P and D are population-specific matrices of size $F \times A$ and $B \times T$, $\mathcal{V}_i$ is an $A \times B$-dimensional matrix of subject specific coefficients, and E_i is an $F \times T$-dimensional matrix of residuals.

The primary application of PVD is to reduce the dimension of a population of two-dimensional images to a manageable size. PVD will decompose a two-dimensional image stored as a second order tensor or matrix into a product of P, V, and D plus the residual matrix. The representation of PVD below illustrates the dimension (denoted in parentheses under each block) reduction.



Here Y_i represents a two-dimensional image stored as a matrix of size $F \times T$, and i is a an indicator that can stand for subject, visit, or other meaningful indicator depending on the source of image data. P and D are population specific matrices, which remain the same for each Y_i. They are of dimension $F \times A$ and $B \times T$. $\mathcal{V}_i$ is a subject specific matrix, which is unique to each Y_i. $\mathcal{V}_i$ is of dimension $A \times B$, where A and B are much smaller than F and T. The PVD algorithm effectively achieves data reduction by storing the two population specific matrices P, D and the much smaller subject specific matrix $\mathcal{V}_i$. Each original image Y_i is approximated by taking the product of P, $\mathcal{V}_i$, and D. The accuracy of approximation can be controlled. We will now describe how to compute the default PVD, which will eventually reveal how the accuracy of approximation is controlled.

4.3 The Default Population Value Decomposition Algorithm

The default PVD consists of two steps. The first step involves computing the singular value decomposition (SVD) for each two-dimensional image as below and keeping the first L_i leading columns (left singular vectors) from U_i to make a new matrix U_{L_i}. The choice of L_i can be based on various criteria (including variance explained, signal-to-noise ratio, or other practical standard).

$$\underset{F\times T}{\mathrm{Y}_i} = \underset{F\times F}{\mathrm{U}_i} \quad \underset{F\times T}{\Sigma_i} \quad \underset{T\times T}{\mathrm{V}_i^T} .$$

After repeating this step for all Y_i, we will combine all U_{L_i} matrices by columns to make an $F \times L$-dimensional matrix U as below. The number of columns (L) of the larger U matrix is equal to the sum of the number of columns for each U_{L_i} matrix. Repeat the above two steps for the V_i matrices to obtain a larger V matrix.

$$U = [U_{L_1}|, ..., |U_{L_n}] ;$$
$$V = [V_{L_1}|, ..., |V_{L_n}] .$$

The second step in PVD is to perform the spectral decomposition on $F \times F$-dimensional matrix UU^T, where U is obtained by combining U_{L_i} matrices across images. Because U contains information across all images, the spectral decomposition is performed on a population level. We will keep the first A leading eigenvectors of UU^T to form a new $F \times A$-dimensional matrix P. The product of $P(P^TU)$ gives us the best rank A approximation of P. We can also approximate the subject level U_{L_i} matrix by replacing the larger U matrix with the corresponding U_{L_i} matrix. We will apply the same procedures to the larger V matrix, where we will keep the first B leading eigenvector of V to form a new matrix D. We can then obtain the best rank B approximation of $D \approx D(D^TV)$ [Ye, 2005, Eckart and Young, 1936, Golub and Van Loan, 2012, Trefethen and Bau III, 1997].

The singular value decomposition on the subject level together with the spectral decomposition on the population level completes the PVD. We can approximate each two-dimensional image through the following steps:

$$\begin{aligned} Y_i &= U_i \Sigma_i V_i^T \\ &\approx U_{L_i} \Sigma_{L_i,R_i} V_{R_i}^T \\ &\approx P\{P^T U_{L_I} \Sigma_{L_i,R_i} (V_{R_i}^T D^T)\} D \\ &= P \mathcal{V}_i D. \end{aligned}$$

We compress data efficiently by only storing much smaller subject-specific matrices $\mathcal{V}_i$ and population-specific matrices P and D, where P is of dimension $F \times A$, D is of dimension $B \times T$, and $\mathcal{V}_i$ is of size $A \times B$ instead of the entire population of $F \times T$ images. Each 2D image can then be approximated by take the product of P, $\mathcal{V}_i$, and D.

4.4 The Revised Population Value Decomposition

Before generalizing and applying PVD to fifth-order tensor DVF, we noticed that the efficiency of the default PVD algorithm can be improved. Therefore we will revise part of the PVD algorithm here. In the second step of calculating the default PVD, before computing the spectral decomposition, we have to compute UU^T first, which is computationally expensive and actually unnecessary. Because the purpose of the second matrix decomposition is to find the leading eigenvectors for UU^T, which can be achieved through a simpler and more expedient mechanism. Therefore, we are revising the step in the default PVD algorithm. We will skip the calculation of UU^T and directly compute the singular value decomposition on U.

$$U = P_U \Sigma_U V_U^T.$$

We will use the first A leading eigenvector from P_U to form our P matrix. We can obtain the best rank A approximation of U the same way as we do in the spectral decomposition.

$$U \approx P(P^T U).$$

We can illustrate that the new revised PVD is equivalent to the default PVD in terms of the final result as below.
Compute the singular value decomposition of U.

$$U = P_U \Sigma V^T.$$

Then we have:

$$\begin{aligned} UU^T &= P_U \Sigma V^T (P_U \Sigma V^T)^T \\ &= P_U \Sigma V^T V \Sigma^T P_U^T. \end{aligned}$$

Because V and P_U are both unitary matrices, $V^T V$ is the identity matrix, and $P_U^T = P_U^{-1}$. Therefore, the above equation is eventually reduced to the form of spectral decomposition of UU^T :

$$\begin{aligned} UU^T &= P_U \Sigma V^T (P_U \Sigma V^T)^T \\ &= P_U \Sigma V^T V \Sigma^T P_U^T \\ &= P_U \Sigma^2 P_U^T \\ &= P_U \Sigma^2 P_U^{-1}. \end{aligned}$$

4.5 Generalization of the Revised Population Value Decomposition to the Fifth Order Displacement Vector Field Data

In the previous chapter, we discussed the drawbacks of PCA. When vectorizing DVF, which is fifth-order tensor, the spatial structure that has actual physical meaning is broken. We would like to keep the structure of DVF as intact as possible. Therefore, we believe that PVD, which is a matrix-based algorithm, can be helpful. However, as described earlier in this chapter, PVD can be applied to a population of two-dimensional images but cannot be directly applied to our fifth order DVF tensor data due to the high dimensionality. To generalize the PVD algorithm to the fifth order DVF tensor data, it is crucial to thoroughly understand the DVF data collection procedures, structures, and the meaning of each of its five dimensionalities (refer to Chapter 2).

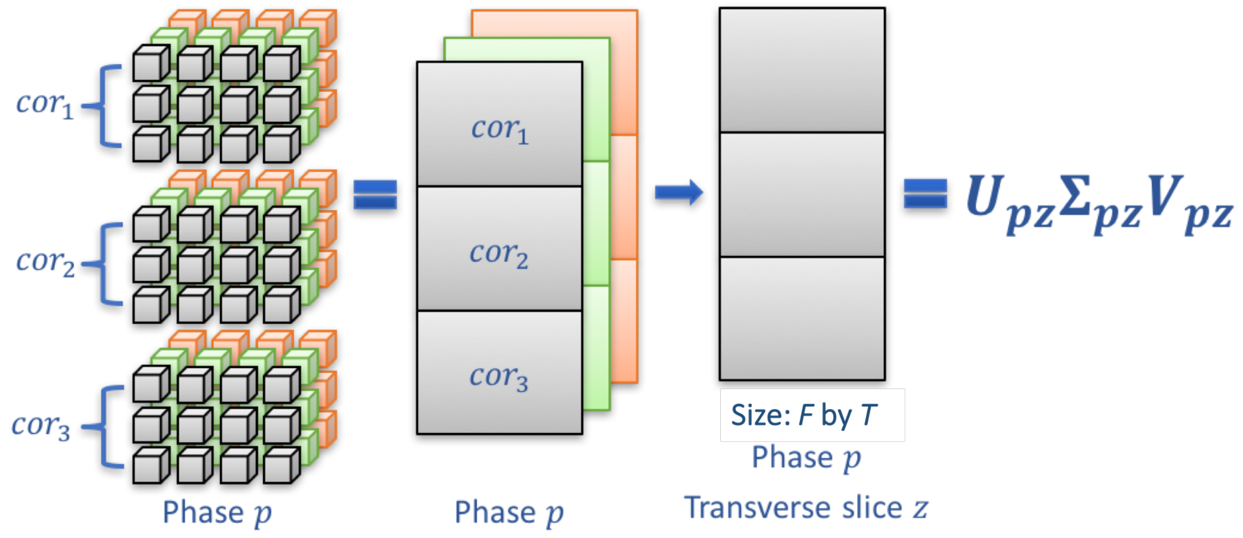


Figure 4.1: Unfold Coordinates and Perform SVD on each Transverse Slice

We will stack the coordinates and then compute the singular value decomposition for each stacked transverse slice composed by stacking all three coordinates at a given phase, which is of size F by T, illustrated in the above Figure 4.1. There are a total of 10 phases in our DVF data, and we will repeat the above procedure for all slices at all phases. Then we only take the leading eigenvectors in each singular value decomposition to get a best low-rank approximation of the corresponding original transverse slice as illustrated in the left side of Figure 4.2 below. The number of eigenvectors to be kept can be based on criteria including percentage of variance explained, signal to noise ratio, and other practical reasons. In our practice, we choose the percentage of variance explained to be the criterion. The third mode in the DVF data corresponds to the total number of transverse slices at a given phase, which can be quite large. Therefore, we are computing a significant number of the singular value decompositions at this stage of our algorithm. Current algorithm and computer architecture allows the computation of singular value decomposition to be completed fairly quickly for tall and skinny matrices [Benson et al., 2014, Faverge et al., 2017]. Because we are stacking the three displacement vector coordinates by rows (shown in Figure 4.1), the resulting matrix fits the tall and skinny category. The size of the final matrices are determined by the DVF tensor data, which will vary from patient to patient. We do observe empirically that the computation of the individual SVD for each slice for a given phase at this stage is not very time-consuming even though we used the default SVD algorithm in R base package and did not adopt the faster algorithms for tall and skinny matrices in the above referenced articles. We will discuss the time consumption and efficiency of the algorithm in chapter 7 in more detail.

After computing the SVD on the subject level, we will combine all the U_{ij}^* matrices into a larger U matrix and follow the second step in the revised PVD to calculate the singular value decomposition of U on the population level. Here i is an indicator for phase, which goes from 1 to 10. j is an indicator for transverse slices under a given phase, which goes from 1 to Z, where Z is the total number of transverse slices, which varies from patient to patient. Based on the percentage of variance explained criterion, we take a certain number of leading A eigenvectors from U_U to make a P matrix. PP^TU gives us the best rank A approximation of U. Repeat the second step in the revised PVD for the V matrix, which is obtained from combining all the V^* matrices. These steps are illustrated on the right side of Figure 4.2.

Figure 4.3 illustrates how we can approximate the original slice 1 at phase 1. Other slices can be approximated in a similar fashion. Data compression is achieved via storing the much smaller

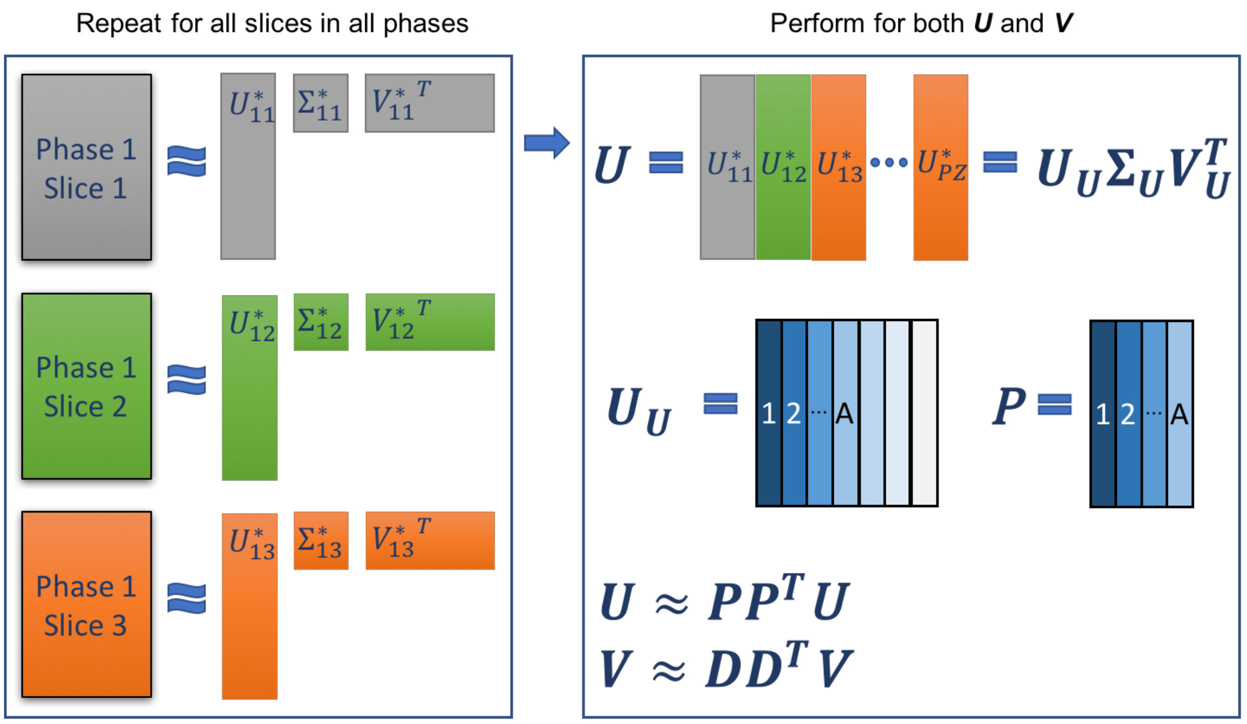

Repeat for all slices in all phases
Perform for both U and V
Phase 1 Slice 1
Phase 1 Slice 2
Phase 1 Slice 3
U_{11}^* Σ_{11}^* ${V_{11}^*}^T$
U_{12}^* Σ_{12}^* ${V_{12}^*}^T$
U_{13}^* Σ_{13}^* ${V_{13}^*}^T$
$U = U_{11}^* U_{12}^* U_{13}^* \cdots U_{PZ}^* = U_U \Sigma_U V_U^T$
$U_U =$ 1 2 ⋯ A
$P =$ 1 2 ⋯ A
$U \approx PP^T U$
$V \approx DD^T V$

Figure 4.2: Revised PVD on DVF

subject-specific V matrices (shown as V_{11} in Figure 4.3) and population-specific matrices P and D.

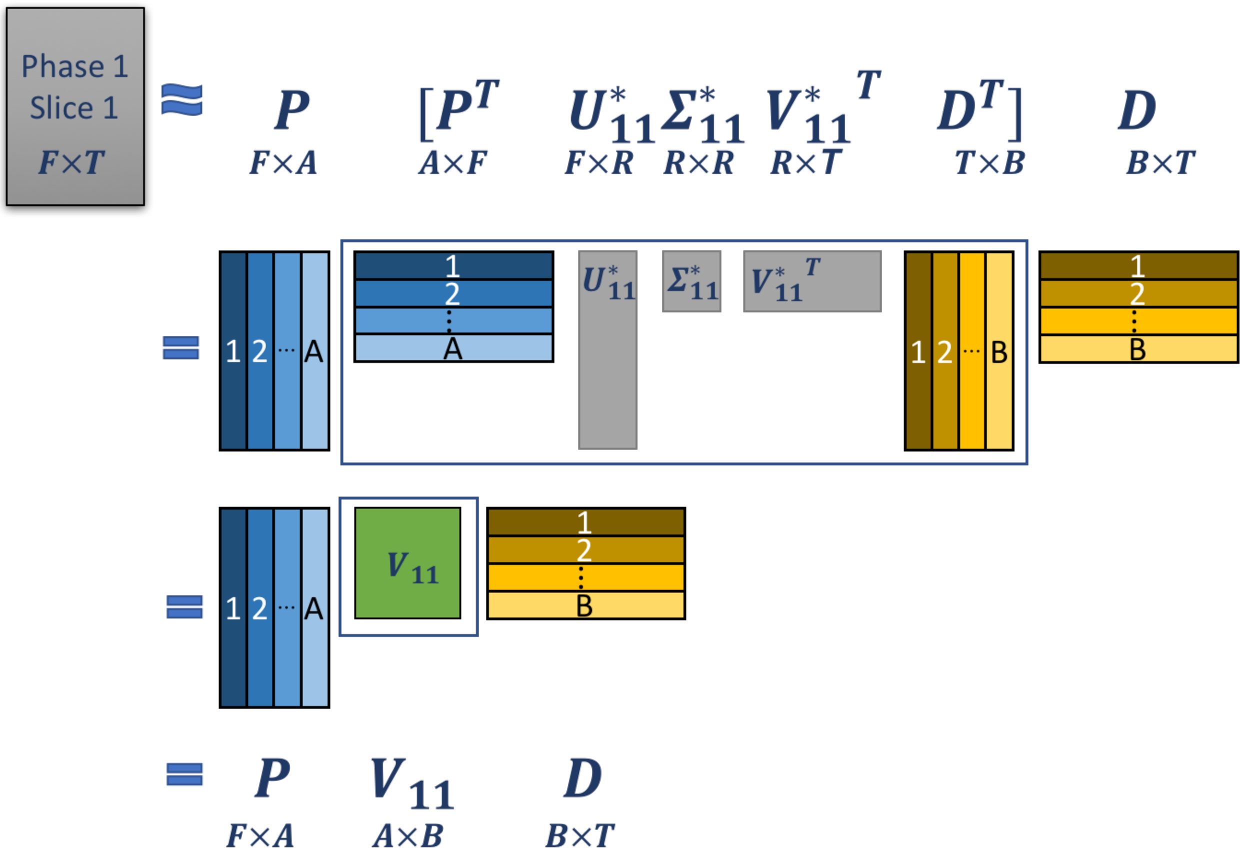

Phase 1
Slice 1
$F \times T$
$\approx P \, [P^T \, U^*_{11} \Sigma^*_{11} \, {V^*_{11}}^T \, D^T] \, D$
$F \times A$ $A \times F$ $F \times R$ $R \times R$ $R \times T$ $T \times B$ $B \times T$
1 2 ⋯ A
1 2 ⋮ A
U^*_{11}
Σ^*_{11}
${V^*_{11}}^T$
1 2 ⋯ B
1 2 ⋮ B
V_{11}
$= P \, V_{11} \, D$
$F \times A$ $A \times B$ $B \times T$

Figure 4.3: Reconstruction of a Transverse Slice

Chapter 5

Estimating the Respiratory Lung Motion Model Using Population Tucker Decomposition on DVF

5.1 Introduction and Chapter Layout

Because PVD does not require the vectorization of DVF data, it retains more of the structure of DVF as compared to the PCA based algorithm. But the iterative transverse slice-wise SVD is performed in order to generalize this matrix-based (SVD) algorithm to a fifth-order DVF tensor. The advantages of PVD over PCA are discussed in Chapter 7. We would like to further retain more of the structure of DVF and progress from a matrix-based algorithm to a truly high-order tensor-based algorithm. In this chapter, we will be presenting to the readers our second algorithm to estimate the respiratory lung motion model. We refer to this algorithm as the Population Tucker Decomposition (PTD). While this algorithm is original, the idea is inspired by both the PVD and Tucker Decomposition, which is a general form of Canonical Polyadic (CP) decomposition.

We will be explaining to our readers how the PTD algorithm works, which will show that this is a natural progression from PVD to PTD. We will again be using examples and figures to help readers understand the PTD algorithm. To understand PTD, one must be familiar with the concepts of PVD and Tucker Decomposition. We skip the explanation of PVD algorithm and advise readers to refer to previous chapters or [Crainiceanu et al., 2011] for details regarding PVD or the revised

PVD. However, we will be explaining the basic CP decomposition and Tucker Decomposition first and show how to compute the Tucker Decomposition. At the end, we will introduce the new algorithm PTD and show the natural progression from PVD to PTD by combining the ideas and concept of PVD with the third-order Tucker Decomposition.

5.2 New Concepts and Basic Definitions in Tensor Network and Decomposition

Tucker Decomposition (TD) [Oh et al., 2017, Hoff et al., 2016, Li et al., 2017] belongs to the family of low-rank tensor decompositions [Zhou et al., 2017, Cichocki et al., 2016, Yan et al., 2015, Rauhut et al., 2017]. It can be seen as a generalization of Canonical Polyadic Decomposition (CPD) [Lebedev et al., 2014, Zou et al., 2016, Wu et al., 2017, Cohen et al., 2015] , which itself can been seen a generalization of Singular Value Decomposition to a higher order tensor. One can also say that singular value decomposition is a special case of Canonical Polyadic Decomposition applied to only second-order tensor. We believe that SVD is a well-established and understood algorithm, and therefore will not be spending time explaining SVD. Readers can refer to [De Lathauwer et al., 1994] and [Golub and Reinsch, 1970] for more information. We will present to our readers CPD and progress to TD and their relative concepts.

5.2.1 Mode-n Product and Multilinear Product

In order to understand CPD, we have to introduce some new yet basic definitions in tensor network and decomposition to our readers.

The first new definition we need to introduce is called the Mode-n product between a tensor and a matrix.

Definition 5.2.1. Mode-n product of a tensor $\underline{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, where I_k represents the size of the

$k-$th mode ($k = 1, 2, ..., N$), and a matrix $B \in \mathbb{R}^{J \times I_n}$ yields a tensor

$$\underline{C} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times In+1 \times \cdots \times I_N}$$

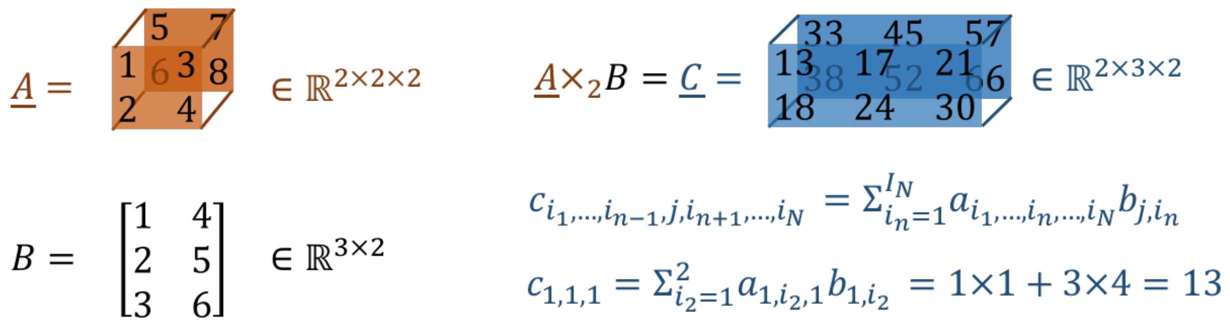
with entries

$$c_{i_1,\ldots,i_{n-1},j,i_{n+1},\ldots,i_N} = \sum_{i_n=1}^{I_n} a_{i_1,\ldots,i_n,\ldots,i_N} b_{j,i_n}.$$

This operation is written as:

$$\underline{C} = \underline{A} \times_n B.$$

We denote a tensor using a capital letter with an underline, a matrix with a capital letter, and an entry in a tenor using a lower case letter with subscripts representing the position at its corresponding mode. The best way to become familiar with new concepts is by example. We present to our reader an example of mode-2 product of a tensor with a matrix below. Here $\underline{A}$ is a 3rd order tensor whose modes are all of size 2 and B is a 3-by-2 matrix. Because the size of all three modes of tensor $\underline{A}$ is 2, we can technically preform mode-n product between tensor $\underline{A}$ and matrix B on any mode of tensor $\underline{A}$. Here we choose to perform the mode-2 product.



$\underline{A} =$
5 7
1 6 3 8
2 4
$\in \mathbb{R}^{2\times2\times2}$
$\underline{A} \times_2 B = \underline{C} =$
33 45 57
13 38 17 52 21 66
18 24 30
$\in \mathbb{R}^{2\times3\times2}$
$B = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \in \mathbb{R}^{3\times2}$
$c_{i_1,\ldots,i_{n-1},j,i_{n+1},\ldots,i_N} = \Sigma_{i_n=1}^{I_N} a_{i_1,\ldots,i_n,\ldots,i_N} b_{j,i_n}$
$c_{1,1,1} = \Sigma_{i_2=1}^{2} a_{1,i_2,1} b_{1,i_2} = 1\times1 + 3\times4 = 13$

Figure 5.1: Mode-2 Product of 3rd Order Tensor and 3-by-2 Matrix

Based on the definition of mode-n product, the final product tensor $\underline{C}$ will be a 3rd order tensor whose sizes of all three modes are 2, 3, and 2, respectively. Each entry in the product tensor $\underline{C}$ can be calculated using the definition one by one. The calculation of the entries in the final product tensor can be seen as the dot product between every mode-2 fiber from tensor $\underline{C}$ and each row from matrix B. We calculate the entry $c_{1,1,1}$ as an example.

To progress from mode-n product, we will now present the definition of multilinear product of a tensor with multiple matrices, where these matrices are often referred to as factor matrices or component matrices, and the tensor is referred to as the core tensor.

