RANDOMNESS AS A COMPUTATIONAL STRATEGY: ON MATRIX AND TENSOR DECOMPOSITIONS

N. Benjamin Erichson

A Thesis Submitted for the Degree of PhD
at the
University of St Andrews

2017

Identifiers to use to cite or link to this thesis:

DOI: https://doi.org/10.17630/10023-16693
http://hdl.handle.net/10023/16693

Randomness as a Computational Strategy:

On Matrix and Tensor Decompositions

N. Benjamin Erichson

University of St Andrews

This thesis is submitted in partial fulfilment for the degree
Doctor of Philosophy
at the
University of St Andrews

July 2017

Dedicated to my Father and Gabriela.

"Love many things, for therein lies the true strength, and whosoever loves much performs much, and can accomplish much, and what is done in love is done well."

— Vincent van Gogh

Declaration

Candidate's declaration

I, N. Benjamin Erichson, hereby certify that this thesis, which is approximately 45000 words in length, has been written by me, and that it is the record of work carried out by me, or principally by myself in collaboration with others as acknowledged, and that it has not been submitted in any previous application for a higher degree.

I was admitted as a research student in October 2013 and as a candidate for the degree of PhD also in October 2013; the higher study for which this is a record was carried out in the University of St Andrews between 2013 and 2018.

Date:
Signature of candidate:

Supervisor's declaration

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of PhD in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

Date:
Signature of supervisor:

Permission for publication

In submitting this thesis to the University of St Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force,

subject to any copyright vested in the work not being affected thereby. I also understand that the title and the abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker, that my thesis will be electronically accessible for personal or research use unless exempt by award of an embargo as requested below, and that the library has the right to migrate my thesis into new electronic forms as required to ensure continued access to the thesis. I have obtained any third-party copyright permissions that may be required in order to allow such access and migration, or have requested the appropriate embargo below.

The following is an agreed request by candidate and supervisor regarding the publication of this thesis:

PRINTED COPY

Embargo on all or part of print copy for a period of two years on the following ground(s):

- Publication would preclude future publication.

ELECTRONIC COPY

Embargo on all or part of electronic copy for a period of two years on the following ground(s):

- Publication would preclude future publication.

Supporting statement for electronic embargo request: The ability to publish work found within this thesis may be compromised if it is not embargoed.

Date:
Signature of candidate:
Signature of supervisor:

N. Benjamin Erichson
July 2017

Acknowledgements

I would like to thank all the incredible people who I have met and who have supported me during the last three years. Most notably, I would like to thank my advisor Carl Donovan for making this endeavor possible and for his support. One of the most enjoyable parts of my PhD program has been the opportunity to travel. I spent half a year in Auckland and almost a year in Seattle. During this time, I have been incredibly fortunate to work with an amazing group of collaborators: J. Nathan Kutz, Steven L. Brunton, Cameron Walker, and Krithika Manohar. Much of this thesis is built upon work with these collaborators and could not have been done without them. My special thanks goes to Nathan, whose inspiration and guidance was invaluable as well as for the many fantastic coffees he prepared. Finally, I must thank my wonderful friends for motivating and supporting me over the years: Daniel, Ardita, Yeannie, Quentin, Mikkel, Sophie, Ingrid, Livia and Tim, with whom I had my best study breaks. In particular, I would like to thank Claire for being there when I needed you the most. Most importantly, I want to express my gratitude to my sister, who was always there for me and who encouraged me so much.

Abstract

Matrix and tensor decompositions are fundamental tools for finding structure and data processing. In particular, the efficient computation of low-rank matrix approximations is an ubiquitous problem in the area of machine learning and elsewhere. However, massive data arrays pose a computational challenge for these techniques, placing significant constraints on both memory and processing power. Recently, the fascinating and powerful concept of randomness has been introduced as a strategy to ease the computational load of deterministic matrix and data algorithms. The basic idea of these algorithms is to employ a degree of randomness as part of the logic in order to derive from a high-dimensional input matrix a smaller matrix, which captures the essential information of the original data matrix. Subsequently, the smaller matrix is then used to efficiently compute a near-optimal low-rank approximation. Randomized algorithms have been shown to be robust, highly reliable, and computationally efficient, yet simple to implement. In particular, the development of the randomized singular value decomposition can be seen as a milestone in the era of 'big data'. Building up on the great success of this probabilistic strategy to compute low-rank matrix decompositions, this thesis introduces a set of new randomized algorithms. Specifically, we present a randomized algorithm to compute the dynamic mode decomposition, which is a modern dimension reduction technique designed to extract dynamic information from dynamical systems. Then, we advocate the randomized dynamic mode decomposition for background modeling of surveillance video feeds. Further, we show that randomized algorithms are embarrassingly parallel by design and that graphics processing units (GPUs) can be utilized to substantially accelerate the computations. Finally, the concept of randomized algorithms is generalized for tensors in order to compute the canonical CANDECOMP/PARAFAC (CP) decomposition.

Table of contents

List of Figures

List of Tables

Some Nomenclature

Roman Symbols

$\mathbf{A}$	Matrix of dimension $m \times n$.
$\mathbf{A}_k$	Best rank k approximation of dimension $m \times k$.
$\mathbf{B}$	Smaller matrix of dimension $k \times n$.
$\mathbf{C}$	Correlation matrix.
$\mathbf{F}$	F-measure.
k	Target rank.
$\mathbf{L}$	Denotes both the loading and low-rank matrix.
p	Sampling parameter equal to $l = k + p$.
$\mathbf{M}$	Linear map.
$\tilde{\mathbf{M}}$	Projected linear map.
$\mathbf{P}$	Orthogonal projector.
p	Oversampling parameter.
$\mathbf{Q}$	Orthonormal basis matrix of dimension $m \times k$.
q	Power iteration parameter.
r	Numerical rank.
$\mathbf{S}$	Sparse matrix.
$\mathbf{U}$	Left singular vectors.

$\mathbf{V}$	Right singular vectors.
$\mathbf{W}$	Eigenvectors.
$\mathbf{x}$	Vector of dimension n.
$\mathbf{Y}$	Samples matrix of dimension $m \times k$.
$\mathbf{Z}$	Principal components.

Greek Symbols

$\boldsymbol{\Lambda}$	Diagonal matrix containing the eigenvalues.
λ	Eigenvalue.
$\boldsymbol{\Omega}$	Random test matrix of dimension $k \times n$.
ω	Random vector.
$\boldsymbol{\Phi}$	Dynamic Modes.
$\boldsymbol{\Sigma}$	Diagonal matrix containing the singular values.
σ	Singular value.

Subscripts

2	Spectral norm.
F	Frobenius norm.
i, j	Indices of matrix/vector elements.
t	Time indice.

Other Symbols

$\boldsymbol{\mathcal{B}}$	Smaller (Compressed) Tensor.
$\mathcal{L}$	Lagrangian function.
$\mathcal{V}$	Vandermonde matrix.
$\boldsymbol{\mathcal{X}}$	Tensor.

Acronyms / Abbreviations

ALS Alternating Least Squares.

BCD Block Coordinate Decent.

BMC Background Models Challenge.

CD Change Detection Dataset.

CP CANDECOMP/PARAFAC.

CPD CANDECOMP/PARAFAC Decomposition.

ALU Arithmetic Logic Unit.

CPU Central Processing Unit.

DMD Dynamic Mode Decomposition.

FPS Frames Per