Definition 5.2.2. Multilinear product of a core tensor $\underline{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ and factor matrices $B^{(n)} \in \mathbb{R}^{I_n \times R_n}$ for $n = 1, 2, \ldots, N$ gives

$$\underline{C} = \underline{G} \times_1 B^{(1)} \times_2 B^{(2)} \cdots \times_N B^{(N)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}.$$

And this operation is written as:

$$\underline{C} = [\![\underline{G}, B^{(1)}, \ldots, B^{(N)}]\!].$$

The superscript (n) on the factor matrices $B^{(n)}$ indicates for which mode a particular factor matrix $B^{(n)}$ is. Therefore, the superscript (n) matches the n in the mode-n product. The size of the mode of the final product tensor $\underline{C}$ is determined by the size of the first mode in the factor matrices.

As long as readers are familiar with mode-n product, multilinear product of a core tensor is basically preforming mode-n product multiple times with different factor matrices. We will not be giving an example for this definition.

5.2.2 Kronecker Product and Mode-k Matricization

Next we will give the definition for Kronecker product for tensor. There are two types of Kronecker products: the left Kronecker product and the right Kronecker product. Here we present the definition for the left Kronecker product:

Definition 5.2.3. The left Kronecker product of tensors $\underline{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and $\underline{B} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$ yields a tensor

$$\underline{C} \in \mathbb{R}^{I_1 J_1 \times I_2 J_2 \times \cdots \times I_N J_N}$$

with entires

$$c_{\overline{i_1 j_1 i_2 j_2, \ldots, i_N j_N}} = a_{i_1, \ldots, i_N} b_{j_1 j_2, \ldots, j_N}.$$

This operation can be written as:

$$\underline{C} = \underline{A} \otimes_L \underline{B}.$$

In the above definition of left Kronecker Product, we introduce a new notation $c_{\overline{i_1 j_1 i_2 j_2, \ldots, i_N j_N}}$ that has not been mentioned before. This notation is called the little-endian notation, which derives from Jonathan Swift's Gulliver's Travels in which the little-endians are a political faction who breaks their eggs at the small end.

Definition 5.2.4. $i = \overline{i_1 i_2, \ldots, i_N}$ is a multi-index which takes all possible combinations of values of indices, $i_1, i_2, \ldots, i_N$, for $i_n = 1, 2, \ldots, I_n$, $n = 1, 2, \ldots, N$ in the following specific order convention:

$$\begin{aligned} &\overline{i_1 i_2, \ldots, i_N} \\ =& i_1 + (i_2 - 1) I_1 + (i_3 - 1) I_1 I_2 + \cdots + (i_N - 1) I_1 \cdots I_{N-1}. \end{aligned}$$

The little-endian convention is the reverse lexicographic ordering. As one might expect, there is also a big-endian convention. Different programing softwares use different ordering conventions.

The kronecker product for matrices is also necessary for understanding the computation of TD and is a much easier definition to grasp compared to its definition in tensor. We will not present the definition here.

In Chapter 3, we discussed the vectorization of a tensor. Here will will present another technique in dealing with tensor data: matricization.

Definition 5.2.5. $X_{(k)}$ is the mode-k matricization, which arranges the mode-k fibers of X as columns into a matrix.

The best way to understand this definition is to give a figure representation. In Figure 5.2, we have a 3rd-order tensor and we present the three matricizations of this tensor.

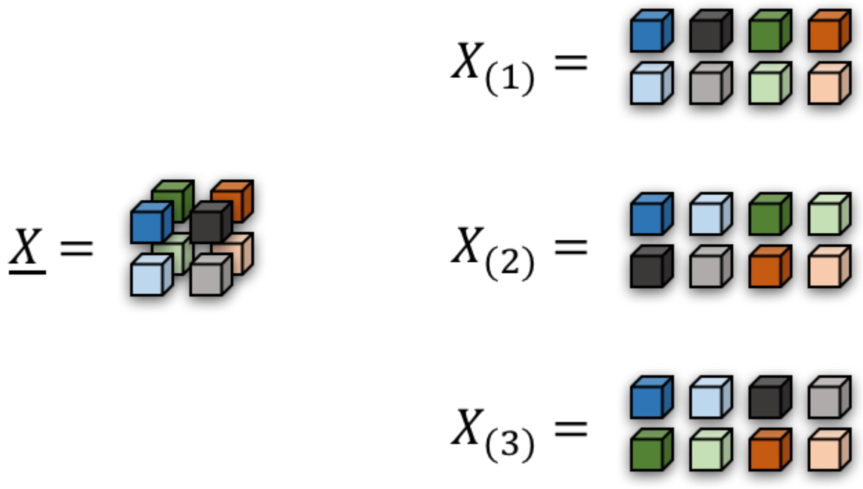


Figure 5.2: Three Matricizations of a 3rd-order Tensor

5.3 Tucker Decomposition

We mentioned at the beginning of this chapter that CP decomposition is a generalization of SVD to higher order tensor. While CP decomposition can be expressed using multiple different notations, we will be summarizing CP decomposition using the multilinear product definition we introduced in the previous section. The CP decomposition (sometimes also referred to as the PARAFAC or CANDECOMP) can decompose any n-th order tensor $\underline{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ into a space diagonal (defined later) core tensor $\underline{\Lambda} \in \mathbb{R}^{R \times R \times \cdots \times R}$ and n factor matrices $B^{(N)}$ [Hitchcock, 1928, Harshman, 1970, Carroll and Chang, 1970], and can be written as:

$$\underline{X} = \underline{\Lambda} \times_1 B^{(1)} \times_2 B^{(2)} \cdots \times_N B^{(N)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N},$$

or

$$\underline{X} = [\![\underline{\Lambda}, B^{(1)}, B^{(2)}, \ldots, B^{(N)}]\!].$$

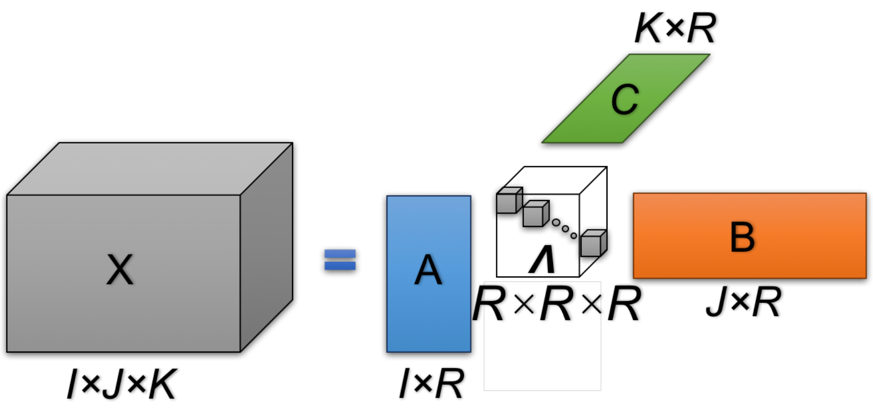


Figure 5.3: CP Decomposition of a 3rd-order Tensor

Figure 5.3 presents what the CP decomposition does to a 3rd order tensor. Here we have a 3rd-order tensor $\underline{X} \in \mathbb{R}^{I\times J\times\times K}$, which can be decomposed into a multilinear product between a space diagonal core tensor (defined below) $\underline{\Lambda} \in \mathbb{R}^{R\times R\times\times R}$ and three factor matrices $A \in \mathbb{R}^{I\times R}$, $B \in \mathbb{R}^{J\times R}$, and $C \in \mathbb{R}^{K\times R}$. The core tensor $\underline{\Lambda}$ only has non-zero entries on the space diagonal of the core tensor $\underline{\Lambda}$.

Definition 5.3.1. A tensor $\underline{\Lambda} \in \mathbb{R}^{I\times I\times I}$ is a space diagonal tensor if its element $r_{ijk} = 0$ for all $i, j, k = 1, 2, ..., I$ except where $i = j = k$.

Compared to the CP decomposition, Tucker decomposition is less memory-consuming [Khoromskij and Khoromskaia, 2007] and provides a more flexible and general decomposition of any nth order tensor in the sense that it will result in a much smaller core tensor, even though not space diagonal, and smaller factor matrices. The Tucker decomposition will decompose any nth order tensor $\underline{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ into a product of core tensor $\underline{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ and n factor matrices $B^{(N)}$. We can summarize Tucker Decomposition using the multilinear product definition:

$$\underline{X} = \underline{G} \times_1 B^{(1)} \times_2 B^{(2)} \cdots \times_N B^{(N)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N},$$

or

$$\underline{X} = [\![\underline{G}, B^{(1)}, B^{(2)}, \ldots, B^{(N)}]\!].$$

The multilinear product notations for both CP decomposition and Tucker decomposition are very similar. The difference is that CP decomposition will result in a space diagonal core tensor where the only non-zero entries are on the space diagonal, whereas the Tucker decomposition will result in a general core tensor where usually all entries in this core tensor are non-zero. However, the core tensor from the Tucker decomposition are usually much smaller ($R_N << I_N$), as are the factor matrices. Because of this reason and the fact that it gives us more flexibility in choosing the size of the core tensor, we prefer using Tucker decomposition over CP decomposition. In fact, CP decomposition can be seen as a special case of Tucker decomposition [Bergqvist and Larsson, 2010]. Regarding the theory, applications, and connection between CP decomposition and Tucker decomposition, readers can refer to [Bergqvist and Larsson, 2010]. Both CP and Tucker decomposition have a long history. For more details regarding both algorithms, readers can refer to [Kolda and Bader, 2009, Sears et al., 2009, Grasedyck et al., 2013, Comon, 2014, Cichocki et al., 2015, Huang et al., 2016]. In Figure 5.4, we present the figure representation of the Tucker decomposition.

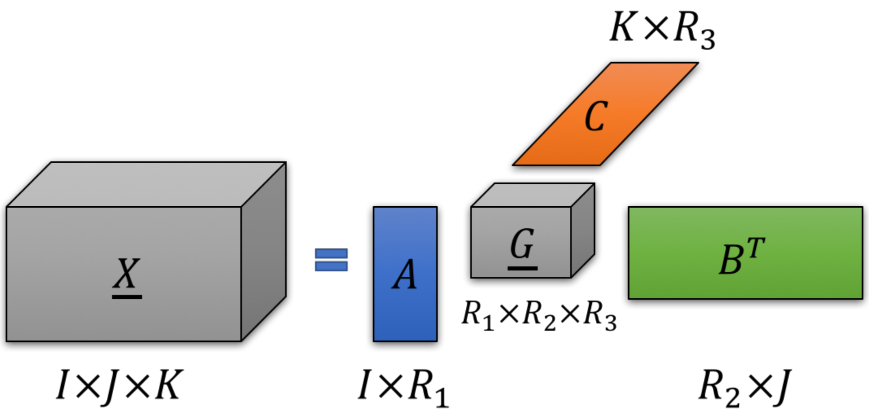


Figure 5.4: Tucker Decomposition of a 3rd-order Tensor

5.4 Computing the Tucker Decomposition

To compute the Tucker decomposition, we will use the higher order orthogonal iteration (HOOI) algorithm [Liu et al., 2014, Sheehan and Saad, 2007, De Lathauwer et al., 2000b] which utilizes higher order singular value decomposition (HOSVD) [Haardt et al., 2008, Huang et al., 2008, Hoge and Westin, 2005, Afra et al., 2014]. We will explain to readers how to compute the Tucker decomposition using an example of a 3rd-order tensor.

The computation of Tucker decomposition utilizes the following equivalence:

$$\begin{aligned}
\underline{X} &= [\![\underline{G}, B^{(1)}, B^{(2)}, B^{(3)}]\!] \\
\Leftrightarrow X_{(1)} &= B^{(1)} G_{(1)} (B^{(3)} \otimes B^{(2)}) \\
X_{(2)} &= B^{(2)} G_{(2)} (B^{(3)} \otimes B^{(1)}) \\
X_{(3)} &= B^{(3)} G_{(3)} (B^{(2)} \otimes B^{(1)}).
\end{aligned}$$

Here $X_{(1)}$, $X_{(2)}$, and $X_{(3)}$ are the three mode-k matricizations of the third-order tensor $\underline{X}$. $B^{(k)}$ is the factor matrix, obtained by taking the left orthogonal matrix of SVD of $X_{(k)}$. After solving the

$B^{(k)}$, we need to obtain the core tensor through the following steps:

$$\begin{aligned}
\underline{X} &= [\![\underline{G}, B^{(1)}, B^{(2)}, B^{(3)}]\!] \\
&= \underline{G} \times_1 B^{(1)} \times_2 B^{(2)} \times_3 B^{(3)} \\
\underline{X} \times_1 B^{(1)^T} &= \underline{G} \times_1 B^{(1)} \times_2 B^{(2)} \times_3 B^{(3)} \times_1 B^{(1)^T} \\
\underline{X} \times_1 B^{(1)^T} &= \underline{G} \times_1 B^{(1)} \times_1 B^{(1)^T} \times_2 B^{(2)} \times_3 B^{(3)} \\
\underline{X} \times_1 B^{(1)^T} &= \underline{G} \times_1 (B^{(1)^T} B^{(1)}) \times_2 B^{(2)} \times_3 B^{(3)} \\
\underline{X} \times_1 B^{(1)^T} \times_2 B^{(2)^T} \times_3 B^{(3)^T} &= \underline{G}. \\
[\![\underline{X}, B^{(1)}, B^{(2)}, B^{(3)}]\!] &= \underline{G}.
\end{aligned}$$

From the third line to fourth we can move the term $B^{(1)^T}$ on the right side of the equation ahead based on the following tensor matrix product operation law.

Lemma 5.4.1. *Given a tensor* $\underline{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, *a matrix* $A \in \mathbb{R}^{J \times I_n}$, *and another matrix* $B \in \mathbb{R}^{K \times I_m}$,

$$\underline{X} \times_n A \times_m B = \underline{X} \times_m B \times_n A.$$

if $I_n \neq I_m$.

From the fourth line to the fifth line, we will use the following tensor matrix product operation law.

Lemma 5.4.2. *Given a tensor* $\underline{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, *a matrix* $A \in \mathbb{R}^{J \times I_n}$, *and another matrix* $B \in \mathbb{R}^{K \times J}$,

$$\underline{X} \times_n A \times_n B = \underline{X} \times_n (BA).$$

The truncated HOSVD does not result in the best low multilinear rank approximation, even though the low multilinear rank approximation is always a well-posed problem. The truncated

HOSVD does satisfy the quasi-best approximation property (De Lathauwer et al., 2000a). The quasi-best approximation property states that

$$\|\underline{X} - [\![\underline{G}, B^{(1)}, B^{(2)}, \cdots, B^{(N)}]\!]\| \leq \sqrt{N}\|\underline{X} - \underline{X}_{Best}\|,$$

where $\underline{X}_{Best}$ is the best low multilinear rank approximation of $\underline{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ for a given tensor norm $[\![\cdot]\!]$.

We achieve the optimal approximation by minimizing a Frobenius Norm cost function $J = \|\underline{X} - [\![\underline{G}, B^{(1)}, B^{(2)}, B^{(3)}]\!]\|_F^2$. We minimize this cost function J using the higher order orthogonal iteration (HOOI), which is an alternating least square (ALS) based algorithm. Again, using a third-order tensor $\underline{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ as a example, here is how to find the best low multilinear rank approximation using the HOOI:

1. Initialize factor matrices $B^{(1)}$, $B^{(2)}$, and $B^{(3)}$ using HOSVD
   a) for $i = 1, 2, 3$ $G \leftarrow \underline{X} \times_{p \neq i} \{B^{(p)^T}\}$.
   b) Update $B^{(i)} \leftarrow R_i$ leading left singular vectors of $G_{(i)}$.
2. Repeat step 2 and 3 until $J = \|\underline{X} - [\![\underline{G}, B^{(1)}, B^{(2)}, B^{(3)}]\!]\|_F^2$ converges.

This ALS type algorithm can be easily extended to any Nth-order tensor $\underline{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$,

1. Initialize factor matrices $B^{(1)}, B^{(2)}, \cdots, B^{(N)}$ using HOSVD
2. Repeat
   a) For $i = 1$ to N do
   b) $G \leftarrow \underline{X} \times_{p \neq i} \{B^{(p)^T}\}$.
   c) Update $B^{(i)} \leftarrow R_i$ leading left singular vectors of $G_{(i)}$.
   d) End for

3. Stop repeat if the cost function $J = \|\underline{X} - [\![\underline{G}, B^{(1)}, B^{(2)}, \cdots, B^{(N)}]\!]\|_F^2$ converges.

Another algorithm for HOOI using randomization for large scale data [Zhou et al.m 2015] is also available but we will not list the detailed steps of the algorithm here. While other sophisticated algorithms for Tucker decomposition with different constraints are also available [Phan and Cichocki, 2011; Zhou et al., 2012; Constantin et al., 2014; Jeon et al., 2016], we will not discuss their advantages or drawbacks as they are not the focus of this dissertation. If readers are interested in learning more about this topic, please refer to the articles referenced above and this book [Cichocki et al., 2016] for more information.

5.5 Population Tucker Decomposition

After introducing the new concepts and basics of computing Tucker decomposition, in this chapter we dive into our new algorithm, which we call the Population Tucker Decomposition (PTD). The name Population Tucker Decomposition arises from the fact that this algorithm is essentially a combination of the idea of PVD and Tucker Decomposition, which is in turn a natural progression from PVD. We will illustrate the ideas behind PTD by detailing the steps.

The steps of PTD can be summarized in Figure 5.5. In the left part of the figure, each grey cuboid represents a third-order tensor obtained by fixing the phase index and stacking the three coordinates vertically in DVF data. Therefore, we will end up with a total ten cuboids each representing a unique phase in a breathing cycle. For each cuboid, we perform a Tucker decomposition using the HOOI to achieve the best low multilinear rank approximation of each cuboid. Comparing this to the PVD algorithm (as shown below in 5.6), we jump from a matrix-based algorithm to a truly high order tensor-based algorithm. The similarity between PTD and PVD at this stage is that the first step in both algorithm is performed on a subject-level. However, one should not confuse the definition of subject-level in PTD and PVD. In the case of PVD, the subject-level decomposition

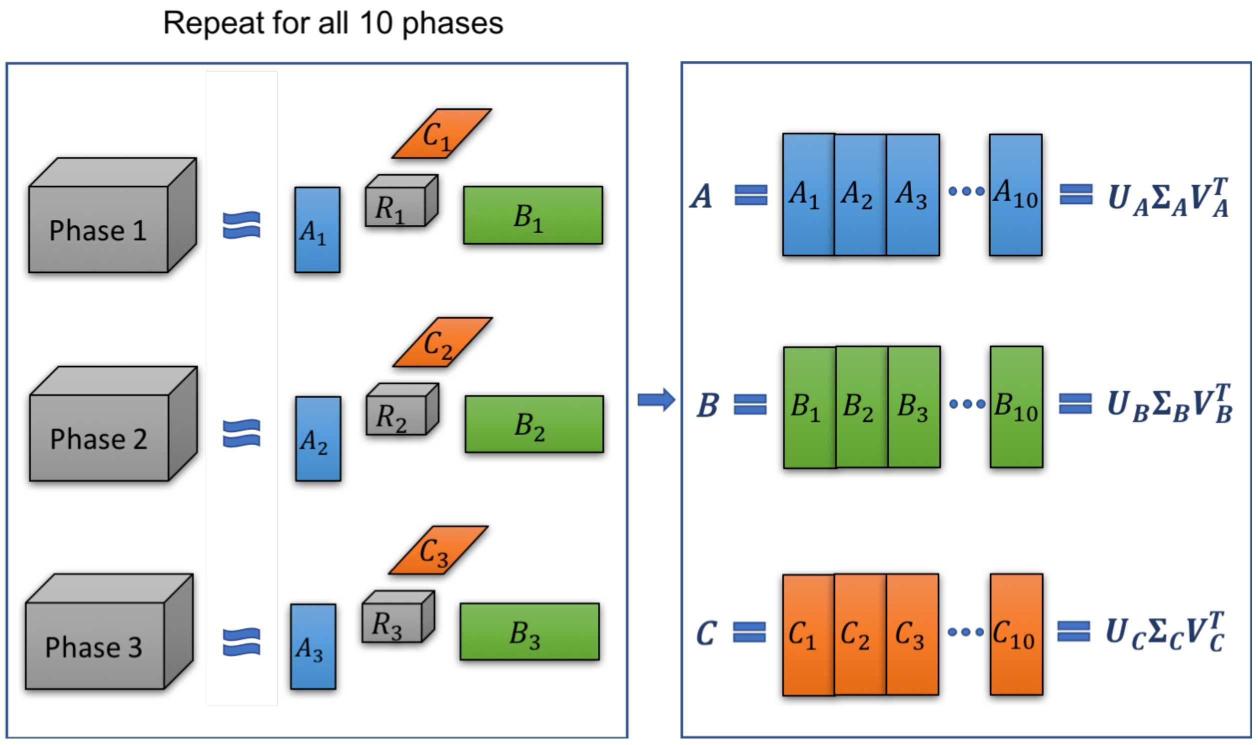

Repeat for all 10 phases
Phase 1
Phase 2
Phase 3
A_1 R_1 C_1 B_1
A_2 R_2 C_2 B_2
A_3 R_3 C_3 B_3
$A = A_1\ A_2\ A_3 \cdots A_{10} = U_A \Sigma_A V_A^T$
$B = B_1\ B_2\ B_3 \cdots B_{10} = U_B \Sigma_B V_B^T$
$C = C_1\ C_2\ C_3 \cdots C_{10} = U_C \Sigma_C V_C^T$

Figure 5.5: Population Tucker Decomposition

happens on each individual two-dimensional image, whereas in the case of PTD, the subject level decomposition refers to the best low multilinear rank approximation for each individual phase.

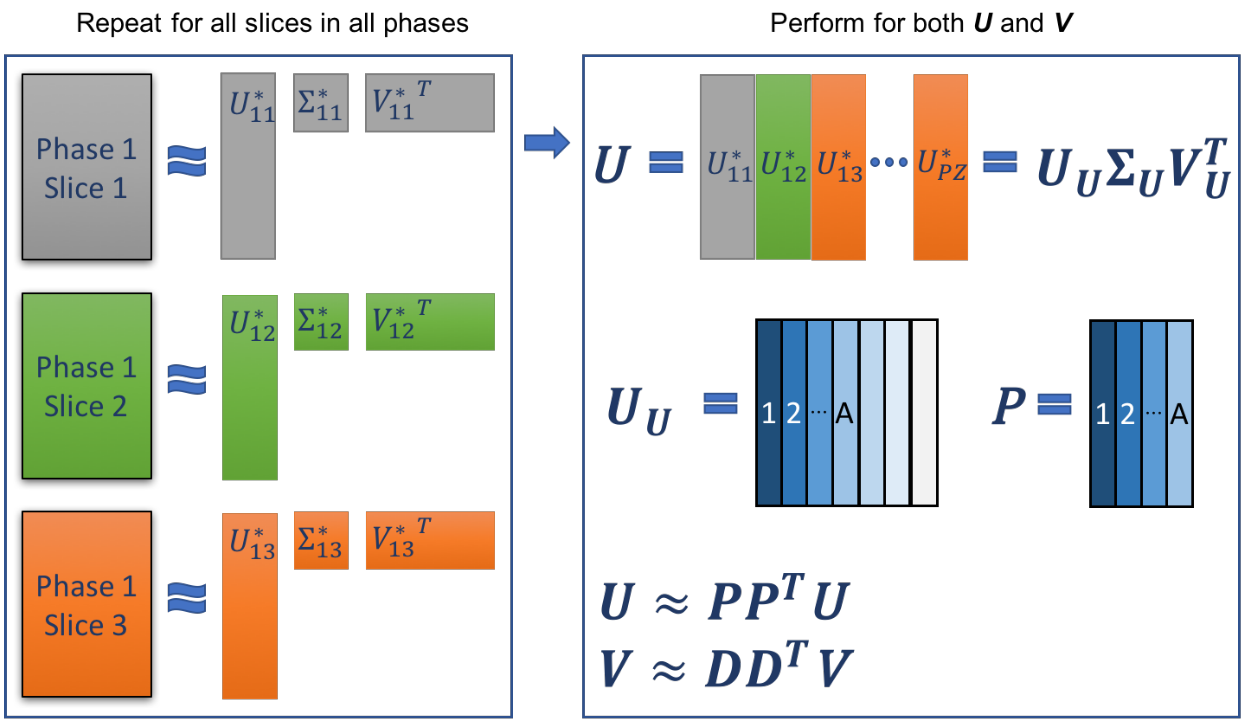

Repeat for all slices in all phases
Perform for both U and V
Phase 1 Slice 1
Phase 1 Slice 2
Phase 1 Slice 3
U^*_{11} Σ^*_{11} ${V^*_{11}}^T$
U^*_{12} Σ^*_{12} ${V^*_{12}}^T$
U^*_{13} Σ^*_{13} ${V^*_{13}}^T$
$U = U^*_{11} U^*_{12} U^*_{13} \cdots U^*_{PZ} = U_U \Sigma_U V_U^T$
$U_U =$ 1 2 ⋯ A
$P =$ 1 2 ⋯ A
$U \approx PP^T U$
$V \approx DD^T V$

Figure 5.6: Revised PVD on DVF

Once we obtained the Tucker decomposition, we will combine all the corresponding subject factor matrices A_i, B_i, C_i to make larger factor matrices A, B, and C. We may refer to these matrices A, B, and C as population factor matrices since they contain factor matrix from each phase. The second step is to perform SVD on the population factor matrices. The similarity between PVD and PTD at this stage is that this step is performed on a population-level. We keep the leading singular vectors from U_A, U_B, U_C in the SVD, which will be referred to as U_{A_L}, U_{B_L}, and U_{C_L}. The best low-rank approximation of the population and subject factor matrices can be obtained through the following steps:

$$A \approx U_{A_L} U_{A_L}^T A$$

$$A_i \approx U_{A_L} U_{A_L}^T A_i$$

$$B \approx U_{B_L} U_{B_L}^T B$$

$$B_i \approx U_{B_L} U_{B_L}^T B_i$$

$$C \approx U_{C_L} U_{C_L}^T C$$

$$C_i \approx U_{C_L} U_{C_L}^T C_i.$$

Subsequently, each cuboid can be approximated through the following steps:

$$\underline{X}_{phase_i} \approx [\![\underline{R}_i, F_{1i}, F_{2i}, F_{3i}]\!];$$

$$F_{1i} = U_{A_L} U_{A_L}^T A_i;$$

$$F_{2i} = U_{B_L} U_{B_L}^T B_i;$$

$$F_{3i} = U_{C_L} U_{C_L}^T C_i;$$

where $i = 1, \cdots, 10$ and represents 10 phases.

As can be observed from the above equations, the factor matrices are not population-specific because they include indicator i. The core tensor is also subject-specific since it depends on i. This is not the result we desire. We would like to have a result similar to PVD where we only want to store population specific factor matrices, which do not change across different phases, and subject-specific core tensors that are unique to each phase. That way we achieve the best data compression and utilize less memory space by storing only three factor matrices and ten core tensors since there are ten phases. In order to avoid directly storing the core tensor $\underline{R_i}$ and factor matrices F_{1i}, F_{2i}, and F_{3i}, we will be performing the following calculations to approximate each phase and to further compress the data using the two lemmas introduced earlier:

$$\underline{X_i} \approx [\![\underline{R_i}, F_{1i}, F_{2i}, F_{3i}]\!]$$

$$
\begin{aligned}
&= \underline{R_i} \times_1 F_{1i} \times_2 F_{2i} \times_3 F_{3i} \\
&= \underline{R_i} \times_1 U_{A_L} U_{A_L}^T A_i \times_2 U_{B_L} U_{B_L}^T B_i \times_3 U_{C_L} U_{C_L}^T C_i \\
&= \underline{R_i} \times_1 U_{A_L}^T A_i \times_1 U_{A_L} \times_2 U_{B_L} U_{B_L}^T B_i \times_3 U_{C_L} U_{C_L}^T C_i \\
&= \underline{R_i^*} \times_1 U_{A_L} \times_2 U_{B_L} U_{B_L}^T B_i \times_3 U_{C_L} U_{C_L}^T C_i \ (\underline{R_i^*} = \underline{R_i} \times_1 U_{A_L}^T A_i) \\
&= \underline{R_i^*} \times_1 U_{A_L} \times_2 U_{B_L}^T B_i \times_2 U_{B_L} \times_3 U_{C_L} U_{C_L}^T C_i \\
&= \underline{R_i^*} \times_2 U_{B_L}^T B_i \times_1 U_{A_L} \times_2 U_{B_L} \times_3 U_{C_L} U_{C_L}^T C_i \\
&= \underline{R_i^{**}} \times_1 U_{A_L} \times_2 U_{B_L} \times_3 U_{C_L} U_{C_L}^T C_i \ (\underline{R_i^{**}} = \underline{R_i^*} \times_2 U_{B_L}^T B_i) \\
&= \underline{R_i^{***}} \times_1 U_{A_L} \times_2 U_{B_L} \times_3 U_{C_L} \ \ (\underline{R_i^{***}} = \underline{R_i^{**}} \times_3 U_{C_L}^T C_i) \\
&= [\![\underline{R_i^{***}}, U_{A_L}, U_{B_L}, U_{C_L}]\!],
\end{aligned}
$$

where $\underline{R_i^{***}} = \underline{R_i} \times_1 U_{A_L}^T A_i \times_2 U_{B_L}^T B_i \times_3 U_{C_L}^T C_i$.

Through the above operations, we are able to obtain population-specific factor matrices U_{A_L}, U_{B_L}, and U_{C_L} that do not depend on indicator i and phase-specific core tensor $\underline{R_i^{***}}$ that does depend on indicator i.

Chapter 6

Model Performance Comparisons

So far, we have introduced to our readers three motion models. The first one is a PCA-based algorithm. The second one is the matrix-based PVD algorithm. The last one is the tensor-based PTD algorithm. To assess and compare the performance of each of the three models, we are going to first apply these algorithms to both clinical and simulated DVF data. We have one set of clinical full lung DVF data and nine sets of clinical half lung DVF data. On top of that we have four sets of simulated half lung DVF data. The size of each clinical and simulated DVF data is listed below. (Even though the size of the clinical half lung DVF tensors is in the same ballpark, and the size of the simulated half lung DVF tensors is exactly the same, the DVF tensors are still very different in terms of the anatomical shape and size of each patient. Their tumor location and scanning location also differ in machinery and technology. Even in the simulated DVF tensors, the anatomy is intentionally set to differ from patient to patient. All the aforementioned differences are unavoidable and compose major obstacles in combining information across different patients to make an overall cross-patient population model.)

Set	1^{st} mode	2^{nd} mode	3^{rd} mode	4^{th} mode	5^{th} mode
1	300	450	110	10	3

Table 6.1: Size of one Set of Full Lung Clinical DVF

Set	1^{st} mode	2^{nd} mode	3^{rd} mode	4^{th} mode	5^{th} mode
1	210	143	94	10	3
2	230	143	112	10	3
3	222	130	104	10	3
4	246	128	106	10	3
5	326	238	128	10	3
6	280	180	116	10	3
7	300	170	128	10	3
8	278	164	80	10	3
9	304	148	108	10	3

Table 6.2: Sizes of Nine Sets of Half Lung Clinical DVF

Set	1^{st} mode	2^{nd} mode	3^{rd} mode	4^{th} mode	5^{th} mode
1	512	256	100	10	3
2	512	256	100	10	3
3	512	256	100	10	3
4	512	256	100	10	3

Table 6.3: Sizes of Four Sets of Simulated Half Lung DVF

6.1 Model Performance Metrics

The performance of the PCA, PVD, and PTD models will be assessed using multiple criteria. Some of the criteria are adopted from the PCA papers (meanwhile some used in the PCA paper will not be recommended by us). Some of the criteria are newly developed and used in this dissertation. In the PCA paper, authors used MSE on the absolute differences between the reconstructed images and the original images on a voxel level. Authors of the PCA paper argue that smaller MSE values indicate a better model. We argue that MSE is not an ideal metric for measuring model performance due to the following reason: the actual anatomical body parts do not occupy the entire area of an image. On top of that, some anatomical body parts have no or little motions (such as ribs and spine). Therefore the DVF tensor is relatively sparse (with many elements of zero or close to zero values). Many voxels have no change from one phase to the next phase. The following Figure 6.1 contains ten plots of coordinate 3 in transverse slices 61 from the clinical full lung DVF data at all ten phases within a breathing cycle. Intensity of colors represents the magnitude of an element's value in DVF, which in turn represents magnitude of motion of the corresponding element or voxel.

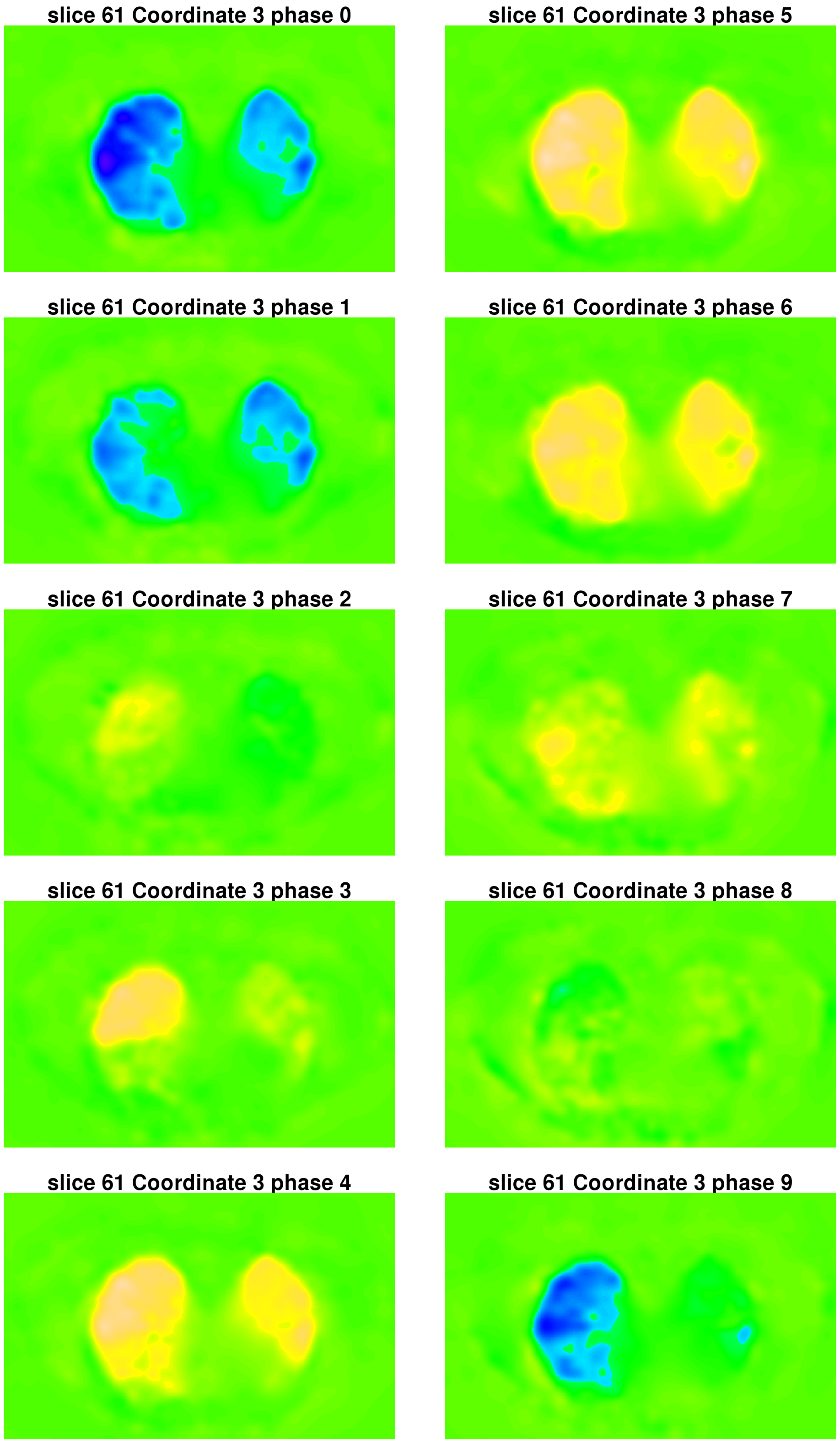

slice 61 Coordinate 3 phase 0
slice 61 Coordinate 3 phase 5
slice 61 Coordinate 3 phase 1
slice 61 Coordinate 3 phase 6
slice 61 Coordinate 3 phase 2
slice 61 Coordinate 3 phase 7
slice 61 Coordinate 3 phase 3
slice 61 Coordinate 3 phase 8
slice 61 Coordinate 3 phase 4
slice 61 Coordinate 3 phase 9

Figure 6.1: Full Lung DVF Transverse Slice 61 Coordinate 3 From Phase 0 to Phase 4

The color scales for all plots in Figure 6.1 are the same. As can be observed, the number of pixels whose changes of color from one phase to the next phase that are observable by naked human eyes at this level of resolution is relatively low compared to the total number of pixels in a given slice. This means that the majority of the pixels in any given slices has a change of value of almost zero from one phase to the next. When models are performed and images are reconstructed, these voxels will have an absolute difference between the reconstructed image and the original image of almost zero, which will in turn dilutes the MSE value. Therefore, a good model with relatively accurate reconstructed images compared to the original images will have small MSE. However, a small MSE does not imply a good model (which can be observed in the tables which will be presented in this chapter later). Due to the aforementioned reason, we do not recommend using MSE as a metrics for model performance, especially in terms of judging the accuracy of the reconstructed images compared to the original images. With that being said, we will still record the MSE values in case readers are interested.

Instead of MSE, the performance of each models will be assessed in the following aspects and compared using the following metrics:

1. Data compression level: Each algorithm is able to compress the tensor data and store the information in the original tensor data using either matrices or tensors of much smaller sizes. Therefore we will record the reduced data size (we will refer to this as the number of free parameters) together with a percentage where the reduced data size is compared to the original image size;

2. Accuracy of reconstructed images: This criterion is related to the first one. As the accuracy of the reconstructed images are dependent on the level of data compression, which is in turn controlled by the level of percentage of variance explained set in each data reduction step. Therefore we will record the percentage of variance explained in each data reduction step; For the PCA algorithm, there is a one step data reduction, so there will be one percentage

recorded. Whereas for the PVD and PTD algorithms, there are two percentages recorded since they are two-step data reduction algorithms (refer to Chapter 4 and 5), and the total percentage of variance explained after the two-step data reductions is difficult to summarize using one percentage (we will give a more detailed explanation of this in Chapter 7);

3. Count of voxels with large absolute difference: Since we are concerned about voxels with large absolute difference between the reconstructed and the original images, we will record the number of voxels whose absolute difference between the reconstructed and original images is greater than a certain threshold (in our analysis, we pick the threshold to be 0.15 cm and 0.20 cm, which we believe are reasonable but can definitely be adjusted if needed).

4. Computation time: CPU time. We record CPU time just to make sure that each algorithm can complete in a reasonable amount of time. (We believe that the new algorithms (PVD and PTD) can be sped up by using more advanced decomposition algorithms and parallel computing. But speeding up the algorithm is not the focus of this dissertation.) The architecture of the computer we used for testing all three models are listed below:

   a) 54 Dell PE R620/R630 servers with CentOS 6.8 and 7.3 64 bits Linux OS

   b) 1336 cores /2672 Threads using Intel Xeon 56XX processors (2.67GHz to 3.4GHz)

   c) 9TB RAM (128GB-200GB per node)

   d) 360 TB parallel file system network storage running Intel IEEL lustre with 100TB backup storage with InfiniBand connection.

   e) 30TB internal disk storage (120GB-900GB per node SSD drives)

   f) 40GB InfiniBand network connections to all nodes and storage.

   g) 380TB Network backup storage running daily backup

   h) Fail-over redundant master servers

6.2 Performance of PCA based Motion Model

Here we first present the result of PCA-based algorithm on the full lung DVF dataset. The size of this set of DVF can be found in Table 4.3. For this model, we follow the guidelines in the PCA paper [Li et al., 2011] and kept two eigenvectors. Because there are a total of 10 phases, keeping two eigenvectors will result in storing compressed data that is 20% of the original data size. Therefore this model will use 20% of the original data size to approximate the entire DVF data. In fact every PCA model we performed follows this guideline provided in the PCA paper [Li et al., 2011], therefore for both the 9 sets of half lung DVF and the 4 sets of simulated DVF, the percentage of the free parameters, which is the size of the compressed data, is always 20% of the original DVF data size. The result of applying PCA model on the full lung DVF is presented below in Table 6.4. Table 6.4 summarizes the performance of PCA model in the four aspects we described in the previous section 'Model Performance Metrics'. We also recorded the MSE in case readers are interested, but again we do not recommend using MSE as a model performance metric.

	PCA on Full Lung DVF
Free Parameters (%)	89100000 (20%)
Percentage of Variance Explained	89%
0.15 cm - Max (‰)	7231415 (16.23‰)
0.20 cm - Max (‰)	3423442 (7.68‰)
Max (cm)	1.58
MSE	$1.90e10^{-3}$
CPU time (sec)	367

Table 6.4: Performance of PCA on Clinical Full Lung DVF

Here we present the results of PCA based algorithm on 9 sets of half lung DVF datasets. For the majority of the half lung DVF (besides set 1 and 4), keeping two Eigen vectors is able to explain more than 80% of the total variance.

For the simulated DVF, we perform the model on the noisy DVF, and the voxel level absolute

PCA Performance on Half Lung	DVF Set 1	DVF Set 2	DVF Set 3
Free Parameters (%)	16936920 (20%)	22102080 (20%)	18008640 (20%)
Percentage of Variance Explained	49.5%	86.8%	87.5%
0.15 cm - Max (‰)	1484809 (17.53‰)	535741 (4.85‰)	263257 (2.92‰)
0.20 cm - Max (‰)	645274 (7.62‰)	227163 (2.06‰)	135149 (1.50‰)
Max (cm)	14.25	3.52	3.64
MSE	$5.39e10^{-3}$	$1.06e10^{-3}$	$1.03e10^{-3}$
CPU time (sec)	24	42	46

Table 6.5: Performance of PCA on Clinical Half Lung DVF Set 1, 2, and 3

PCA Performance on Half Lung	DVF Set 4	DVF Set 5	DVF Set 6
Free Parameters (%)	20026368 (20%)	59587584 (20%)	35078400 (20%)
Percentage of Variance Explained	59.3%	87.9%	90.6%
0.15 cm - Max (‰)	1689800 (16.88‰)	1675138 (5.62‰)	437299 (2.49‰)
0.20 cm - Max (‰)	905234 (9.04‰)	879379(2.95‰)	159943 (0.91‰)
Max (cm)	2.09	3.04	1.26
MSE	$2.42e10^{-3}$	$1.03e10^{-3}$	$0.68e10^{-3}$
CPU time (sec)	37	122	167

Table 6.6: Performance of PCA on Clinical Half Lung DVF Set 4, 5, and 6

PCA Performance on Half Lung	DVF Set 7	DVF Set 8	DVF Set 9
Free Parameters (%)	39168000 (20%)	21884160 (20%)	29154816 (20%)
Percentage of Variance Explained	94.95%	82.5%	93.7%
0.15 cm - Max (‰)	660726 (3.37‰)	790609 (7.23‰)	317941 (2.18‰)
0.20 cm - Max (‰)	266175 (1.36‰)	273182 (2.50‰)	94871 (0.65‰)
Max (cm)	0.95	1.94	0.45
MSE	$0.67e10^{-3}$	$1.11e10^{-3}$	$0.64e10^{-3}$
CPU time (sec)	241	42	59

Table 6.7: Performance of PCA on Clinical Half Lung DVF Set 7, 8 , and 9

error is calculated by comparing the reconstructed DVF to the noiseless DVF. It is worth mentioning that in all 4 sets of simulated DVF, PCA performs extremely well in terms of the percentage of variance explained (above 99%) by keeping two Eigen vectors.

PCA on Simulated Half Lung	DVF Set 1	DVF Set 2
Free Parameters (%)	78643200 (20%)	78643200 (20%)
Percentage of Variance Explained	99.8%	99.9%
0.15 cm - Max (‰)	29442668 (74.8‰)	157869 (0.40‰)
0.20 cm - Max (‰)	12166792 (30.9‰)	54068 (0.14‰)
Max (cm)	0.49	0.27
MSE	$7.57e10^{-3}$	$1.59e10^{-3}$
CPU time (sec)	811	1299

Table 6.8: Performance of PCA on Simulated Half Lung DVF Set 1 and 2

PCA on Simulated Half Lung	DVF Set 3	DVF Set 4
Free Parameters (%)	78643200 (20%)	78643200 (20%)
Percentage of Variance Explained	99.8%	99.6%
0.15 cm - Max (‰)	20471819 (52.1‰)	72281 (0.18‰)
0.20 cm - Max (‰)	6517063 (16.6‰)	0 (0‰)
Max (cm)	0.45	0.19
MSE	$6.14e10^{-3}$	$1.22e10^{-3}$
CPU time (sec)	1163	325

Table 6.9: Performance of PCA on Simulated Half Lung DVF Set 3 and 4

6.3 Performance of Revised Population Value Decomposition

In this section, we present the results of revised PVD algorithm (which we will simply call the PVD or the PVD algorithm) on the clinical full lung DVF, clinical half lung DVF, and simulated DVF dataset. Because the PVD algorithm is a two-step data reduction method, we need to choose the bottom threshold for the percentage of variance explained in each data reduction step. The total percentage of variance explained is hard to describe and summarize with one number. We are unable to provide a formula for the total percentage of variance explained after two-step data reduction, but believe that the result might take a function form of the two percentages chosen in both data reduction steps. Therefore we recorded the two percentages in both data reduction steps in the performance tables. Once the bottom thresholds of the percentage of variance explained are determined for both data reduction steps, the algorithm is able to find the minimum number of singular values needed to meet the percentage threshold, and only keep the selected leading singular values with their corresponding singular vectors.

In order to obtain a result from the PVD model, which can be later fairly compared to the result of PCA model, we originally tried to match the total number and percentage of free parameters between the PCA and PVD algorithm. The idea behind doing so is that we would like to see when the data is compressed to the same size by both algorithms, which algorithm is able to achieve a more accurate approximation by looking at the rest of the performance metrics including the number of voxels with an absolute difference between the reconstructed and original images greater than the set thresholds (0.15 cm and 0.20 cm). However, based on our empirical experience, we realize that matching the number of free parameters is not necessary, because the PVD algorithm uniformly achieves better approximation with fewer free parameters on every set of clinical DVF data tested. We briefly discussed the comparison here so that reader will understand the original rationale of how we choose the percentages of variance explained in both data reduction steps in the PVD algorithm. We are not alleging that the same or similar percentage should be used on all DVF data. We will further discuss the comparisons among different model results in Chapter 7. Again MSE is recorded in case readers are interested.

	PVD on Full Lung DVF
Free Parameters (%)	1882650 (0.42%)
Percentage of Variance Explained	98% & 98%
0.15 cm - Max (‰)	186746 (0.42‰)
0.20 cm - Max (‰)	39668 (0.09‰)
Max (cm)	0.85
MSE	$3.93e10^{-4}$
CPU time (sec)	1383

Table 6.10: Performance of PVD on Clinical Full Lung DVF

Next we present the results of PVD algorithm on 9 sets of clinical half lung DVF datasets. For each set of DVF in this category, we choose a similar but yet slightly different level of percentage of variance explained in both data reduction steps in the PVD algorithm, which are recorded.

PVD Performance on Half Lung	DVF Set 1	DVF Set 2	DVF Set 3
Free Parameters (%)	513846 (0.61%)	1795674 (1.62%)	1091370 (1.21%)
Percentage of Variance Explained	95% & 95%	99% & 99%	98% & 98%
0.15 cm - Max (‰)	287850 (3.40‰)	4896 (0.04‰)	13851 (0.15‰)
0.20 cm - Max (‰)	153843 (1.82‰)	2219 (0.02‰)	5331 (0.06‰)
Max (cm)	4.30	0.79	0.50
MSE	$0.97e10^{-3}$	$0.10e10^{-3}$	$0.20e10^{-3}$
CPU time (sec)	102	153	109

Table 6.11: Performance of PVD on Clinical Half Lung DVF Set 1, 2, and 3

PVD Performance on Half Lung	DVF Set 4	DVF Set 5	DVF Set 6
Free Parameters (%)	6656372 (6.64%)	1944954 (0.65%)	1442260 (0.82%)
Percentage of Variance Explained	95% & 95%	97% & 97%	98% & 98%
0.15 cm - Max (‰)	110782 (1.11‰)	146462 (0.49‰)	15586 (0.09‰)
0.20 cm - Max (‰)	38808 (0.39‰)	38947 (0.13‰)	4098 (0.02‰)
Max (cm)	0.98	0.67	0.40
MSE	$0.40e10^{-3}$	$0.31e10^{-3}$	$0.16e10^{-3}$
CPU time (sec)	202	820	369

Table 6.12: Performance of PVD on Clinical Half Lung DVF Set 4, 5, and 6

PVD Performance on Half Lung	DVF Set 7	DVF Set 8	DVF Set 9
Free Parameters (%)	720910 (0.37%)	784506 (0.72%)	622216 (0.43%)
Percentage of Variance Explained	97% & 97%	97% & 97%	96% & 96%
0.15 cm - Max (‰)	169894 (0.87‰)	16921 (0.15‰)	130428 (0.89‰)
0.20 cm - Max (‰)	38697 (0.20‰)	3277 (0.03‰)	30781 (0.21‰)
Max (cm)	0.49	0.37	0.42
MSE	$0.43e10^{-3}$	$0.27e10^{-3}$	$0.40e10^{-3}$
CPU time (sec)	385	236	264

Table 6.13: Performance of PVD on Clinical Half Lung DVF Set 7, 8, and 9

For the simulated DVF, we perform the PVD model on the noisy DVF, and the absolute error is calculated by comparing the reconstructed DVF to the noiseless DVF.

PVD on Simulated Half Lung	DVF Set 1	DVF Set 2
Free Parameters (%)	875488 (0.22%)	935744 (0.24%)
Percentage of Variance Explained	99.9% & 99.9%	99.9% & 99.9%
0.15 cm - Max (‰)	29892842 (76.0‰)	5210 (0.013‰)
0.20 cm - Max (‰)	12615631 (32.1‰)	0 (0‰)
Max (cm)	0.50	0.16
MSE	$7.60e10^{-3}$	$1.61e10^{-3}$
CPU time (sec)	1270	1375

Table 6.14: Performance of PVD on Simulated Half Lung DVF Set 1 and 2

PVD on Simulated Half Lung	DVF Set 3	DVF Set 4
Free Parameters (%)	875488 (0.22%)	935744 (0.24%)
Percentage of Variance Explained	99.9% & 99.9%	99.9% & 99.9%
0.15 cm - Max (‰)	29892842 (76.0‰)	5210 (0.013‰)
0.20 cm - Max (‰)	12615631 (32.1‰)	0 (0‰)
Max (cm)	0.50	0.16
MSE	$7.60e10^{-3}$	$1.61e10^{-3}$
CPU time (sec)	1270	1375

Table 6.15: Performance of PVD on Simulated Half Lung DVF Set 3 and 4

6.4 Performance of Population Tucker Decomposition

In this section, we present the results of PTD algorithm on the clinical full lung DVF, clinical half lung, and simulated half lung dataset. In the previous section, we mentioned that for the PVD algorithm, we need to choose the bottom threshold for percentage of variance explained in both data reduction steps. Once the threshold is set, PVD is able to find the minimum number of singular values and vectors needed to achieve that threshold. In turn, the number of singular values and vectors determine the compressed data size. However, in the case of PTD, the logic is reversed in the first-step data reduction. In the first-step data reduction in PTD, Tucker decomposition is used, and we will need to decided the core tensor size, which in turns determines the size of the compressed data. Once the core tensor size is given, the algorithm using HOOI is guaranteed to find the best low multilinear rank approximation of the original tensor in terms of the Frobenius norm (a cost function introduced in Chapter 5). Because at step one data reduction, we perform a tucker decomposition on each of ten third-order tensors, one at each phase (refer to Chapter 5 regarding algorithm steps and process). When the size of the core tensor is determined first, the compressed tensor will have different percentage of norm explained for third-order tensor at different phase. The percentage of norm explained is calculated through the following steps:

$$percent_{norm} = 1 - \frac{fnorm(residual)}{fnorm(oDVF)},$$

where, $fnorm(residual)$ is the Frobenius norm of the error, which is the difference between the approximated DVF (aDVF) and the original DVF (oDVF) and $fnorm(oDVF)$ is the Frobenius norm of the original DVF (oDVF). At this point, we do not know a principled approach to choose the optimal core tensor size. The chosen core tensor sizes in the first step is through trial and error. In the second data reduction step, the logic of PTD is the same as that of PVD.

	PTD on Full Lung DVF
Free Parameters (%)	742420(0.17%)
Percentage of Variance Explained	91-95% & 95%
0.15 cm - Max (‰)	7873 (0.02‰)
0.20 cm - Max (‰)	2623 (0.006‰)
Max (cm)	0.42
MSE	$8.30e10^{-5}$
CPU time (sec)	4428

Table 6.16: Performance of PTD on Full Lung DVF

Here we present the results of PTD algorithm on 9 sets of half lung DVF datasets.

PTD Performance on Half lung	DVF Set 1	DVF Set 2	DVF Set 3
Free Parameters (%)	1017575 (1.2%)	1263507 (1.14%)	1351714 (1.5%)
Percentage of Variance Explained	>99% & 95%	90-95% & 95%	94-97% & 95%
0.15 cm - Max (‰)	0 (0‰)	2848 (0.03‰)	942 (0.01‰)
0.20 cm - Max (‰)	0 (0‰)	1506 (0.01‰)	250 (0.003‰)
Max (cm)	0.07	0.87	0.29
MSE	$1.40e10^{-6}$	$7.13e10^{-5}$	$2.76e10^{-5}$
CPU time (sec)	654	2976	1254

Table 6.17: Performance of PTD on Simulated Half Lung DVF Set 1, 2, and 3

PTD Performance on Half lung	DVF Set 4	DVF Set 5	DVF Set 6
Free Parameters (%)	1477784 (1.5%)	771042 (0.29%)	1201134 (0.68%)
Percentage of Variance Explained	40-73% & 95%	89-94% & 95%	92-97% & 95%
0.15 cm - Max (‰)	128470 (1.28‰)	6811 (0.02‰)	62 (0.0003‰)
0.20 cm - Max (‰)	50106 (0.50‰)	2178 (0.007‰)	0 (0‰)
Max (cm)	1.49	0.41	0.17
MSE	$0.83e10^{-3}$	$7.27e10^{-5}$	$2.57e10^{-5}$
CPU time (sec)	3960	5040	2820

Table 6.18: Performance of PTD on Half Lung DVF Set 4, 5, and 6

PTD Performance on Half lung	DVF Set 7	DVF Set 8	DVF Set 9
Free Parameters (%)	1228812 (0.63%)	1380806 (1.26%)	1329818 (0.91%)
Percentage of Variance Explained	93-97% & 95%	94-96% & 95%	94-98% & 95%
0.15 cm - Max (‰)	484 (0.002‰)	44 (0.0004‰)	0 (0‰)
0.20 cm - Max (‰)	58 (0.0003‰)	0 (0‰)	0 (0‰)
Max (cm)	0.24	0.16	0.07
MSE	$3.39e10^{-5}$	$3.20e10^{-5}$	$2.08e10^{-5}$
CPU time (sec)	3600	1644	2406

Table 6.19: Performance of PTD on Half Lung DVF Set 7, 8, and 9

For the simulated DVF, we perform the PTD model on the noisy DVF, and the absolute error is calculated by comparing the reconstructed DVF to the noiseless DVF.

PTD on Simulated Half Lung	DVF Set 1	DVF Set 2
Free Parameters (%)	6030504 (1.53%)	4307676 (1.10%)
Percentage of Variance Explained	100% & 95%	100% & 95%
0.15 cm - Max (‰)	29368597 (74.7‰)	5000 (0.013‰)
0.20 cm - Max (‰)	12482683 (31.7‰)	0 (0‰)
Max (cm)	0.50	0.16
MSE	$7.55e10^{-3}$	$1.60e10^{-3}$
CPU time (sec)	72720	71352

Table 6.20: Performance of PTD on Simulated Half Lung DVF Set 1 and 2

PTD on Simulated Half Lung	DVF Set 3	DVF Set 4
Free Parameters (%)	5965580 (1.52%)	5430568 (1.38%)
Percentage of Variance Explained	100% & 95%	100% & 95%
0.15 cm - Max (‰)	20816848 (52.9‰)	42572 (0.11‰)
0.20 cm - Max (‰)	6913448 (17.6‰)	0 (0‰)
Max (cm) 0.47	0.19	
MSE	$6.13e10^{-3}$	$1.23e10^{-3}$
CPU time (sec)	74052	51912

Table 6.21: Performance of PTD on Simulated Half Lung DVF Set 3 and 4

Chapter 7

Discussion on Model Performance and Future Work

7.1 Chapter Layout

In this chapter, we will be discussing and comparing the performances of PCA, PVD, and PTD algorithms. Meanwhile we will present some advantages and drawbacks of PVD and PTD based on both theoretical reasons and empirical experience. At the end of each section, we will briefly give some interpretation of the results for PVD and PTD algorithms and discuss some future work that can be done in this area of research.

7.2 Discussion on Revised Population Value Decomposition

Based on the table comparisons shown in the previous chapter, PVD shows consistent better results in terms of data compression and accuracy of approximation on the clinical DVF tensors compared to the PCA algorithm. The PVD algorithm runs about 4 times longer than PCA, but we believe 23 minutes (as in the case of clinical full lung DVF) is still reasonably fast. We will discuss the time consumption of the algorithm in this section later.

Below we present to readers the performance comparison between PCA and PVD on the full lung DVF set. As can be seen in Table 7.1, PVD requires less free parameters to achieve a higher percentage of variance explained. The numbers of elements with an absolute difference between the reconstructed and original DVF above the 0.15 cm and 0.20 cm thresholds are much smaller in PVD than those in PCA. PVD also reduces the maximum absolute difference (measured by Max (cm) to 0.85cm almost half of PCA.

	PCA on Full Lung	PVD on Full Lung
Free Parameters (%)	89100000 (20%)	1882650 (0.42%)
% of Variance Explained	89%	98% & 98%
0.15 cm - Max (‰)	7231415 (16.23‰)	186746 (0.42‰)
0.20 cm - Max (‰)	3423442 (7.68‰)	39668 (0.09‰)
Max (cm)	1.58	0.85
MSE	$1.90e10^{-3}$	$3.93e10^{-4}$
CPU time (min)	6.12	23.05

Table 7.1: Performance of PCA and PVD on Full Lung DVF

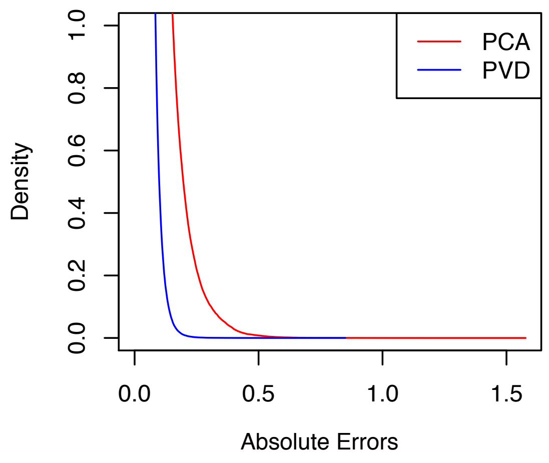


Figure 7.1: Tail Portion Density Plot of the Absolute Error for PCA and PVD on Full Lung DVF

Besides the summary tables, we will also attach a tail-portion density plot of the absolute error between the reconstructed DVF and the original DVF. In the density plot, the x-axis is the absolute error on a voxel level between the reconstructed and original DVF. The red curve represents the density of absolute error for PCA algorithm, and the blue curve represents the density of absolute error for PVD algorithm. As shown in Figure 7.1, the tail portion of the PVD curve is below the PCA curve suggesting less frequent large deviation of PVD over PCA.

Here we will show the table and tail portion density plot comparisons of PCA and PVD on all half-lung DVF, but we will not give further discussion on each individual set of DVF since the performance comparisons between PCA and PVD are very consistent with the comparison for the full lung DVF.

	PCA on Half Lung	PVD on Half Lung
Free Parameters (%)	16936920 (20%)	513846 (0.61%)
% of Variance Explained	49.5%	95% & 95%
0.15 cm - Max (‰)	1484809 (17.53‰)	287850 (3.40‰)
0.20 cm - Max (‰)	645274 (7.62‰)	153843 (1.82‰)
Max (cm)	14.25	4.30
MSE	$5.39e10^{-3}$	$0.97e10^{-3}$
CPU time (sec)	24	102

Table 7.2: Performance of PCA and PVD on Half Lung DVF Set 1

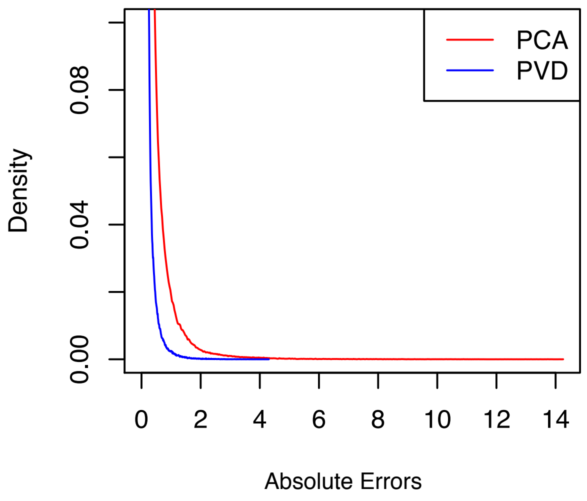


Figure 7.2: Tail Portion Density Plot of the Absolute Error for PCA and PVD on Half Lung DVF Set 1

	PCA on Half Lung	PVD on Half Lung
Free Parameters (%)	22102080 (20%)	1795674 (1.62%)
% of Variance Explained	86.8%	99% & 99%
0.15 cm - Max (‰)	535741 (4.85‰)	4896 (0.04‰)
0.20 cm - Max (‰)	227163 (2.06‰)	2219 (0.02‰)
Max (cm)	3.52	0.79
MSE	$1.06e10^{-3}$	$0.10e10^{-3}$
CPU time (sec)	42	153

Table 7.3: Performance of PCA and PVD on Half Lung DVF Set 2

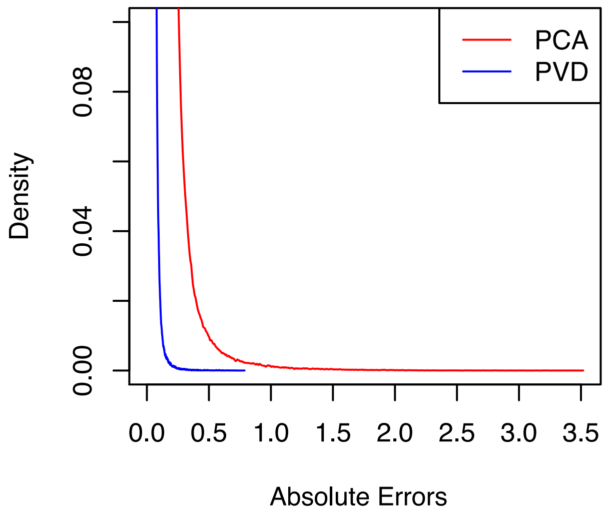


Figure 7.3: Tail Portion Density Plot of the Absolute Error for PCA and PVD on Half Lung DVF Set 2

	PCA on Half Lung	PVD on Half Lung
Free Parameters (%)	18008640 (20%)	1091370 (1.21%)
% of Variance Explained	87.5%	98% & 98%
0.15 cm - Max (‰)	263257 (2.92‰)	13851 (0.15‰)
0.20 cm - Max (‰)	135149 (1.50‰)	5331 (0.06‰)
Max (cm)	3.64	0.50
MSE	$1.03e10^{-3}$	$0.20e10^{-3}$
CPU time (sec)	46	109

Table 7.4: Performance of PCA and PVD on Half Lung DVF Set 3

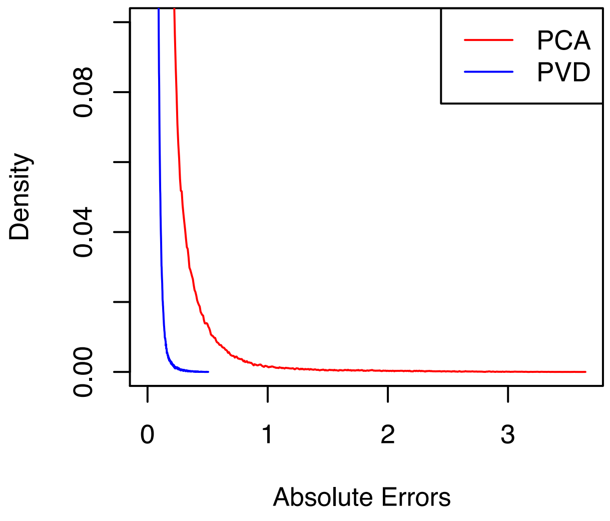


Figure 7.4: Tail Portion Density Plot of the Absolute Error for PCA and PVD on Half Lung DVF Set 3

	PCA on Half Lung	PVD on Half Lung
Free Parameters (%)	20026368 (20%)	6656372 (6.64%)
% of Variance Explained	59.3%	95% & 95%
0.15 cm - Max (‰)	1689800 (16.88‰)	110782 (1.11‰)
0.20 cm - Max (‰)	905234 (9.04‰)	38808 (0.39‰)
Max (cm)	2.09	0.98
MSE	$2.42e10^{-3}$	$0.40e10^{-3}$
CPU time (sec)	37	202

Table 7.5: Performance of PCA and PVD on Half Lung DVF Set 4

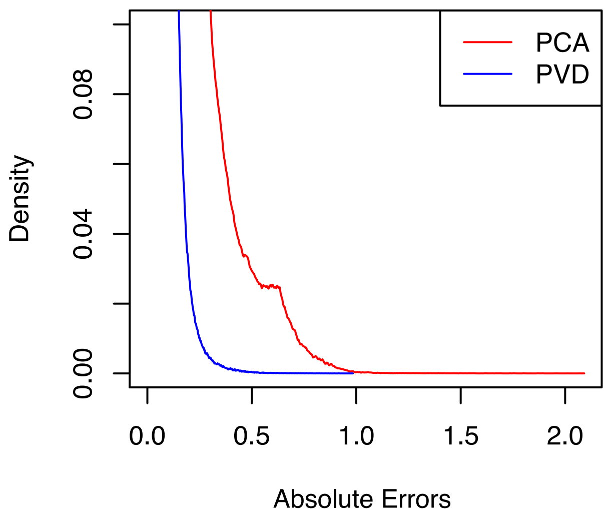


Figure 7.5: Tail Portion Density Plot of the Absolute Error for PCA and PVD on Half Lung DVF Set 4

	PCA on Half Lung	PVD on Half Lung
Free Parameters (%)	59587584 (20%)	1944954 (0.65%)
% of Variance Explained	87.9%	97% & 97%
0.15 cm - Max (‰)	1675138 (5.62‰)	146462 (0.49‰)
0.20 cm - Max (‰)	879379(2.95‰)	38947 (0.13‰)
Max (cm)	3.04	0.67
MSE	$1.03e10^{-3}$	$0.31e10^{-3}$
CPU time (sec)	122	820

Table 7.6: Performance of PCA on PVD on Half Lung DVF Set 5

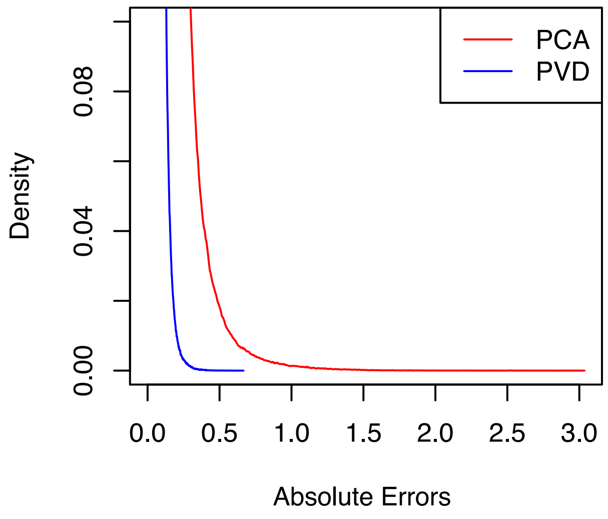


Figure 7.6: Tail Portion Density Plot of the Absolute Error for PCA and PVD on Half Lung DVF Set 5

	PCA on Half Lung	PVD on Half Lung
Free Parameters (%)	35078400 (20%)	1442260 (0.82%)
% of Variance Explained	90.6%	98% & 98%
0.15 cm - Max (‰)	437299 (2.49‰)	15586 (0.09‰)
0.20 cm - Max (‰)	159943 (0.91‰)	4098 (0.02‰)
Max (cm)	1.26	0.40
MSE	$0.68e10^{-3}$	$0.16e10^{-3}$
CPU time (sec)	167	369

Table 7.7: Performance of PCA and PVD on Half Lung DVF Set 6

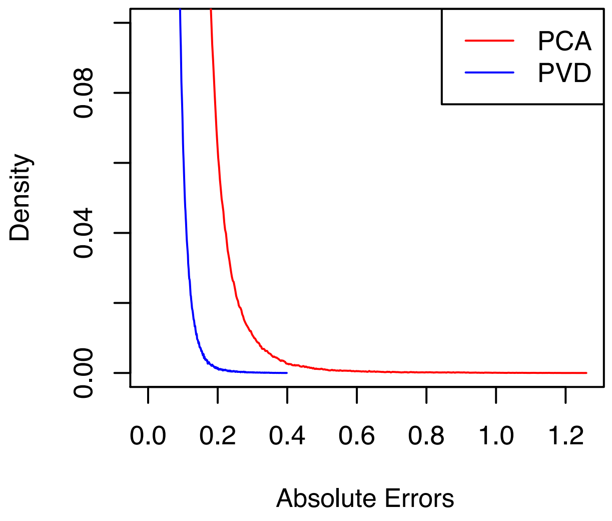


Figure 7.7: Tail Portion Density Plot of the Absolute Error for PCA and PVD on Half Lung DVF Set 6

	PCA on Half Lung	PVD on Half Lung
Free Parameters (%)	39168000 (20%)	720910 (0.37%)
% of Variance Explained	94.95%	97% & 97%
0.15 cm - Max (‰)	660726 (3.37‰)	169894 (0.87‰)
0.20 cm - Max (‰)	266175 (1.36‰)	38697 (0.20‰)
Max (cm)	0.95	0.49
MSE	$0.67e10^{-3}$	$0.43e10^{-3}$
CPU time (sec)	241	385

Table 7.8: Performance of PCA and PVD on Half Lung DVF Set 7

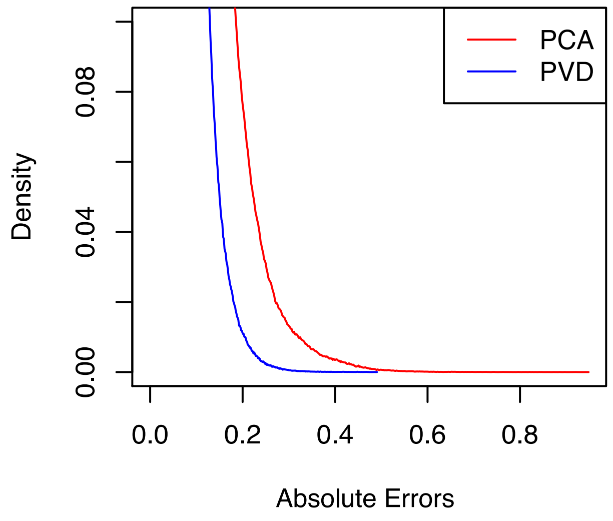


Figure 7.8: Tail Portion Density Plot of the Absolute Error for PCA and PVD on Half Lung DVF Set 7

	PCA on Half Lung	PVD on Half Lung
Free Parameters (%)	21884160 (20%)	784506 (0.72%)
% of Variance Explained	82.5%	97% & 97%
0.15 cm - Max (‰)	790609 (7.23‰)	16921 (0.15‰)
0.20 cm - Max (‰)	273182 (2.50‰)	3277 (0.03‰)
Max (cm)	1.94	0.37
MSE	$1.11e10^{-3}$	$0.27e10^{-3}$
CPU time (sec)	42	264

Table 7.9: Performance of PCA and PVD on Half Lung DVF Set 8

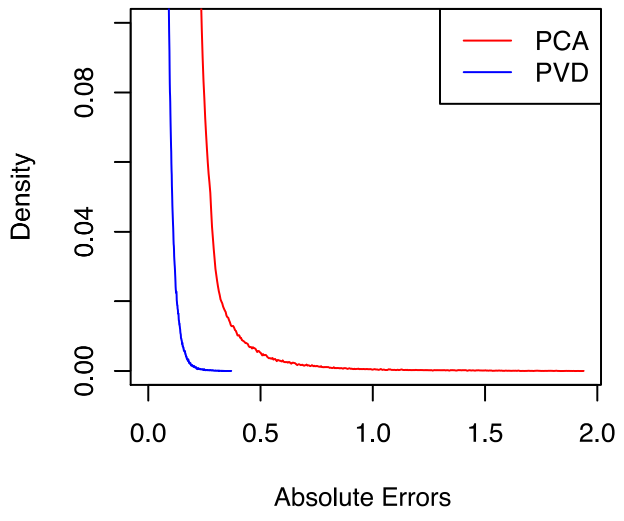


Figure 7.9: Tail Portion Density Plot of the Absolute Error for PCA and PVD on Half Lung DVF Set 8

	PCA on Half Lung	PVD on Half Lung
Free Parameters (%)	29154816 (20%)	622216 (0.43%)
% of Variance Explained	93.7%	96% & 96%
0.15 cm - Max (‰)	317941 (2.18‰)	130428 (0.89‰)
0.20 cm - Max (‰)	94871 (0.65‰)	30781 (0.21‰)
Max (cm)	0.45	0.42
MSE	$0.64e10^{-3}$	$0.40e10^{-3}$
CPU time (sec)	59	264

Table 7.10: Performance of PCA and PVD on Half Lung DVF Set 9

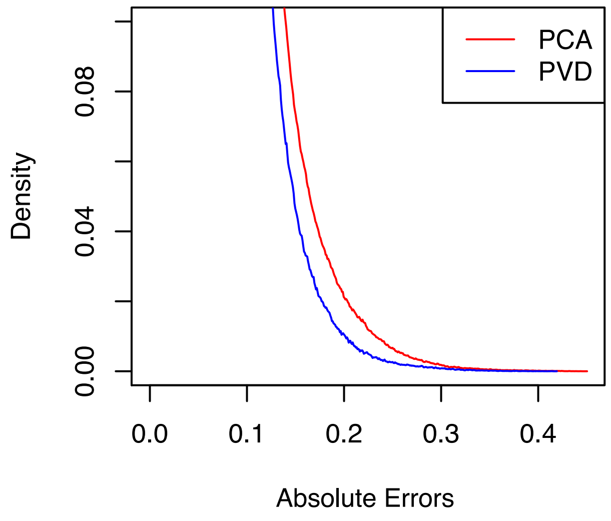


Figure 7.10: Tail Portion Density Plot of the Absolute Error for PCA and PVD on Half Lung DVF Set 9

Now, we present the table comparisons between PCA and PVD on simulated DVF tensors followed by tail portion distribution on absolute error on a voxel level.

Simulated DVF Set 1	PCA	PVD
Free Parameters (%)	78643200 (20%)	875488 (0.22%)
% of Variance Explained	99.8%	99.9% & 99.9%
0.15 cm - Max (‰)	29442668 (74.8‰)	29892842 (76.0‰)
0.20 cm - Max (‰)	12166792 (30.9‰)	12615631 (32.1‰)
Max (cm)	0.49	0.50
MSE	$7.57e10^{-3}$	$7.60e10^{-3}$
CPU time (sec)	811	1270

Table 7.11: Performance of PCA and PVD on Simulated DVF Set 1

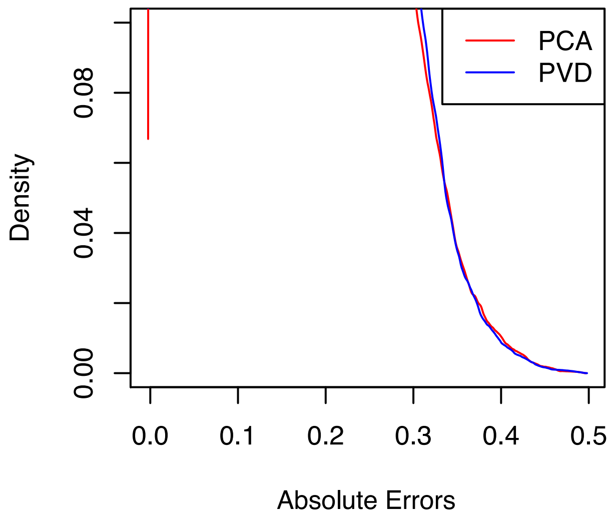


Figure 7.11: Tail Portion Density Plot of the Absolute Error for PCA and PVD on Simulated Lung DVF Set 1

Simulated DVF Set 2	PCA	PVD
Free Parameters (%)	78643200 (20%)	935744 (0.24%)
% of Variance Explained	99.9%	99.9% & 99.9%
0.15 cm - Max (‰)	157869 (0.40‰)	5210 (0.013‰)
0.20 cm - Max (‰)	54068 (0.14‰)	0 (0‰)
Max (cm)	0.27	0.16
MSE	$1.59e10^{-3}$	$1.61e10^{-3}$
CPU time (sec)	1299	1375

Table 7.12: Performance of PCA and PVD on Simulated DVF Set 2

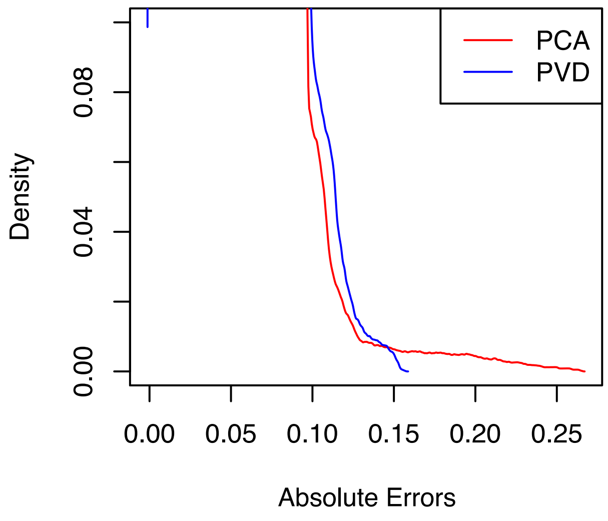


Figure 7.12: Tail Portion Density Plot of the Absolute Error for PCA and PVD on Simulated Lung DVF Set 2

Simulated DVF Set 3	PCA	PVD
Free Parameters (%)	78643200 (20%)	920208 (0.23%)
% of Variance Explained	99.8%	99.9% & 99.9%
0.15 cm - Max (‰)	20471819 (52.1‰)	21221841 (54.0‰)
0.20 cm - Max (‰)	6517063 (16.6‰)	7024940 (17.9‰)
Max (cm)	0.45	0.47
MSE	$6.14e10^{-3}$	$6.19e10^{-3}$
CPU time (sec)	1163	798

Table 7.13: Performance of PCA and PVD on Simulated DVF Set 3

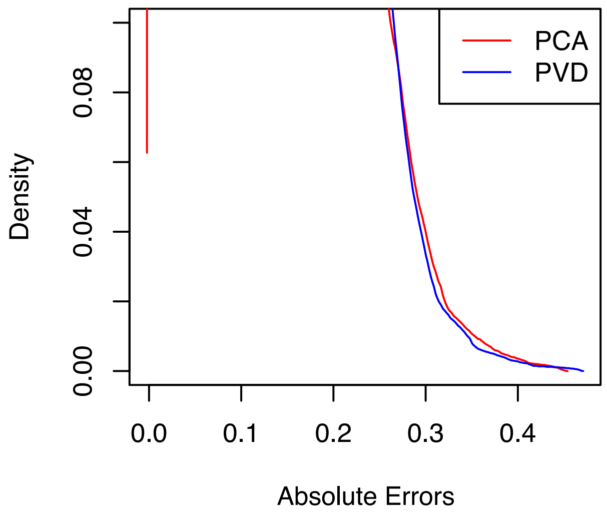


Figure 7.13: Tail Portion Density Plot of the Absolute Error for PCA and PVD on Simulated Lung DVF Set 3

Simulated DVF Set 4	PCA	PVD
Free Parameters (%)	78643200 (20%)	920208 (0.23%)
% of Variance Explained	99.6%	99.9% & 99.9%
0.15 cm - Max (‰)	72281 (0.18‰)	43065 (0.11‰)
0.20 cm - Max (‰)	0 (0‰)	0 (0‰)
Max (cm)	0.19	0.18
MSE	$1.22e10^{-3}$	$1.23e10^{-3}$
CPU time (sec)	1163	1015

Table 7.14: Performance of PCA and PVD on Simulated DVF Set 4

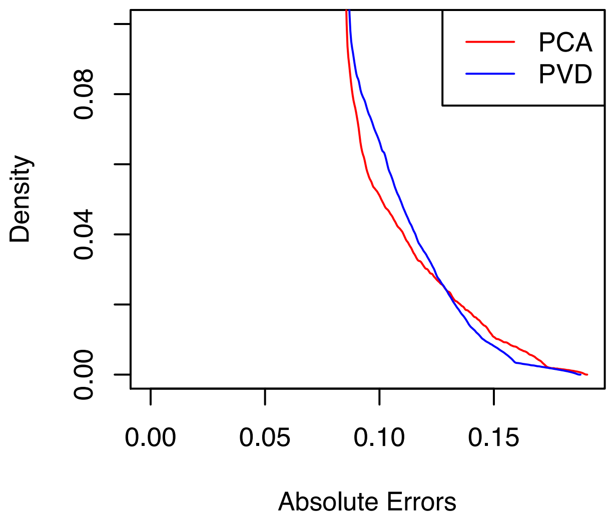


Figure 7.14: Tail Portion Density Plot of the Absolute Error for PCA and PVD on Simulated Lung DVF Set 4

The performance on the simulated DVF tensors based on our metrics are practically indistinguishable between PCA and PVD. But we suggest readers put more weight on the results from clinical DVF tensors. Because in all four cases of simulated DVF, the percentage of variance explained are all very high (>99%) in both PCA and PVD, which is not realistic because neither algorithms are able to achieve this level when applied to the clinical DVF tensors. The phantom DVF is too simple and deterministic, whereas the real human respiratory motion is way more complicated than a phantom can mimic.

For the clinical DVF tensors, the advantages of PVD algorithm over PCA results from retaining more structure of the DVF tensor data by avoiding vectorizing the DVF tensor. PVD exploits features in the first two modes of the DVF tensor data using an iterative matrix-based algorithm (SVD) and shows better result than PCA in terms of data compression and accuracy of approximation. The time consumption of PVD is consistently higher than PCA on each case of clinical DVF we tested, because PCA algorithm vectorizes the entire DVF tensor and performs only one SVD, whereas PVD performs a series of SVD on individual two-dimensional slices. The number of SVD performed depends on the DVF tensor size. We will not repeat the exact algorithm steps and process. Readers can refer to Chapter 4 for details. PVD does require more CPU time, but it is still within practical limit, which makes it usable in clinical setting. We also want to give a brief discussion on the interpretation of the results of the PVD algorithm from a mathematical and statistical point of view. Each SVD in the first step of PVD finds large components (leading left singular vectors) in the subspace spanned by the rows of a particular transverse slice and large components (leading right singular vectors) in the subspace spanned by the columns of the same transverse slice under a given phase. Therefore, the first step in PVD is to find spatial components (mode 1 and 2 in the DVF). In the second step of PVD, the combined U and V matrices contains large mode-1 and mode-2 components across transverse slices and phases. Therefore, the second step is decomposing both spatial and temporal aspects of DVF. The leading left and right singular vectors are for the subspace spanned by both the mode-3 (number of transverse slices) and mode-4 fibers (phases) in the DVF, whose clinical interpretation would be hard to understand, since the second step of PVD is not able to separate the spatial and temporal components. This complication of the PVD algorithm is improved in the PTD algorithm, which we will discuss in the next section.

What we are not able to achieve with respect to analyzing DVF is to discover a universal way to determine the optimal percentages of variance explained in both data reduction steps of PVD, which will require a cost function and can be subjective, which we will leave as future work. In our research, the percentages were determined through trial and error.

7.3 Discussion on Population Tucker Decomposition

In this section, we will be discussing the performance of PTD compared to PVD. We will not be discussing the comparisons between PCA and PTD, because the advantage is obvious and we are more interested to see if the progression form PVD, a matrix-based iterative algorithm, to PTD, a tensor-based iterative algorithm, gives us better result in terms of data compression and accuracy of approximation.

Below we present the comparisons between PVD and PTD applied to the full lung DVF data. Based on the table summary, PTD achieve a better result in terms of data compression and accuracy of approximation. The time consumption of PTD is higher than PVD. Followed by the table is the plot of the tail portion density of the absolute error. The red curve is PVD and the blue curve is PTD. As shown in the full lung DVF case, the PTD curve is below the PVD curve suggesting less voxels with large deviations.

	PVD on Full Lung	PTD on Full Lung
Free Parameters (%)	1882650 (0.42%)	742420(0.17%)
% of Variance Explained	98% & 98%	91-95% & 95%
0.15 cm - Max (‰)	186746 (0.42‰)	7873 (0.02‰)
0.20 cm - Max (‰)	39668 (0.09‰)	2623 (0.006‰)
Max (cm)	0.85	0.42
MSE	$3.93e10^{-4}$	$8.30e10^{-5}$
CPU time (min)	23.05	73.8

Table 7.15: Performance of PVD and PTD on Full Lung DVF

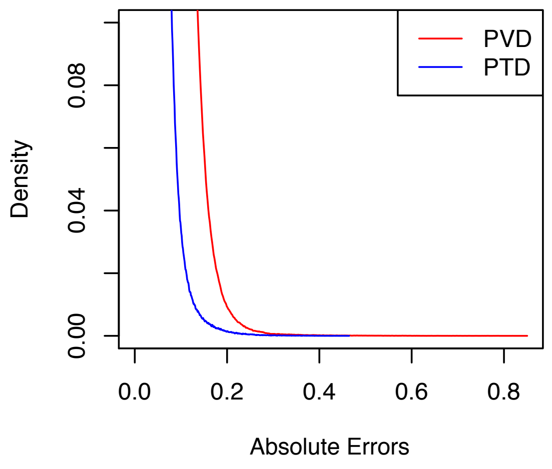


Figure 7.15: Tail Portion Density Plot of the Absolute Error for PVD and PTD on Full Lung DVF

Next we present the results of PTD algorithm on 9 sets of half lung DVF datasets. For set 1 below, the accuracy of approximation from PTD is drastically (but not uniformly) better than PVD. As shown in the table below for DVF set 1, the number of voxels with an absolute difference above 0.15 cm and 0.20 cm is reduced to 0 in the case of PTD. PVD has a better data compression for this case, but we believe PTD can achieve a better level of data compression in sacrifice of some accuracy of approximation. Followed by the table is the tail portion distribution of the absolute error.

	PVD on Half Lung	PTD on Half Lung
Free Parameters (%)	513846 (0.61%)	1017575 (1.2%)
% of Variance Explained	95% & 95%	>99% & 95%
0.15 cm - Max (‰)	287850 (3.40‰)	0 (0‰)
0.20 cm - Max (‰)	153843 (71.82‰)	0 (0‰)
Max (cm)	4.30	0.07
MSE	$0.97e10^{-3}$	$01.40e10^{-6}$
CPU time (sec)	102	654

Table 7.16: Performance of PVD and PTD on Half Lung DVF Set 1

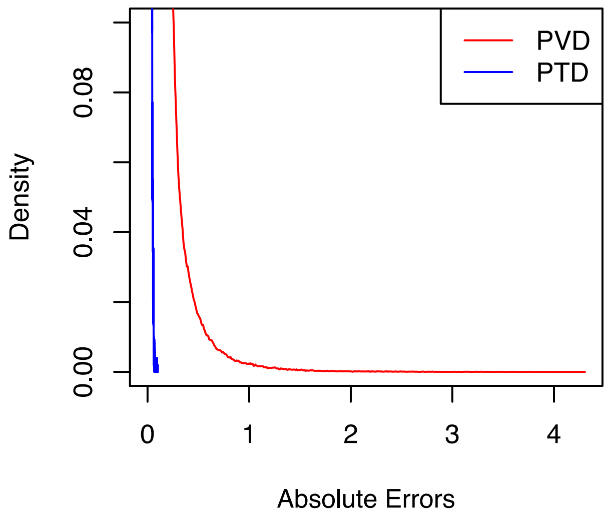


Figure 7.16: Tail Portion Density Plot of the Absolute Error for PVD and PTD on Clinical Half Lung DVF Set 1

	PVD on Half Lung	PTD on Half Lung
Free Parameters (%)	1795674 (1.62%)	1263507 (1.14%)
% of Variance Explained	99% & 99%	90-95% & 95%
0.15 cm - Max (‰)	4896 (0.04‰)	2848 (0.03‰)
0.20 cm - Max (‰)	2219 (0.02‰)	1506 (0.01‰)
Max (cm)	0.79	0.87
MSE	$0.10e10^{-3}$	$7.13e10^{-5}$
CPU time (sec)	153	2976

Table 7.17: Performance of PVD and PTD on Half Lung DVF Set 2

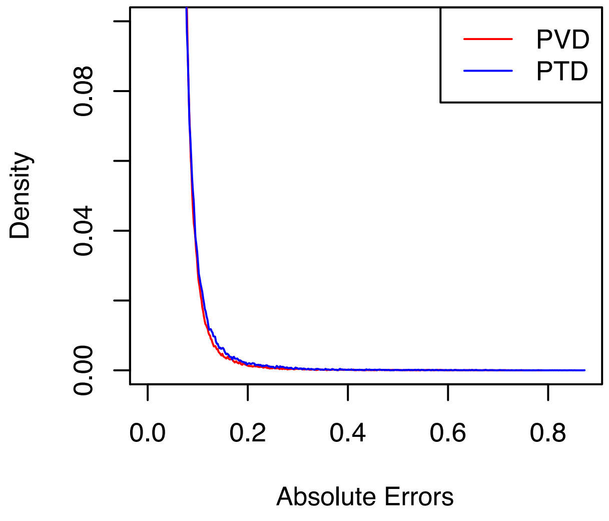


Figure 7.17: Tail Portion Density Plot of the Absolute Error for PVD and PTD on Clinical Half Lung DVF Set 2

	PVD on Half Lung	PTD on Half Lung
Free Parameters (%)	1091370 (1.21%)	1351714 (1.5%)
% of Variance Explained	98% & 98%	94-97% & 95%
0.15 cm - Max (‰)	13851 (0.15‰)	942 (0.01‰)
0.20 cm - Max (‰)	5331 (0.06‰)	250 (0.003‰)
Max (cm)	0.50	0.29
MSE	$0.20e10^{-3}$	$2.76e10^{-5}$
CPU time (sec)	109	1254

Table 7.18: Performance of PVD and PTD on Half Lung DVF Set 3

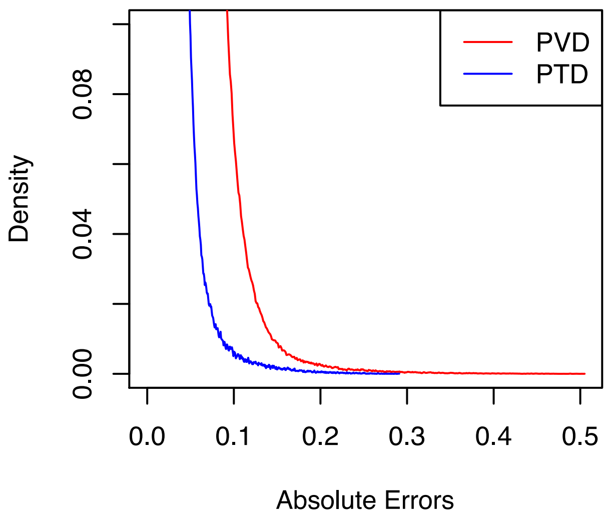


Figure 7.18: Tail Portion Density Plot of the Absolute Error for PVD and PTD on Clinical Half Lung DVF Set 3

	PVD on Half Lung	PTD on Half Lung
Free Parameters (%)	6656372 (6.64%)	1477784 (1.5%)
% of Variance Explained	95% & 95%	40-73% & 95%
0.15 cm - Max (‰)	110782 (1.11‰)	128470 (1.28‰)
0.20 cm - Max (‰)	38808 (0.39‰)	50106 (0.50‰)
Max (cm)	0.98	1.49
MSE	$0.40e10^{-3}$	$0.83e10^{-3}$
CPU time (sec)	202	3960

Table 7.19: Performance of PVD and PTD on Half Lung DVF Set 4

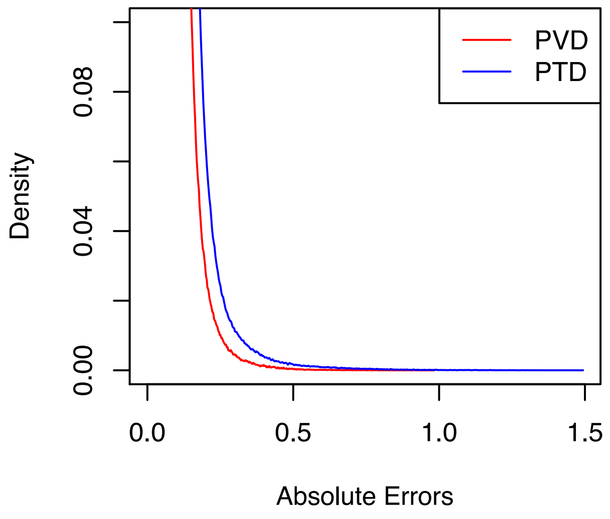


Figure 7.19: Tail Portion Density Plot of the Absolute Error for PVD and PTD on Clinical Half Lung DVF Set 4

	PVD on Half Lung	PTD on Half Lung
Free Parameters (%)	1944954 (0.65%)	771042 (0.29%)
% of Variance Explained	97% & 97%	89-94% & 95%
0.15 cm - Max (‰)	146462 (0.49‰)	6811 (0.02‰)
0.20 cm - Max (‰)	38947 (0.13‰)	2178 (0.007‰)
Max (cm)	0.67	0.41
MSE	$0.31e10^{-3}$	$7.27e10^{-5}$
CPU time (sec)	820	5040

Table 7.20: Performance of PVD on Half Lung DVF Set 5

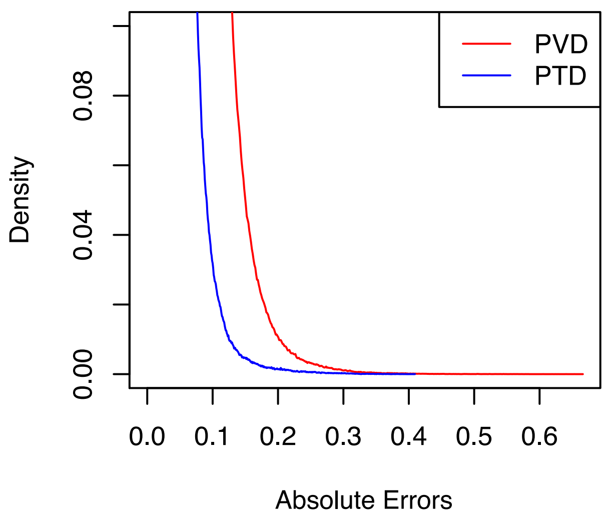


Figure 7.20: Tail Portion Density Plot of the Absolute Error for PVD and PTD on Clinical Half Lung DVF Set 5

	PVD on Half Lung	PTD on Half Lung
Free Parameters (%)	1442260 (0.82%)	1201134 (0.68%)
% of Variance Explained	98% & 98%	92-97% & 95%
0.15 cm - Max (‰)	15586 (0.09‰)	62 (0.0003‰)
0.20 cm - Max (‰)	4098 (0.02‰)	0 (0‰)
Max (cm)	0.40	0.17
MSE	$0.16e10^{-3}$	$2.57e10^{-5}$
CPU time (sec)	369	2820

Table 7.21: Performance of PVD and PTD on Half Lung DVF Set 6

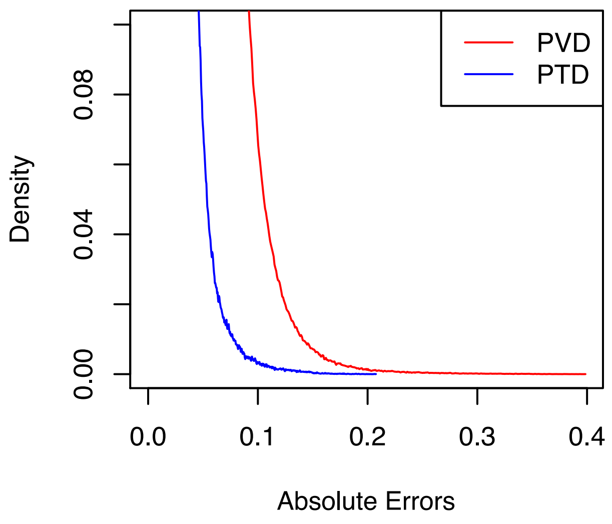


Figure 7.21: Tail Portion Density Plot of the Absolute Error for PVD and PTD on Clinical Half Lung DVF Set 6

	PVD on Half Lung	PTD on Half Lung
Free Parameters (%)	720910 (0.37%)	1228812 (0.63%)
% of Variance Explained	97% & 97%	93-97% & 95%
0.15 cm - Max (‰)	169894 (0.87‰)	484 (0.002‰)
0.20 cm - Max (‰)	38697 (0.20‰)	58 (0.0003‰)
Max (cm)	0.49	0.24
MSE	$0.43e10^{-3}$	$3.39e10^{-5}$
CPU time (sec)	385	3600

Table 7.22: Performance of PVD and PTD on Half Lung DVF Set 7

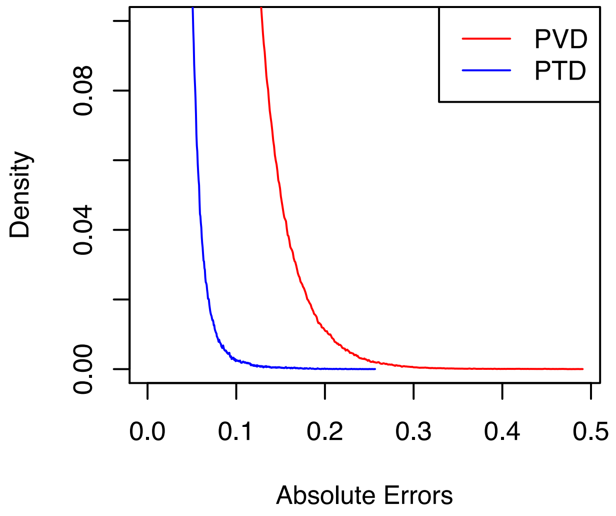


Figure 7.22: Tail Portion Density Plot of the Absolute Error for PVD and PTD on Clinical Half Lung DVF Set 7

	PVD on Half Lung	PTD on Half Lung
Free Parameters (%)	784506 (0.72%)	1380806 (1.26%)
% of Variance Explained	97% & 97%	94-96% & 95%
0.15 cm - Max (‰)	16921 (0.15‰)	44 (0.0004‰)
0.20 cm - Max (‰)	3277 (0.03‰)	0 (0‰)
Max (cm)	0.37	0.16
MSE	$0.27e10^{-3}$	$3.20e10^{-5}$
CPU time (sec)	236	1644

Table 7.23: Performance of PVD and PTD on Half Lung DVF Set 8

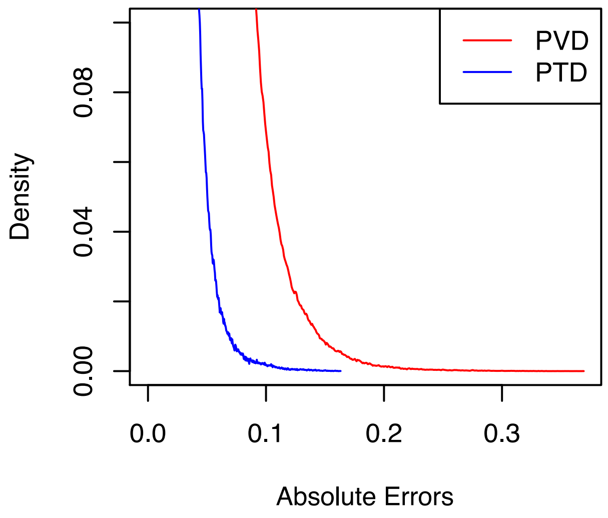


Figure 7.23: Tail Portion Density Plot of the Absolute Error for PVD and PTD on Clinical Half Lung DVF Set 8

	PVD on Half Lung	PTD on Half Lung
Free Parameters (%)	622216 (0.43%)	1329818 (0.91%)
% of Variance Explained	96% & 96%	94-98% & 95%
0.15 cm - Max (‰)	130428 (0.89‰)	0 (0‰)
0.20 cm - Max (‰)	30781 (0.21‰)	0 (0‰)
Max (cm)	0.42	0.07
MSE	$0.40e10^{-3}$	$2.08e10^{-5}$
CPU time (sec)	264	2406

Table 7.24: Performance of PVD on Half Lung DVF Set 9

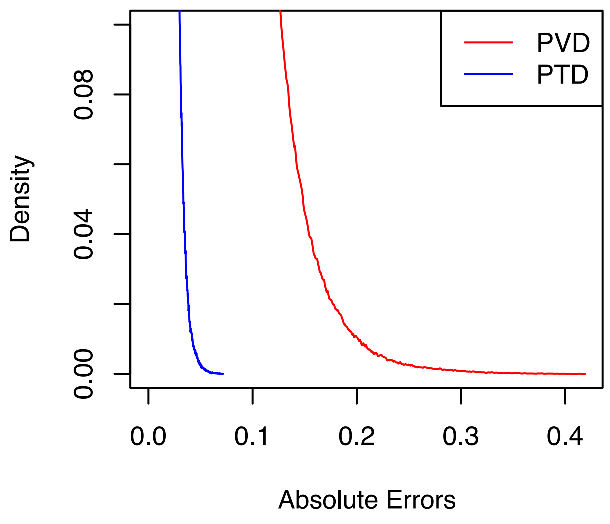


Figure 7.24: Tail Portion Density Plot of the Absolute Error for PVD and PTD on Clinical Half Lung DVF Set 9

Now, we present the table comparisons between PCA and PVD on simulated DVF tensor followed by tail portion distribution on absolute error on a voxel level.

Simulated DVF Set 1	PTD	PVD
Free Parameters (%)	6030504 (1.53%)	875488 (0.22%)
% of Variance Explained	100% & 95%	99.9% & 99.9%
0.15 cm - Max (‰)	29368597 (74.7‰)	29892842 (76.0‰)
0.20 cm - Max (‰)	12482683 (31.7‰)	12615631 (32.1‰)
Max (cm)	0.50	0.50
MSE	$7.55e10^{-3}$	$7.60e10^{-3}$
CPU time (sec)	72720	1270

Table 7.25: Performance of PVD and PTD on Simulated DVF Set 1

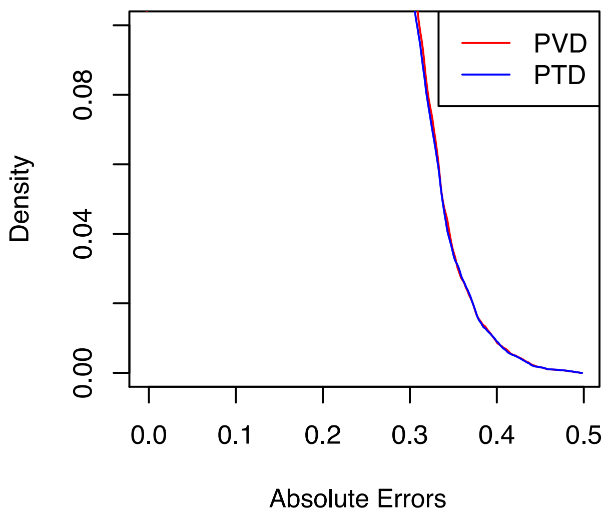


Figure 7.25: Tail Portion Density Plot of the Absolute Error for PVD and PTD on Simulated Lung DVF Set 1

Simulated DVF Set 2	PTD	PVD
Free Parameters (%)	4307676 (1.10%)	935744 (0.24%)
% of Variance Explained	100% & 95%	99.9% & 99.9%
0.15 cm - Max (‰)	5000 (0.013‰)	5210 (0.013‰)
0.20 cm - Max (‰)	0 (0‰)	0 (0‰)
Max (cm)	0.16	0.16
MSE	$1.60e10^{-3}$	$1.61e10^{-3}$
CPU time (sec)	71352	1375

Table 7.26: Performance of PVD and PTD on Simulated DVF Set 2

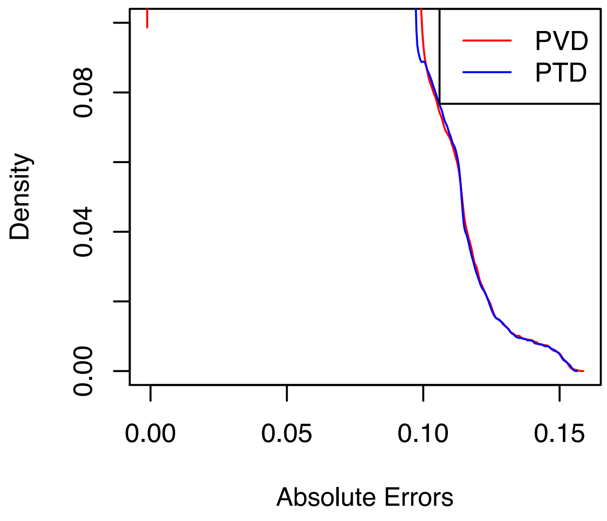


Figure 7.26: Tail Portion Density Plot of the Absolute Error for PVD and PTDD on Simulated Lung DVF Set 2

Simulated DVF Set 3	PTD	PVD
Free Parameters (%)	5965580 (1.52%)	920208 (0.23%)
% of Variance Explained	100% & 95%	99.9% & 99.9%
0.15 cm - Max (‰)	20816848 (52.9‰)	21221841 (54.0‰)
0.20 cm - Max (‰)	6913448 (17.6‰)	7024940 (17.9‰)
Max (cm)	0.47	0.47
MSE	$6.13e10^{-3}$	$6.19e10^{-3}$
CPU time (sec)	74052	798

Table 7.27: Performance of PVD and PTD on Simulated DVF Set 3

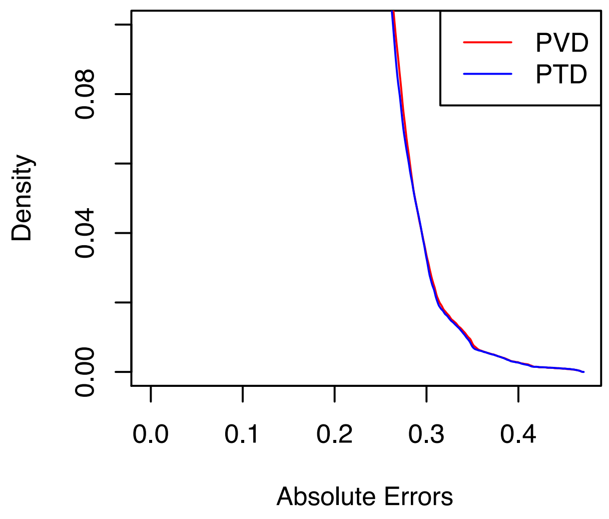


Figure 7.27: Tail Portion Density Plot of the Absolute Error for PVD and PTD on Simulated Lung DVF Set 3

Simulated DVF Set 4	PTD	PVD
Free Parameters (%)	5430568 (1.38%)	920208 (0.23%)
% of Variance Explained	100% & 95%	99.9% & 99.9%
0.15 cm - Max (‰)	42572 (0.11‰)	43065 (0.11‰)
0.20 cm - Max (‰)	0 (0‰)	0 (0‰)
Max (cm)	0.19	0.18
MSE	$1.23e10^{-3}$	$1.23e10^{-3}$
CPU time (sec)	51912	1015

Table 7.28: Performance of PVD and PTD on Simulated DVF Set 4

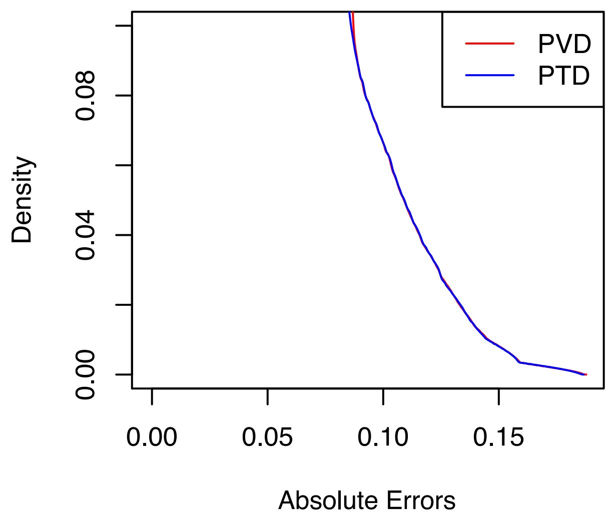


Figure 7.28: Tail Portion Density Plot of the Absolute Error for PVD and PTD on Simulated Lung DVF Set 4

One thing that seems to be common across all results of clinical half lung DVF data with one exception (set 4) is that PTD is able to achieve better accuracy of approximation as measured by the number of voxels with an absolute difference above 0.15 cm and 0.20 cm and maximum absolute difference. The one exception is in clinical half lung DVF set 4, where PVD has a slight better accuracy of approximation. However, looking carefully at the data compression level for this set, PVD's compressed data is more than 4 times larger than PTD. The performance for clinical DVF set 2 is very similar. We are able to conclude that PTD shows some if not uniform advantage over PVD, which results from retaining the entire DVF structure and exploits features in the first three modes in the DVF tensor data at step one data reduction (refer to Chapter 5). The time consumption is drastically higher than PVD, because PTD utilizes an iterative (HOOI) algorithm to find the best low multi-linear rank approximation in step one data reduction. This problem in tensor decomposition is non-deterministic polynomial-time (NP) hard and takes time. The trade-off between accuracy of approximation and time consumption can be a subjective call in our opinion because PVD's performance is already very satisfying. For the simulated DVF, the performance between PVD and PTD are nearly indistinguishable. But the PTD algorithm does take longer to complete due to its iterative nature. We believe the results on clinical DVF tensors are more realistic.

The PTD algorithm is able to retain the DVF tensor structure. PTD works different from PVD and PCA in the sense that both PCA and PVD let users set the percentages of variance explained threshold first, and the algorithms will find the best low rank approximation that achieves the set threshold. Whereas, PTD requires us to set the size of the core tensor, and the algorithm is able to find the best low multilinear rank approximation using a core tensor of the given size. Then the percentage of norms explained can be calculated later (refer to Section 6.4). We are not able to provide a guideline for readers to follow in terms of finding the optimal core tensor size in analyzing DVF. But there are some literature regarding this particular topic that readers can refer to [Chen et al., 2013, Chen et al., 2015].

There is no particular reason why we chose to stack the three coordinates of the displacement vector along mode 1 of DVF. Technically one can stack the three coordinates along any one of the first three modes of DVF. The only difference is that the variations across the coordinates are present in the subspace spanned by mode-1 fibers if coordinates are stacked along mode 1, in subspace spanned by mode-2 fibers if coordinates are stacked along mode 2, and in subspace spanned by mode-3 fibers if coordinates are stacked along mode 3. When performing the Tucker decomposition on the third-order tensor obtained by stacking the coordinates, we packed all the spatial information in the first step of PTD, and Tucker decomposition finds the best low multi-linear rank approximation though HOOI for each third-order tensor with respect to Frobenius norm. The factor matrices for each mode are multi-linearly related through the core tensor. The population factor matrices obtained through combing phase-specific factor matrices computed by Tucker decomposition in the first step only contains temporal information. Therefore, the SVD for each of the three population factor matrice in the second step of PTD finds large components for the row and column subspaces of the corresponding mode along the phase dimension. The actual clinical interpretation still requires a more thorough study and research.

In the end we will discuss the time consumption of PTD, the HOOI algorithm is iterative and can be time consuming. But in the first data reduction step of PTD, the 10 tensors, one at each phase, can be performed on distributed system using parallel computing to speed up the algorithm. We believe that this advantage of PTD is especially helpful as the dimensionality of the tensor increases. Because PCA based algorithm requires vectorization of the entire tensor and performs only one step of data reduction and will not be able to take advantage of parallel computing and might have problems in coupling with bigger tensors (which we are not able to test). As for PVD, since there are already efficient algorithm for SVD for tall and skinny matrices, using distributing systems with parallel computing at step 1 data reduction will most likely results in wasting too much on distributing jobs to different systems and communicating between different systems, all of which remain to be tested. Unfortunately, we are not able to code and test a parallel version of PTD algorithm, which will be left as future work and some articles on this topic are emerging

[Sidiropoulos et al., 2014, Sidiropoulos et al., 2014, Austin et al., 2016].

Bibliography

[Abdi, 2007a] Abdi, H. (2007a). The eigen-decomposition: Eigenvalues and eigenvectors. Encyclopedia of measurement and statistics, pages 304–308.

[Abdi, 2007b] Abdi, H. (2007b). Singular value decomposition (svd) and generalized singular value decomposition. Encyclopedia of measurement and statistics. Thousand Oaks (CA): Sage, pages 907–12.

[Afra et al., 2014] Afra, S., Gildin, E., and Tarrahi, M. (2014). Heterogeneous reservoir characterization using efficient parameterization through higher order svd (hosvd). In American Control Conference (ACC), 2014, pages 147–152. IEEE.

[Austin et al., 2016] Austin, W., Ballard, G., and Kolda, T. G. (2016). Parallel tensor compression for large-scale scientific data. In Parallel and Distributed Processing Symposium, 2016 IEEE International, pages 912–922. IEEE.

[Bellman, 2015] Bellman, R. E. (2015). Adaptive control processes: a guided tour. Princeton university press.

[Benson et al., 2014] Benson, A. R., Lee, J. D., Rajwa, B., and Gleich, D. F. (2014). Scalable methods for nonnegative matrix factorizations of near-separable tall-and-skinny matrices. In Advances in Neural Information Processing Systems, pages 945–953.

[Bergqvist and Larsson, 2010] Bergqvist, G. and Larsson, E. G. (2010). The higher-order singular value decomposition: Theory and an application [lecture notes]. IEEE Signal Processing Magazine, 27(3):151–154.

[Cardoso, 1990] Cardoso, J.-F. (1990). Eigen-structure of the fourth-order cumulant tensor with application to the blind source separation problem. In Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on, pages 2655–2658. IEEE.

[Cardoso, 1991] Cardoso, J.-F. (1991). Super-symmetric decomposition of the fourth-order cumulant tensor. blind identification of more sources than sensors. In Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on, pages 3109–3112. IEEE.

[Carroll and Chang, 1970] Carroll, J. D. and Chang, J.-J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of ?eckart-young? decomposition. Psychometrika, 35(3):283–319.

[Chatterjee et al., 1997] Chatterjee, C., Roychowdhury, V. P., Ramos, J., and Zoltowski, M. D. (1997). Self-organizing algorithms for generalized eigen-decomposition. IEEE Transactions on Neural Networks, 8(6):1518–1530.

[Chen et al., 2013] Chen, B., Li, Z., and Zhang, S. (2013). On tensor tucker decomposition: the case for an adjustable core size. Technical Report, University of Minnesota.

[Chen et al., 2015] Chen, B., Li, Z., and Zhang, S. (2015). On optimal low rank tucker approximation for tensors: the case for an adjustable core size. Journal of Global Optimization, 62(4):811–832.

[Chen et al., 2014] Chen, Z.-Y., Wang, Y.-X., Lin, Y., Zhang, J.-S., Yang, F., Zhou, Q.-L., and Liao, Y.-Y. (2014). Advance of molecular imaging technology and targeted imaging agent in imaging and therapy. BioMed research international, 2014.

[Chung and Kim, 2015] Chung, Y. E. and Kim, K. W. (2015). Contrast-enhanced ultrasonography: advance and current status in abdominal imaging. Ultrasonography, 34(1):3.

[Cichocki et al., 2016] Cichocki, A., Lee, N., Oseledets, I., Phan, A.-H., Zhao, Q., Mandic, D. P., et al. (2016). Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. Foundations and Trends® in Machine Learning, 9(4-5):249–429.

[Cichocki et al., 2015] Cichocki, A., Mandic, D., De Lathauwer, L., Zhou, G., Zhao, Q., Caiafa, C., and Phan, H. A. (2015). Tensor decompositions for signal processing applications: From two-way to multiway component analysis. IEEE Signal Processing Magazine, 32(2):145–163.

[Cohen et al., 2015] Cohen, J., Farias, R. C., and Comon, P. (2015). Fast decomposition of large nonnegative tensors. IEEE Signal Processing Letters, 22(7):862–866.

[Collignon et al., 1995] Collignon, A., Maes, F., Delaere, D., Vandermeulen, D., Suetens, P., and Marchal, G. (1995). Automated multi-modality image registration based on information theory. In Information processing in medical imaging, volume 3, pages 263–274.

[Comon, 2014] Comon, P. (2014). Tensors: a brief introduction. IEEE Signal Processing Magazine, 31(3):44–53.

[Crainiceanu et al., 2011] Crainiceanu, C. M., Caffo, B. S., Luo, S., Zipunnikov, V. M., and Punjabi, N. M. (2011). Population value decomposition, a framework for the analysis of image populations. Journal of the American Statistical Association, 106(495):775–790.

[De Lathauwer et al., 2000a] De Lathauwer, L., De Moor, B., and Vandewalle, J. (2000a). A multilinear singular value decomposition. SIAM journal on Matrix Analysis and Applications, 21(4):1253–1278.

[De Lathauwer et al., 2000b] De Lathauwer, L., De Moor, B., and Vandewalle, J. (2000b). On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors. SIAM journal on Matrix Analysis and Applications, 21(4):1324–1342.

[De Lathauwer et al., 1994] De Lathauwer, L., De Moor, B., Vandewalle, J., and by Higher-Order, B. S. S. (1994). Singular value decomposition. In Proc. EUSIPCO-94, Edinburgh, Scotland, UK, volume 1, pages 175–178.

[Doi, 2007] Doi, K. (2007). Computer-aided diagnosis in medical imaging: historical review, current status and future potential. Computerized medical imaging and graphics, 31(4):198–211.

[Dolgov and Khoromskij, 2015] Dolgov, S. and Khoromskij, B. (2015). Simultaneous state-time approximation of the chemical master equation using tensor product formats. Numerical Linear Algebra with Applications, 22(2):197–219.

[Dolgov et al., 2014] Dolgov, S. V., Khoromskij, B. N., Oseledets, I. V., and Savostyanov, D. V. (2014). Computation of extreme eigenvalues in higher dimensions using block tensor train format. Computer Physics Communications, 185(4):1207–1216.

[Dzingwa and Hayes, 2017] Dzingwa, A. and Hayes, E. (2017). Ct mr image registration for 3d image based treatment planning in cervical cancer brachytherapy–an initial experience. Physica Medica: European Journal of Medical Physics, 42:355.

[Eckart and Young, 1936] Eckart, C. and Young, G. (1936). The approximation of one matrix by another of lower rank. Psychometrika, 1(3):211–218.

[Faverge et al., 2017] Faverge, M., Langou, J., Robert, Y., and Dongarra, J. (2017). Bidiagonalization and r-bidiagonalization: Parallel tiled algorithms, critical paths and distributed-memory implementation. In IPDPS'17-31st IEEE International Parallel and Distributed Processing Symposium.

[Fukuhara et al., 2013] Fukuhara, T., Kantian, A., Endres, M., Cheneau, M., Schauß, P., Hild, S., Bellem, D., Schollwöck, U., Giamarchi, T., Gross, C., et al. (2013). Quantum dynamics of a mobile spin impurity. Nature Physics, 9(4):235–241.

[Golub and Reinsch, 1970] Golub, G. H. and Reinsch, C. (1970). Singular value decomposition and least squares solutions. Numerische mathematik, 14(5):403–420.

[Golub and Van Loan, 2012] Golub, G. H. and Van Loan, C. F. (2012). Matrix computations, volume 3. JHU Press.

[Grasedyck, 2010] Grasedyck, L. (2010). Hierarchical singular value decomposition of tensors. SIAM Journal on Matrix Analysis and Applications, 31(4):2029–2054.

[Grasedyck et al., 2013] Grasedyck, L., Kressner, D., and Tobler, C. (2013). A literature survey of low-rank tensor approximation techniques. GAMM-Mitteilungen, 36(1):53–78.

[Haardt et al., 2008] Haardt, M., Roemer, F., and Del Galdo, G. (2008). Higher-order svd-based subspace estimation to improve the parameter estimation accuracy in multidimensional harmonic retrieval problems. IEEE Transactions on Signal Processing, 56(7):3198–3213.

[Hanley et al., 1999] Hanley, J., Debois, M. M., Mah, D., Mageras, G. S., Raben, A., Rosenzweig, K., Mychalczak, B., Schwartz, L. H., Gloeggler, P. J., Lutz, W., et al. (1999). Deep inspiration breath-hold technique for lung tumors: the potential value of target immobilization and reduced lung density in dose escalation. International Journal of Radiation Oncology* Biology* Physics, 45(3):603–611.

[Harshman, 1970] Harshman, R. A. (1970). Foundations of the parafac procedure: models and conditions for an" explanatory" multimodal factor analysis.

[Hitchcock, 1928] Hitchcock, F. L. (1928). Multiple invariants and generalized rank of a p-way matrix or tensor. Studies in Applied Mathematics, 7(1-4):39–79.

[Hoff et al., 2016] Hoff, P. D. et al. (2016). Equivariant and scale-free tucker decomposition models. Bayesian Analysis, 11(3):627–648.

[Hoge and Westin, 2005] Hoge, W. S. and Westin, C.-F. (2005). Identification of translational displacements between n-dimensional data sets using the high-order svd and phase correlation. IEEE Transactions on Image Processing, 14(7):884–889.

[Huang et al., 2008] Huang, H., Ding, C., Luo, D., and Li, T. (2008). Simultaneous tensor subspace selection and clustering: the equivalence of high order svd and k-means clustering. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge Discovery and Data mining, pages 327–335. ACM.

[Huang et al., 2016] Huang, K., Sidiropoulos, N. D., and Liavas, A. P. (2016). A flexible and efficient algorithmic framework for constrained matrix and tensor factorization. IEEE Transactions on Signal Processing, 64(19):5052–5065.

[Jemal et al., 2008] Jemal, A., Siegel, R., Ward, E., Hao, Y., Xu, J., Murray, T., and Thun, M. J. (2008). Cancer statistics, 2008. CA: a cancer journal for clinicians, 58(2):71–96.

[Kazeev et al., 2013] Kazeev, V. A., Khoromskij, B. N., and Tyrtyshnikov, E. E. (2013). Multilevel toeplitz matrices generated by tensor-structured vectors and convolution with logarithmic complexity. SIAM Journal on Scientific Computing, 35(3):A1511–A1536.

[Keall et al., 2006] Keall, P. J., Mageras, G. S., Balter, J. M., Emery, R. S., Forster, K. M., Jiang, S. B., Kapatoes, J. M., Low, D. A., Murphy, M. J., Murray, B. R., et al. (2006). The management of respiratory motion in radiation oncology report of aapm task group 76. Medical physics, 33(10):3874–3900.

[Khoromskij and Khoromskaia, 2007] Khoromskij, B. and Khoromskaia, V. (2007). Low rank tucker-type tensor approximation to classical potentials. Open Mathematics, 5(3):523–550.

[Kolda and Bader, 2009] Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. SIAM review, 51(3):455–500.

[Kolda and Sun, 2008] Kolda, T. G. and Sun, J. (2008). Scalable tensor decompositions for multi-aspect data mining. In Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on, pages 363–372. IEEE.

[Kressner et al., 2014] Kressner, D., Steinlechner, M., and Vandereycken, B. (2014). Low-rank tensor completion by riemannian optimization. BIT Numerical Mathematics, 54(2):447–468.

[Lebedev et al., 2014] Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., and Lempitsky, V. (2014). Speeding-up convolutional neural networks using fine-tuned cp-decomposition. arXiv preprint arXiv:1412.6553.

[Li et al., 2011] Li, R., Lewis, J. H., Jia, X., Zhao, T., Liu, W., Wuenschel, S., Lamb, J., Yang, D., Low, D. A., and Jiang, S. B. (2011). On a pca-based lung motion model. Physics in medicine and biology, 56(18):6009.

[Li et al., 2017] Li, X., Ng, M. K., Cong, G., Ye, Y., and Wu, Q. (2017). Mr-ntd: manifold regularization nonnegative tucker decomposition for tensor data dimension reduction and representation. IEEE transactions on neural networks and learning systems, 28(8):1787–1800.

[Liao et al., 2015] Liao, S., Hu, Y., Zhu, X., and Li, S. Z. (2015). Person re-identification by local maximal occurrence representation and metric learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2197–2206.

[Lindner, 2004] Lindner, J. R. (2004). Microbubbles in medical imaging: current applications and future directions. Nature Reviews Drug Discovery, 3(6):527–533.

[Liu et al., 2014] Liu, Y., Shang, F., Fan, W., Cheng, J., and Cheng, H. (2014). Generalized higher-order orthogonal iteration for tensor decomposition and completion. In Advances in Neural Information Processing Systems, pages 1763–1771.

[Lucas et al., 1981] Lucas, B. D., Kanade, T., et al. (1981). An iterative image registration technique with an application to stereo vision.

[Maes et al., 1997] Maes, F., Collignon, A., Vandermeulen, D., Marchal, G., and Suetens, P. (1997). Multimodality image registration by maximization of mutual information. IEEE transactions on medical imaging, 16(2):187–198.

[Maintz and Viergever, 1998] Maintz, J. A. and Viergever, M. A. (1998). A survey of medical image registration. Medical image analysis, 2(1):1–36.

[Marro et al., 2016] Marro, A., Bandukwala, T., and Mak, W. (2016). Three-dimensional printing and medical imaging: a review of the methods and applications. Current problems in diagnostic radiology, 45(1):2–9.

[Mattes et al., 2003] Mattes, D., Haynor, D. R., Vesselle, H., Lewellen, T. K., and Eubank, W. (2003). Pet-ct image registration in the chest using free-form deformations. IEEE transactions on medical imaging, 22(1):120–128.

[McClelland et al., 2013] McClelland, J. R., Hawkes, D. J., Schaeffter, T., and King, A. P. (2013). Respiratory motion models: a review. Medical image analysis, 17(1):19–42.

[Oh et al., 2017] Oh, J., Shin, K., Papalexakis, E. E., Faloutsos, C., and Yu, H. (2017). S-hot: Scalable high-order tucker decomposition. In Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, pages 761–770. ACM.

[Oliveira and Tavares, 2014] Oliveira, F. P. and Tavares, J. M. R. (2014). Medical image registration: a review. Computer methods in biomechanics and biomedical engineering, 17(2):73–93.

[Orús, 2014] Orús, R. (2014). A practical introduction to tensor networks: Matrix product states and projected entangled pair states. Annals of Physics, 349:117–158.

[Oseledets and Tyrtyshnikov, 2009] Oseledets, I. V. and Tyrtyshnikov, E. E. (2009). Breaking the curse of dimensionality, or how to use svd in many dimensions. SIAM Journal on Scientific Computing, 31(5):3744–3759.

[Picano et al., 2014] Picano, E., Vañó, E., Rehani, M. M., Cuocolo, A., Mont, L., Bodi, V., Bar, O., Maccia, C., Pierard, L., Sicari, R., et al. (2014). The appropriate and justified use of medical radiation in cardiovascular imaging: a position document of the esc associations of cardiovascular imaging, percutaneous cardiovascular interventions and electrophysiology. European heart journal, 35(10):665–672.

[Pietrzyk et al., 1994] Pietrzyk, U., Herholz, K., Fink, G., Jacobs, A., Mielke, R., Slansky, I., Würker, M., and Heiss, W.-D. (1994). An interactive technique for three-dimensional image registration: validation for pet, spect, mri and ct brain studies. Journal of nuclear medicine, 35(12):2011–2018.

[Rauhut et al., 2017] Rauhut, H., Schneider, R., and Stojanac, Ž. (2017). Low rank tensor recovery via iterative hard thresholding. Linear Algebra and its Applications, 523:220–262.

[Ries et al., 2006] Ries, L. A., Harkins, D., Krapcho, M., Mariotto, A., Miller, B. A., Feuer, E. J., Clegg, L. X., Eisner, M., Horner, M.-J., Howlader, N., et al. (2006). Seer cancer statistics review, 1975-2003.

[Rosenkrantz et al., 2016] Rosenkrantz, A. B., Verma, S., Choyke, P., Eberhardt, S. C., Eggener, S. E., Gaitonde, K., Haider, M. A., Margolis, D. J., Marks, L. S., Pinto, P., et al. (2016). Prostate magnetic resonance imaging and magnetic resonance imaging targeted biopsy in patients with a prior negative biopsy: a consensus statement by aua and sar. The Journal of urology, 196(6):1613–1618.

[Rosenzweig et al., 2000] Rosenzweig, K. E., Hanley, J., Mah, D., Mageras, G., Hunt, M., Toner, S., Burman, C., Ling, C., Mychalczak, B., Fuks, Z., et al. (2000). The deep inspiration breath-hold technique in the treatment of inoperable non–small-cell lung cancer. International Journal of Radiation Oncology* Biology* Physics, 48(1):81–87.

[Savas and Eldén, 2007] Savas, B. and Eldén, L. (2007). Handwritten digit classification using higher order singular value decomposition. Pattern recognition, 40(3):993–1003.

[Sears et al., 2009] Sears, M. P., Bader, B. W., and Kolda, T. (2009). Parallel implementation of tensor decompositions for large data analysis. SIAM AN09 July, 8.

[Sheehan and Saad, 2007] Sheehan, B. N. and Saad, Y. (2007). Higher order orthogonal iteration of tensors (hooi) and its relation to pca and glram. In Proceedings of the 2007 SIAM International Conference on Data Mining, pages 355–365. SIAM.

[Sidiropoulos et al., 2014] Sidiropoulos, N. D., Papalexakis, E. E., and Faloutsos, C. (2014). Parallel randomly compressed cubes: A scalable distributed architecture for big tensor decomposition. IEEE Signal Processing Magazine, 31(5):57–70.

[Studholme et al., 1996] Studholme, C., Hill, D. L., and Hawkes, D. J. (1996). Automated 3-d registration of mr and ct images of the head. Medical image analysis, 1(2):163–175.

[Szalay et al., 2015] Szalay, S., Pfeffer, M., Murg, V., Barcza, G., Verstraete, F., Schneider, R., and Legeza, Ö. (2015). Tensor product methods and entanglement optimization for ab initio quantum chemistry. International Journal of Quantum Chemistry, 115(19):1342–1391.

[Trefethen and Bau III, 1997] Trefethen, L. N. and Bau III, D. (1997). Numerical linear algebra, volume 50. Siam.

[Vasilescu and Terzopoulos, 2003] Vasilescu, M. A. O. and Terzopoulos, D. (2003). Multilinear subspace analysis of image ensembles. In Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on, volume 2, pages II–93. IEEE.

[Verstraete et al., 2008] Verstraete, F., Murg, V., and Cirac, J. I. (2008). Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems. Advances in Physics, 57(2):143–224.

[Vervliet et al., 2014] Vervliet, N., Debals, O., Sorber, L., and De Lathauwer, L. (2014). Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis. IEEE Signal Processing Magazine, 31(5):71–79.

[Wu et al., 2017] Wu, J., Wang, Z., Wu, Y., Liu, L., Deng, S., and Huang, H. (2017). A tensor cp decomposition method for clustering heterogeneous information networks via stochastic gradient descent algorithms. Scientific Programming, 2017.

[Yan et al., 2015] Yan, H., Paynabar, K., and Shi, J. (2015). Image-based process monitoring using low-rank tensor decomposition. IEEE Transactions on Automation Science and Engineering, 12(1):216–227.

[Yang et al., 2006] Yang, J., Xi, H., Yang, F., and Zhao, Y. (2006). Rls-based adaptive algorithms for generalized eigen-decomposition. IEEE Transactions on Signal Processing, 54(4):1177–1188.

[Ye, 2005] Ye, J. (2005). Generalized low rank approximations of matrices. Machine Learning, 61(1-3):167–191.

[Zhou et al., 2017] Zhou, Z., Fang, J., Yang, L., Li, H., Chen, Z., and Blum, R. (2017). Low-rank tensor decomposition-aided channel estimation for millimeter wave mimo-ofdm systems. IEEE Journal on Selected Areas in Communications.

[Zitova and Flusser, 2003] Zitova, B. and Flusser, J. (2003). Image registration methods: a survey. Image and vision computing, 21(11):977–1000.

[Zou et al., 2016] Zou, L., Wang, Z. J., Chen, X., and Ji, X. (2016). Underdetermined joint blind source separation based on tensor decomposition. In Electrical and Computer Engineering (CCECE), 2016 IEEE Canadian Conference on, pages 1–4. IEEE.

Appendix A

.1 Revised PVD Algorithm

In this Appendix, we include all the algorithms for performing PVD and PTD. The purpose of each algorithm, input class, and output are included and explained at the beginning of each algorithm. We also include additional algorithms for plotting slices to identify voxels with large deviations and examples of running the PVD, PTD, and plotting images.

To read in .mat files, one can use the 'R.atlab' package in R. The PVD algorithm does not require any dependent packages besides some R base functions including 'svd'. The PTD algorithm requires the 'rTensor' package in R.

Listing .1: Revised PVD in R

```
#The following algorithm performs the Population Value Decomposition
    generalized to a fifth-order tensor by first unfolding the fifth mode
    along mode 1 and performing SVD by fixing mode 3 and 4. Then algorithm
    then combines all the leading left and right singular vectors across mode
    3 and 4 into two large U and V matrices and performs SVD on each of the
    two large U and V matrices.

#input needed:
```

```
#rs: a 3-level ([[sub]][[phase]][[dim3(transverse slice)]]) nested list of 2D
    arrays [dim1*3(DVF contains x,y,z cor),dim2]
#p1: the percentage (between 0 and 1) of variance explained in step-one PVD
    data reduction
#p2: the percentage (between 0 and 1) of variance explained in step-two PVD
    data reduction

#output
#an approximated fifth-order tensor

pvd<-function(rs,p1,p2){

source('/home/kangk4/dissertation/code/source/pvdstep1.R')
source('/home/kangk4/dissertation/code/source/combineuv.R')
source('/home/kangk4/dissertation/code/source/pvdstep2.R')
source('/home/kangk4/dissertation/code/source/recon.R')
if(p1<0 || p2<0 ||p1>1 ||p2>1){stop("p1␣and␣p2␣must␣be␣from␣[0,1]!")}
if (typeof(rs)!="list") {stop("The␣function␣pvd␣only␣takes␣a␣list␣as␣an␣input.
    ")}
if (typeof(rs)=="list")
{
        print("input␣class␣check␣passed.")
        if (!is.null(dim(rs))) {stop("The␣first/subject␣level␣in␣input␣data␣
            should␣have␣length␣at␣least␣1.")}
        if (is.null(dim(rs)) && length(rs)>0)
        {
                print("The␣first/subject␣level␣check␣passed.")
                if (!is.null(dim(rs[[1]])))
                {stop("The␣second/phase␣level␣in␣input␣data␣should␣have␣length
                    ␣at␣least␣1.")}
                if (is.null(dim(rs[[1]])) && length(rs[[1]])>0)
```

```
                    {
                              print("The␣second/phase␣nested␣level␣check␣passed.")
                              if(!is.null(dim(rs[[1]][[1]])))
                              {stop("The␣third/transevers␣slice␣level␣in␣input␣data␣
                                  should␣have␣length␣at␣least␣1.")}
                              if  (is.null(dim(rs[[1]][[1]])) && length(rs[[1]][[1]])
                                  >0)
                              {print("The␣third/transevers␣slice␣nested␣level␣check␣
                                  passed.")}
                    }
          }
}
print("Step␣1:␣Performing␣subject-level␣SVD...")
svdrs<-pvdstep1(rs,p1)
print("PVD␣step␣1␣succeeded.")
print("Stacking␣matrices␣for␣PVD␣step␣2␣succeeded...")
uv<-combineuv(svdrs)
print("Stacking␣matrices␣for␣PVD␣step␣2␣succeeded.")
print("Step␣2:␣Performing␣population-level␣SVD...")
pd<-pvdstep2(uv,p2)
print("PVD␣step␣2␣succeeded.")
print("Step␣3:␣Reconstructing␣tensor/DVF...")
appr<-recon(svdrs,pd)
print("Reconstruction␣succeeded.")
return(appr)

}
```

Listing .2: Revised PVD Step-one in R

```
#The following algorithm performs the first step data reduction in PVD
    algorithm and returns the SVD for each transverse slice as a 4-level [[sub
    ]][[phase]][[dim3(transverse slice)]][[3-tuple(d,u,v)]] nested list

#input needed:
#s: a 3-level [[sub]][[phase]][[dim3(transverse slice)]][dim1,dim2] nested
    list object of images from subject(s).
#p1: percent of variations explained in first step of PVD(*1)

#(*1): the percent of variance explained in SVD is calulated as the percent of
     the sum of the squared singular values from 1 to k.

pvdstep1<-function(rs,p1){

b_t_svd<-Sys.time()
I <- length(rs)
P <- length(rs[[1]])
Z <- length(rs[[1]][[1]])
X <- dim(rs[[1]][[1]][[1]])[1]
Y <- dim(rs[[1]][[1]][[1]])[2]

          svdrs <- vector('list',I)

          #perform SVD (Y=UDV^T)
          #u <- vector('list',I)
          #v <- vector('list',I)

          #sigma <- vector('list',I)

          #listofu <- list()
          #listofv <- list()

for (i in 1:I)
```

```
{
    svdrs[[i]] <- vector('list',P)
    for (p in 1:P)
    {
        svdrs[[i]][[p]] <- vector('list',Z)
        for (z in 1:Z)
        {
            svdrs[[i]][[p]][[z]] <- svd(rs[[i]][[p]][[z]],LINPACK=
              FALSE)
            #approximate SVD based on variations explianed p1
              threshold
            var <- 0
            #for (k in 1:4) #for a fixed number of Vi for each
              subject set to. It was set to 4 because the max
              number for Vi is 4 in the prcedure where Vi is not
               fixed to achieve a certain percentage of
              variations.
            for (k in 1:length(svdrs[[i]][[p]][[z]]$d)) # for
              difference size of Vi for each subject
            {
                var <- var+svdrs[[i]][[p]][[z]]$d[k]^2/sum((
                  svdrs[[i]][[p]][[z]]$d)^2)
                #if the variantions explianed is above a
                  certain percentage then stop
                if (var > p1) break #for different size of Vi
                  for each subject
            }
            svdrs[[i]][[p]][[z]]$u <- svdrs[[i]][[p]][[z]]$u[,1:k]
            svdrs[[i]][[p]][[z]]$v <- svdrs[[i]][[p]][[z]]$v[,1:k]
            svdrs[[i]][[p]][[z]]$d <- svdrs[[i]][[p]][[z]]$d[1:k]
        }
        }
}
e_t_svd<-Sys.time()
e_t_svd-b_t_svd
return(svdrs)

}
```

Listing .3: Revised PVD Step-two in R

```
#The following algorithm performs the second level reduction on combined U and
    V based on p2 percentage of variations expalined criterion and returns a
    list of P & D with a reduced size

#input needed:
#uv: a list containing U & V
#p2: percent of variance exlained in the second step of PVD(*1)

#(*1): the percent of variance explained in SVD is calulated as the percent of
    the sum of the squared singular values from 1 to k.

pvdstep2<-function(uv,p2){

X<-dim(uv[[1]])[1]
Y<-dim(uv[[2]])[1]
U<-uv[[1]]
V<-uv[[2]]
print(dim(U))
print(dim(V))
b_t_ssvd<-Sys.time()
pd<-vector('list',2)
svdu <- svd(U)
```

```
object.size(svdu)
varu <- 0
listofp <- list()
for (x in 1:X)
{
        varu <- varu+svdu$d[x]^2/sum((svdu$d)^2)
        listofp[[x]] <- svdu$u[,x]
        if (varu > p2) break
}

svdv <- svd(V)
object.size(svdv)
varv <- 0
listofd <- list()
for (y in 1:Y)
{
        varv <- varv+svdv$d[y]^2/sum((svdv$d)^2)
        listofd[[y]] <- svdv$u[,y]
        if (varv > p2) break
}

pdf(paste("singular_value_of_U", "pdf", sep="."))
plot(svdu$d)
dev.off()

pdf(paste("singular_value_of_V", "pdf", sep="."))
plot(svdv$d)
dev.off()

#be careful P here is the matrix in PVD
#before it was the index for phase
```

```
P <- do.call(cbind,listofp)
D <- do.call(cbind,listofd)
print(dim(P))
print(dim(D))
pd[[1]]<-P
pd[[2]]<-D
return(pd)
e_t_ssvd<-Sys.time()
e_t_ssvd-b_t_ssvd

}
```

Listing .4: Revised PVD Constructing Cross Population U and V in R

```
#The following algorithm combines u,v from the first step data reduction in
    PVD

#input needed:
#svdrs: SVD for each transverse slice as a 4-level [[sub]][[phase]][[dim3(
    transverse slice)]][[3-tuple(d,u,v)]] nested list
#pd: a list containing P & D

combineuv<-function(svdrs){

I <- length(svdrs)
P <- length(svdrs[[1]])
Z <- length(svdrs[[1]][[1]])
uv <- vector('list',2)
listofu <- list()
listofv <- list()

#combine each subject's each vist's U and V
for (i in 1:I)
{
          for (p in 1:P)
          {
                    for  (z in 1:Z)
                    {
                    listofu[[(P*Z)*(i-1)+Z*(p-1)+z]] <- svdrs[[i]][[p]][[z]]$u
                    listofv[[(P*Z)*(i-1)+Z*(p-1)+z]] <- svdrs[[i]][[p]][[z]]$v
                    }
          }
}

U <- do.call(cbind,listofu)
V <- do.call(cbind,listofv)
uv[[1]]<-U
uv[[2]]<-V
return(uv)

}
```

Listing .5: Revised PVD Constructing Subject Level V_i in R

```
#The following algorithm reconstructs V_ipz matrix for each subject(i) at each
     phase(p) and each transverse slice(z).

#input needed:
#svdrs: SVD for each transverse slice as a 4-level [[sub]][[phase]][[dim3(
    transverse slice)]][[3-tuple(d,u,v)]] nested list
#pd: a list containing P & D

buildv<-function(svdrs,pd){
```

```
b_t_recon_v<-Sys.time()
#construct Vi
P<-pd[[1]]
D<-pd[[2]]
v <- vector('list',length(svdrs))

for (i in 1:length(svdrs))
{
        v[[i]] <- vector('list',length(svdrs[[1]]))
        for (p in 1:length(svdrs[[1]]))
        {
                v[[i]][[p]] <- vector('list',length(svdrs[[1]][[1]]))
                for (z in 1:length(svdrs[[1]][[1]]))
                {
                        v[[i]][[p]][[z]] <- matrix(0,nrow=dim(P)[2], ncol=dim(
                           D)[2])
                        if (length(svdrs[[i]][[p]][[z]]$d)<2)
                        {
                                #here block matrix multiplication can be used
                                   if dim(p)[1] is too big
                                v[[i]][[p]][[z]]<-(t(P)%*%svdrs[[i]][[p]][[z]]
                                   $u)%*%(svdrs[[i]][[p]][[z]]$d%*%diag(1))%*
                                   %(t(svdrs[[i]][[p]][[z]]$v)%*%D)
                        }
                        else
                        {
                                #here block matrix multiplication can be used
                                   if dim(p)[1] is too big
                                v[[i]][[p]][[z]] <- (t(P)%*%svdrs[[i]][[p]][[z
                                   ]]$u)%*%diag(svdrs[[i]][[p]][[z]]$d)%*%(t(
                                   svdrs[[i]][[p]][[z]]$v)%*%D)
```

```
                                        }
                              }
                    }
          }
e_t_recon_img<-Sys.time()
e_t_recon_img-b_t_recon_img
object.size(v)
return(v)

}
```

Listing .6: Revised PVD Reconstructing Tensors in R

```
#The following algorithm reconstructs the image using P & D with reduced sizes
    .

#input needed:
#svdrs: SVD for each transverse slice as a 4-level [[sub]][[phase]][[dim3(
    transverse slice)]][[3-tuple(d,u,v)]] nested list
#pd: a list containing P & D

recon<-function(svdrs,pd){

b_t_recon_img<-Sys.time()
#Reconstruct all images
P<-pd[[1]]
#print(dim(P))
D<-pd[[2]]
#print(dim(D))
apprs <- vector('list',length(svdrs))

for (i in 1:length(svdrs))
```

```
{
        apprs[[i]] <- vector('list',length(svdrs[[1]]))
        for (p in 1:length(svdrs[[1]]))
        {
                apprs[[i]][[p]] <- vector('list',length(svdrs[[1]][[1]]))
                for (z in 1:length(svdrs[[1]][[1]]))
                {
                        apprs[[i]][[p]][[z]] <- matrix(0,nrow=dim(P)[1], ncol=
                            dim(D)[1])
                        if (length(svdrs[[i]][[p]][[z]]$d)<2)
                        {
                                #here block matrix multiplication can be used
                                    if dim(p)[1] is too big
                                apprs[[i]][[p]][[z]]<-P%*%((t(P)%*%svdrs[[i
                                    ]][[p]][[z]]$u)%*%(svdrs[[i]][[p]][[z]]$d%
                                    *%diag(1))%*%(t(svdrs[[i]][[p]][[z]]$v)%*%
                                    D)%*%t(D))
                        }
                        else
                        {
                                #here block matrix multiplication can be used
                                    if dim(p)[1] is too big
                                apprs[[i]][[p]][[z]] <- P%*%((t(P)%*%svdrs[[i
                                    ]][[p]][[z]]$u)%*%diag(svdrs[[i]][[p]][[z
                                    ]]$d)%*%(t(svdrs[[i]][[p]][[z]]$v)%*%D)%*%
                                    t(D))
                        }
                }
        }
}
e_t_recon_img<-Sys.time()
```

```
e_t_recon_img-b_t_recon_img
object.size(apprs)
return(apprs)

}
#save(apprs, file="/home/kangk4/dissertation/data/pvd_apprs.rda")
```

Listing .7: Restructure a Fifth-order Tensor in R

```
#The following algorithm reaaranges the image structure from
#(1)a 1-level list of 5D [dim1,dim2,dim3,phase,(DVF contains x,y,z cor)]
    arrays or
#(2)a 2-level list of 5D [dim1,dim2,dim3,phase,(DVF contains x,y,z cor)]
    arrays *(a)
#into a 3-level ([[sub]][[phase]][[dim3(transverse slice)]]) nested list of 2D
     arrays [dim1*3(DVF contains x,y,z cor),dim2] aka for each subject at each
     phase for each transevrse slice, the algorithm is stacking the transverse
     slice for each (x,y,z) cor in DVF to make a long matrix.

#*(a) The reason we have two types of input struture is because that the
    readMat function (from R.mathlab package) will sometimes return a list
    with one of two different structures. In case (2) the second level is a
    dummy level. The level(s) in either (1) or (2) should have length 1
    assuming you are reading in ".mat" file per patient using the readMat
    function.

#input needed:
#s: the original DVF image is stored as a list of either struture (1) or (2)
l1o5dTOl3o2d <- function(s){

        if (typeof(s)!="list") {stop("The␣function␣l1o5dTOl3o2d␣only␣takes␣a␣
            list␣as␣an␣input.")}
```

```
if (typeof(s)=="list")
{

if (!is.null(dim(s))) {stop("The␣first␣level␣in␣input␣data␣should␣have
    ␣length␣exactly␣1.")}
if (is.null(dim(s)))
{
        I<-length(s)
        if (!is.null(dim(s[[I]])))
        {
                P       <- length(s[[1]][1,1,1,,1])
                X       <- length(s[[1]][,1,1,1,1])
                Y       <- length(s[[1]][1,,1,1,1])
                Z       <- length(s[[1]][1,1,,1,1])
                COR <- length(s[[1]][1,1,1,1,])
                order <- 1
        }
        if (is.null(dim(s[[I]])) && is.null(dim(s[[I]][[1]]))){stop("
            Detected␣a␣second␣nested␣level␣that␣does␣not␣meet␣
            criterion.␣The␣second␣nested␣level␣in␣input␣data␣should␣
            have␣length␣exactly␣1.")}
        if (is.null(dim(s[[I]])) && !is.null(dim(s[[I]][[1]])))
        {
                P       <- length(s[[1]][[1]][1,1,1,,1])
                X       <- length(s[[1]][[1]][,1,1,1,1])
                Y       <- length(s[[1]][[1]][1,,1,1,1])
                Z       <- length(s[[1]][[1]][1,1,,1,1])
                COR <- length(s[[1]][[1]][1,1,1,1,])
                order <- 2
        }
}
```

```
        }

        rs  <- vector('list',I)

        for (i in 1:I)
        {
                rs[[i]] <- vector('list',P)
          for (p in 1:P)
          {
                  rs[[i]][[p]] <- vector('list',Z)
            for (z in 1:Z)
            {
                        rs[[i]][[p]][[z]] <- matrix(0,nrow=X*COR, ncol=Y)
                  for (cor in 1:COR)
                  {
                        if (order==1){rs[[i]][[p]][[z]][(X*(cor-1)+1):(X*cor)
                            ,] <- s[[i]][,,z,p,cor]}
                        if (order==2){rs[[i]][[p]][[z]][(X*(cor-1)+1):(X*cor)
                            ,] <- s[[i]][[1]][,,z,p,cor]}
                  }
            }
          }
        }
        return(rs)
}
```

Listing .8: Supplemental Algorithm for Computing Voxel-based Absolute Errors in R

```
#The following algorithm will convert a three-level ([[sub]][[phase]][[dim3(
    transverse slice)]])[dim1,dim2] list of 2D arrays into a matrix by (1)
    converting 2D arrays into a vector for each level-one+level-two+level-
    three list element and (2) rowbind-ing each vector into a longer vector
l3o2dTOv<-function(l){
        source('/home/kangk4/dissertation/code/source/mtov.R')
        sub     <- length(l)
        phase   <- length(l[[1]])
        Z       <- length(l[[1]][[1]])
        dim1    <- length(l[[1]][[1]][[1]][,1])
        dim2    <- length(l[[1]][[1]][[1]][1,])
        dim     <- dim1*dim2
        sdim    <- dim1*dim2*Z
        tdim    <- sdim*phase
        v       <- rep(0,tdim*sub)
        m       <- matrix(0,nrow=dim1,ncol=dim2)
        for (i in 1:sub){
                for (j in 1:phase){
                        for (z in 1:Z){
                        m <- l[[i]][[j]][[z]]
                        v[(tdim*(i-1)+sdim*(j-1)+dim*(z-1)+1):(tdim*(i-1)+sdim
                            *(j-1)+dim*z)]<-mtov(m)
                        }
                }
        }
        return(v)
}
```

Listing .9: Supplemental Algorithm for Reconstructing a Fifth-order Tensor in R

```
#The following algorithm will vectorize a matrix by stacking its columns
mtov<-function(m){
        m     <- data.matrix(m)
        dim1  <- dim(m)[1]
        dim2  <- dim(m)[2]
```

```
          v      <- rep(0,dim1*dim2)
          for (i in 1:dim2){
                    v[(dim1*(i-1)+1):(dim1*i)] <- m[,i]
          }
          return(v)
}

#The following algorithm will convert vectorize a three-dimensional (3D) array
     by vectorize each 2D slice then cbind those vectors to form one matrix.
    And finally vectorize the one matrix into a long vector
threedtov<-function(threed){
          dim1  <- length(threed[,1,1])
          dim2  <- length(threed[1,,1])
          dim3  <- length(threed[1,1,])
          m     <- matrix(0,nrow=dim1*dim2,ncol=dim3)
          array <- array(0, dim=c(dim1,dim2))
          for (i in 1:dim3){
                    array <- threed[,,i]
                    m[,i] <- mtov(array)
          }
          v      <- mtov(m)
          return(v)
}

#The following algorithm will convert a 4D arrays into a matrix by (1)
    vectorizing each 3D array into a long vector and (2) cbind-ing vectors
    from different phases (4th dimension) into a matrix
fourdtom<-function(fourd){
          dim1  <- length(fourd[,1,1,1])
          dim2  <- length(fourd[1,,1,1])
          dim3  <- length(fourd[1,1,,1])
```

```
        dim4  <- length(fourd[1,1,1,])
        m     <- matrix(0,nrow=dim1*dim2*dim3,ncol=dim4)
        array <- array(0, dim=c(dim1,dim2,dim3))
        for (i in 1:dim4){
                array <- fourd[,,,i]
                m[,i] <- threedtov(array)
        }
        return(m)
}

#The following algorithm will convert a 5D arrays into a matrix by (1)
    converting a 4D arrays into a matrix and (2) stacking each matrix from the
     DVF (x,y,z) (5th dimension) into a longer matrix
fivedtom<-function(fived){
        dim1  <- length(fived[,1,1,1,1])
        dim2  <- length(fived[1,,1,1,1])
        dim3  <- length(fived[1,1,,1,1])
        dim4  <- length(fived[1,1,1,,1])
        dim5  <- length(fived[1,1,1,1,])
        dim   <- dim1*dim2*dim3
        m     <- matrix(0,nrow=dim*dim5,ncol=dim4)
        array <- array(0, dim=c(dim1,dim2,dim3,dim4))
        for (i in 1:dim5){
                array                                  <- fived[,,,,i]
                m[(dim*(i-1)+1):(dim*i),] <- fourdtom(array)
        }
        return(m)
}

#The following algorithm will convert a list of 5D arrays into a matrix by (1)
     converting 5D arrays into a matrix for each level-one list element and
    (2) rbind-ing each matrix into a longer matrix
ltom<-function(l){
        sub <- length(l)
        dim1   <- length(l[[1]][[1]][,1,1,1,1])
        dim2   <- length(l[[1]][[1]][1,,1,1,1])
        dim3   <- length(l[[1]][[1]][1,1,,1,1])
        dim4   <- length(l[[1]][[1]][1,1,1,,1])
        dim5   <- length(l[[1]][[1]][1,1,1,1,])
        dim    <- dim1*dim2*dim3*dim5
        m      <- matrix(0,nrow=dim*sub,ncol=dim4)
        array  <- array(0, dim=c(dim1,dim2,dim3,dim4,dim5))
        for (i in 1:sub){
                if(dim5 == 1){
                        array <- l[[i]][[1]][,,,,]
                        m[(dim*(i-1)+1):(dim*i),] <- fourdtom(array)
                }
                if (dim5 > 1){
                        array <- l[[i]][[1]][,,,,]
                        m[(dim*(i-1)+1):(dim*i),] <- fivedtom(array)
                }
        }
        return(m)
}
```

Listing .10: An Example of Running Revised PVD in R

```
t_start<-Sys.time()
.libPaths()
source('/home/kangk4/dissertation/code/source/l1o5dTOl3o2d.R')
source('/home/kangk4/dissertation/code/source/pvd.R')
source('/home/kangk4/dissertation/code/source/l3o2dTOv.R')
source('/home/kangk4/dissertation/code/source/pden.R')
source('/home/kangk4/dissertation/code/source/pdenfix.R')

####Read in image data####
library(R.matlab)
#case01<-readMat('/home/kangk4/dissertation/data/TenPatientsHalfLung/DVFmat/
    case01.mat')
#save(case01,file="/home/kangk4/dissertation/data/TenPatientsHalfLung/RDA/
    case01.rda")
#If you have the DVF saved as an R object, then use the following will be
    faster
load("/home/kangk4/dissertation/data/TenPatientsHalfLung/RDA/rscase01.rda")
#load("/home/kangk4/dissertation/data/dvfx.rda")
dvf<-list(rscase01)

#make a list containing all subjects
s<-list(dvf)
str(s)
pvd_start<-Sys.time()
rs<-l1o5dTOl3o2d(s)
apprs<-pvd(rs,.95,.95)

vrs<-l3o2dTOv(rs)
vapprs<-l3o2dTOv(apprs)
difvec<-abs(vapprs-vrs)
save(difvec,file="/home/kangk4/dissertation/data/pvd_difvec_case01.1.rda")
length(difvec)
length(difvec[difvec>0.20])
length(difvec[difvec>0.15])
mse_pvd<-sum(difvec^2)/length(difvec)
mse_pvd
max(difvec)
```

```
pvd_end<-Sys.time()
pvd_end-pvd_start
```

.2 PTD Algorithm

Listing .11: PTD in R

```
#The following algorithm performs Population Tucker Decomposition on a fifth-
    order tensor by first unfolding the fifth mode along mode-1 and then
    performing Tucker Decomposition on a third-order tensor when fixing mode 4
     and combining factor matrices for mode 1, 2, and 3 across mode 4 mode-
    wise and performs a SVD on each of the three combined population factor
    matrices.

#input needed:
#tnsr: 5th-order tensor (DVF)
#dim1: size of the first dimension of the core tensor
#dim2: size of the second dimension of the core tensor
#dim3: size of the third dimension of the core tensor
#p1: percentage of SVD of the combined first factor matrix
#p2: percentage of SVD of the combined second factor matrix
#p3: percentage of SVD of the combined third factor matrix

#input
#an approximated fifth-order tensor

ptd<-function(tnsr,dim1,dim2,dim3,p1,p2,p3){

require("rTensor")

tc2<-tnsr
```

```
if(p1<0 || p2<0 || p3<0 ||p1>1 ||p2>1 ||p3>1){stop("p1␣and␣p2␣must␣be␣from␣
    [0,1]!")}
if (typeof(tc2)!="S4") {stop("The␣function␣pvd␣only␣takes␣a␣list␣as␣an␣input."
    )}
if (typeof(tc2)=="S4") {print("input␣class␣check␣passed.")}

tensordim1<-dim(tc2)[1]
tensordim2<-dim(tc2)[2]
tensordim3<-dim(tc2)[3]
tensordim4<-dim(tc2)[4]
tensordim5<-dim(tc2)[5]

rtc2<-as.tensor(array(0,dim=c(tensordim1*tensordim5,tensordim2,tensordim3,
    tensordim4)))
for (i in 1:tensordim5){
        rtc2[(tensordim1*(i-1)+1):(tensordim1*i),,,]<-tc2[,,,,i]
}

A<-matrix(0,nrow=tensordim1*tensordim5,ncol=dim1*tensordim4)
B<-matrix(0,nrow=tensordim2,ncol=dim2*tensordim4)
C<-matrix(0,nrow=tensordim3,ncol=dim3*tensordim4)
S<-array(0,dim=c(dim1,dim2,dim3,tensordim4))

print("Step␣1:␣Performing␣subject-level␣Tucker␣Decomposition...")
for(i in 1:tensordim4){
        tuc<-tucker(rtc2[,,,i],ranks=c(dim1,dim2,dim2),max_iter=100,tol=1e-15)
        print(tuc$norm_percent)
        A[,((i-1)*dim1+1):(i*dim1)]<-tuc$U[[1]]
        B[,((i-1)*dim2+1):(i*dim2)]<-tuc$U[[2]]
        C[,((i-1)*dim3+1):(i*dim3)]<-tuc$U[[3]]
```

```
          S[,,,i]<-tuc$Z@data
}

print(dim(A))
print(dim(B))
print(dim(C))
print(dim(S))

print("PTD␣step␣1␣succeeded.")

print("Step␣2:␣Performing␣population-level␣SVD...")
svda<-svd(A)
X<-dim(A)[1]
vara <- 0
listofa <- list()
for (x in 1:X)
{
          vara <- vara+svda$d[x]^2/sum((svda$d)^2)
          listofa[[x]] <- svda$u[,x]
          if (vara > p1) break
}

svdb<-svd(B)
X<-dim(B)[1]
varb <- 0
listofb <- list()
for (x in 1:X)
{
          varb <- varb+svdb$d[x]^2/sum((svdb$d)^2)
          listofb[[x]] <- svdb$u[,x]
          if (varb > p2) break
```

```
}

svdc<-svd(C)
X<-dim(C)[1]
varc <- 0
listofc <- list()
for (x in 1:X)
{
        varc <- varc+svdc$d[x]^2/sum((svdc$d)^2)
        listofc[[x]] <- svdc$u[,x]
        if (varc > p3) break
}
print("PTD␣step␣2␣succeeded.")

nA <- do.call(cbind,listofa)
nB <- do.call(cbind,listofb)
nC <- do.call(cbind,listofc)
print(dim(nA))
print(dim(nB))
print(dim(nC))

print("Step␣3:␣Reconstructing␣tensor/DVF...")
nS<-as.tensor(array(0,dim=c(dim(nA)[2],dim(nB)[2],dim(nC)[2],tensordim4)))
liz<-list('mat1'=nA,'mat2'=nB,'mat3'=nC)
for(i in 1:tensordim4){
        An<-t(nA)%*%A[,((i-1)*dim1+1):(i*dim1)]
        Bn<-t(nB)%*%B[,((i-1)*dim2+1):(i*dim2)]
        Cn<-t(nC)%*%C[,((i-1)*dim3+1):(i*dim3)]
        liz<-list('mat1'=An,'mat2'=Bn,'mat3'=Cn)
        nS[,,,i]<-ttl(as.tensor(S[,,,i]),liz,ms=c(1,2,3))
}
```

```
nt<-ttl(nS,list(nA,nB,nC),ms=c(1,2,3))
print("Reconstruction␣succeeded.")

return(nt)

}
```

Listing .12: Supplemental Algorithm for Restoring Tensors in R

```
library(rTensor)

vtom<-function(v,c){
          #c is the number of phases
          if(class(v)=='numeric'){
                    r <- length(v)/c
                    m <- matrix(v,nrow=r,ncol=c)
          }
          return(m)
}

mtol<-function(m,mode1,mode2,mode3,mode4,mode5){
          mdim1<-dim(m)[1]
          mdim2<-dim(m)[2]
          sec<-mdim1/mode5
          tnsr<-array(0,dim=c(mode1,mode2,mode3,mode4,mode5))
          for (i in 1:mode5){
                    for (j in 1:mode4){
                              tnsr[,,,j,i]<-array(m[((i-1)*sec+1):(i*sec),j],dim=c(
                                  mode1,mode2,mode3))
                    }
          }
```

```
        return(tnsr)
}

norm<-function(tnsr ,mode1,mode2,mode3,mode4){
        approxnorm<-array(0,dim=c(mode1,mode2,mode3,mode4))
        approxnorm<-sqrt(tnsr[,,,,1]^2+tnsr[,,,,2]^2+tnsr[,,,,3]^2)
        return(approxnorm)
}

binary<-function(approxnorm,standard){
        approxnorm[approxnorm<standard]<-0
        approxnorm[approxnorm>=standard]<-1
        return(approxnorm)
}
```

Listing .13: An Example of Running PTD in R

```
.libPaths()
source('/home/kangk4/dissertation/code/source/ppden.R')
source('/home/kangk4/dissertation/code/source/pdenfix.R')
source('/home/kangk4/dissertation/code/source/ptd.R')
require("tictoc")
require("rTensor")
load("/home/kangk4/dissertation/data/TenPatientsHalfLung/RDA/rscase10.rda")

tc2<-as.tensor(rscase10)

tensordim1<-dim(tc2)[1]
tensordim2<-dim(tc2)[2]
tensordim3<-dim(tc2)[3]
tensordim4<-dim(tc2)[4]
tensordim5<-dim(tc2)[5]
```

```
rtc2<-as.tensor(array(0,dim=c(tensordim1*tensordim5,tensordim2,tensordim3,
    tensordim4)))
for (i in 1:tensordim5){
        rtc2[(tensordim1*(i-1)+1):(tensordim1*i),,,]<-tc2[,,,,i]
}

nt<-ptd(tc2,80,30,30,0.95,0.95,0.95)

ptdend<-Sys.time()
ptdend-ptdstart
error<-nt-rtc2
save(error,file="/home/kangk4/dissertation/data/ptd_error_case10.rda")

length(difvec[difvec>0.15])
length(difvec[difvec>0.20])
max(difvec)
mse_pvd<-sum(difvec^2)/length(difvec)
mse_pvd
```

.3 Supplemental Plotting Algorithm

Listing .14: Supplemental Algorithm for Plotting Tail Portion Density Plot of Voxel-based Absolute Errors R

```
library(ggplot2)
library(reshape2)


ppden<-function(pca,pvd,name){

d1<-density(pca)
```

```
d2<-density(pvd)

pdf(paste(name, "pdf", sep="."))

par(mfrow=c(2,2))

plot(range(d1$x,d2$x),range(0,2),type="n",xlab="Error",ylab="Density")
lines(d1,col="red")
lines(d2,col="blue")
legend("topright", legend=c("PVD", "PTD"),
        col=c("red", "blue"),lty=1)

plot(range(d1$x,d2$x),range(0,1),type="n",xlab="Error",ylab="Density")
lines(d1,col="red")
lines(d2,col="blue")
legend("topright", legend=c("PVD", "PTD"),
        col=c("red", "blue"),lty=1)

plot(range(d1$x,d2$x),range(0,0.1),type="n",xlab="Error",ylab="Density")
lines(d1,col="red")
lines(d2,col="blue")
legend("topright", legend=c("PVD", "PTD"),
        col=c("red", "blue"),lty=1)

plot(range(d1$x,d2$x),range(0,0.01),type="n",xlab="Error",ylab="Density")
lines(d1,col="red")
lines(d2,col="blue")
legend("topright", legend=c("PVD", "PTD"),
        col=c("red", "blue"),lty=1)

dev.off()
```

```
}
```

Listing .15: Supplemental Algorithm for Plotting Cross-phase Animation in R

```
du#animation

setwd("/Users/Kingston/Desktop/dummy")
a = 2
b = 1
slice = a*b
phase = 10

for(i in 1:phase){
        # creating a name for each plot file with leading zero
        if (i < 10) {name = paste("000",i,"plot",sep="")}
        if (i < 100 && i >= 10) {name = paste('00',i,'plot',sep='')}
        if (i >= 100) {name = paste('0',i,'plot',sep='')}

#saves the plot as a .png file in the working directory
        png(paste(name, "png", sep="."),units="px", width=2400, height=2100,
            res=300)
        par(mfrow=c(a,b))
        #for(j in 1:slice){
        #s=20*(j-1)+1
        #image(rscase02[,,s,i,3],col=heat.colors(100),zlim=range(unlist(
            rscase02[,,81,,3])),main=paste("slice",s,"phase",i-1,sep=" "))
        image(nt@data[,,21,i],zlim=range(unlist(rtc2@data[,,21,1])),main=paste
            ("reconstrcuted","slice",21,"phase",i-1,sep="␣"))
        image(rtc2@data[,,21,i],zlim=range(unlist(rtc2@data[,,21,1])),main=
            paste("actual","slice",21,"phase",i-1,sep="␣"))
        #}
        dev.off()
}

#in terminal run convert *.png -delay 7 -loop 0 lung.gif
```

Listing .16: An Example of Plotting Cross Transverse Slices with Large Voxel-based Deviations in R

```
t_start<-Sys.time()
.libPaths()
library(tictoc)
library(rTensor)
library(gplots)
library("RColorBrewer")
library(ComplexHeatmap)
library(circlize)
source('/Users/Kingston/Desktop/source/pcasource.R')
source('/Users/Kingston/Desktop/source/tensorization.R')
source('/Users/Kingston/Desktop/source/l1o5dTOl3o2d.R')
source('/Users/Kingston/Desktop/source/l3o2dTOv.R')
source('/Users/Kingston/Desktop/source/mtov.R')

load("/Users/Kingston/Desktop/pca_difvec_case10.rda")
pcacase10vec<-difvec
pcacase10mat<-vtom(pcacase10vec,10)
pcacase10tns<-mtol(pcacase10mat,304,148,108,10,3)
pcacase10norm<-norm(pcacase10tns,304,148,108,10)
pcacase10bi<-binary(pcacase10norm,0.15)

pcat50p1<-pcacase10norm[,,50,1]
pcat51p1<-pcacase10norm[,,51,1]
pcat52p1<-pcacase10norm[,,52,1]
```

```
pcat53p1<-pcacase10norm[,,53,1]
pcat54p1<-pcacase10norm[,,54,1]

load("/Users/Kingston/Desktop/pvd_difvec_case10.1.rda")
pvdcase10vec<-difvec
pvdcase10mat<-pvdvtom(pvdcase10vec,304,148,108,10,3)
pvdcase10tns<-pvdmtot(pvdcase10mat,304,148,108,10,3)
pvdcase10norm<-norm(pvdcase10tns,304,148,108,10)
pvdcase10bi<-binary(pvdcase10norm,0.15)

pvdt50p1<-pvdcase10norm[,,50,1]
pvdt51p1<-pvdcase10norm[,,51,1]
pvdt52p1<-pvdcase10norm[,,52,1]
pvdt53p1<-pvdcase10norm[,,53,1]
pvdt54p1<-pvdcase10norm[,,54,1]

load("/Users/Kingston/Desktop/ptd_error_case10.rda")
ptdcase10tnsr<-error
ptdcase10tns<-ptdttot(ptdcase10tnsr,304,148,108,10,3)
ptdcase10norm<-norm(ptdcase10tns,304,148,108,10)
ptdcase10bi<-binary(ptdcase10norm,0.15)

ptdt50p1<-ptdcase10norm[,,50,1]
ptdt51p1<-ptdcase10norm[,,51,1]
ptdt52p1<-ptdcase10norm[,,52,1]
ptdt53p1<-ptdcase10norm[,,53,1]
ptdt54p1<-ptdcase10norm[,,54,1]

load("/Users/Kingston/Desktop/rawcase10.rda")
raw<-rawcase10
rawcase10<-rawcase10[[1]]
```

```
rawcase10t50p1<-rawcase10[,,50,1]
rawcase10t51p1<-rawcase10[,,51,1]
rawcase10t52p1<-rawcase10[,,52,1]
rawcase10t53p1<-rawcase10[,,53,1]
rawcase10t54p1<-rawcase10[,,54,1]

rawmax<-max(rawcase10t50p1,rawcase10t51p1,rawcase10t52p1,rawcase10t53p1,
    rawcase10t54p1)+100
rawmin<-min(rawcase10t50p1,rawcase10t51p1,rawcase10t52p1,rawcase10t53p1,
    rawcase10t54p1)
rawmed<-median(c(rawcase10t50p1,rawcase10t51p1,rawcase10t52p1,rawcase10t53p1,
    rawcase10t54p1))
colors=c(seq(rawmin,rawmed,length=50),seq(rawmed,rawmax,length=50),seq(rawmax
    ,5000,length=10))
rawpalette<-colorRampPalette(c("black","white","red"))(n=512)
colors=c(seq(rawmin,rawmed,length=50),seq(rawmed,rawmax,length=50))
rawpalette2<-colorRampPalette(c("black","white"))(n=512)

rawcase10t50p1pca<-rawcase10t50p1
rawcase10t50p1pca[which(pcat50p1 > 0.15, arr.ind = TRUE)]<-5000
rawcase10t51p1pca<-rawcase10t51p1
rawcase10t51p1pca[which(pcat51p1 > 0.15, arr.ind = TRUE)]<-5000
rawcase10t52p1pca<-rawcase10t52p1
rawcase10t52p1pca[which(pcat52p1 > 0.15, arr.ind = TRUE)]<-5000
rawcase10t53p1pca<-rawcase10t53p1
rawcase10t53p1pca[which(pcat53p1 > 0.15, arr.ind = TRUE)]<-5000
rawcase10t54p1pca<-rawcase10t54p1
rawcase10t54p1pca[which(pcat54p1 > 0.15, arr.ind = TRUE)]<-5000

rawcase10t50p1pvd<-rawcase10t50p1
rawcase10t50p1pvd[which(pvdt50p1 > 0.15, arr.ind = TRUE)]<-5000
```

```
rawcase10t51p1pvd<-rawcase10t51p1
rawcase10t51p1pvd[which(pvdt51p1 > 0.15, arr.ind = TRUE)]<-5000
rawcase10t52p1pvd<-rawcase10t52p1
rawcase10t52p1pvd[which(pvdt52p1 > 0.15, arr.ind = TRUE)]<-5000
rawcase10t53p1pvd<-rawcase10t53p1
rawcase10t53p1pvd[which(pvdt53p1 > 0.15, arr.ind = TRUE)]<-5000
rawcase10t54p1pvd<-rawcase10t54p1
rawcase10t54p1pvd[which(pvdt54p1 > 0.15, arr.ind = TRUE)]<-5000

rawcase10t50p1ptd<-rawcase10t50p1
rawcase10t50p1ptd[which(ptdt50p1 > 0.030, arr.ind = TRUE)]<-5000
rawcase10t51p1ptd<-rawcase10t51p1
rawcase10t51p1ptd[which(ptdt51p1 > 0.030, arr.ind = TRUE)]<-5000
rawcase10t52p1ptd<-rawcase10t52p1
rawcase10t52p1ptd[which(ptdt52p1 > 0.030, arr.ind = TRUE)]<-5000
rawcase10t53p1ptd<-rawcase10t53p1
rawcase10t53p1ptd[which(ptdt53p1 > 0.030, arr.ind = TRUE)]<-5000
rawcase10t54p1ptd<-rawcase10t54p1
rawcase10t54p1ptd[which(ptdt54p1 > 0.030, arr.ind = TRUE)]<-5000

#par(mfrow = c(2, 5))
#heatmap.2(as.matrix(rawcase10t50p1pca), col=rawpalette, main = "PCA-C10T50P1
   ",tracecol=NA,density.info='none', key=FALSE)

par(mfrow = c(3, 5),mar = c(3, 1, 1, 0))
image(as.matrix(t(rawcase10t50p1pca[304:1,])), col=rawpalette, main = "PCA-
   C10T50P1", axes=FALSE)

image(as.matrix(t(rawcase10t51p1pca[304:1,])), col=rawpalette, main = "PCA-
   C10T51P1", axes=FALSE)
```

```
image(as.matrix(t(rawcase10t52p1pca[304:1,])), col=rawpalette, main = "PCA–
    C10T52P1", axes=FALSE)

image(as.matrix(t(rawcase10t53p1pca[304:1,])), col=rawpalette, main = "PCA–
    C10T53P1", axes=FALSE)

image(as.matrix(t(rawcase10t54p1pca[304:1,])), col=rawpalette, main = "PCA–
    C10T54P1", axes=FALSE)

image(as.matrix(t(rawcase10t50p1pvd[304:1,])), col=rawpalette2, main = "PVD–
    C10T50P1", axes=FALSE)

image(as.matrix(t(rawcase10t51p1pvd[304:1,])), col=rawpalette2, main = "PVD–
    C10T51P1", axes=FALSE)

image(as.matrix(t(rawcase10t52p1pvd[304:1,])), col=rawpalette2, main = "PVD–
    C10T52P1", axes=FALSE)

image(as.matrix(t(rawcase10t53p1pvd[304:1,])), col=rawpalette, main = "PVD–
    C10T53P1", axes=FALSE)

image(as.matrix(t(rawcase10t54p1pvd[304:1,])), col=rawpalette2, main = "PVD–
    C10T54P1", axes=FALSE)

image(as.matrix(t(rawcase10t50p1ptd[304:1,])), col=rawpalette2, main = "PTD–
    C10T50P1", axes=FALSE)

image(as.matrix(t(rawcase10t51p1ptd[304:1,])), col=rawpalette2, main = "PTD–
    C10T51P1", axes=FALSE)

image(as.matrix(t(rawcase10t52p1ptd[304:1,])), col=rawpalette2, main = "PTD–
    C10T52P1", axes=FALSE)

image(as.matrix(t(rawcase10t53p1ptd[304:1,]))), col=rawpalette, main = "PTD-
    C10T53P1", axes=FALSE)

image(as.matrix(t(rawcase10t54p1ptd[304:1,]))), col=rawpalette, main = "PTD-
    C10T54P1", axes=FALSE)
```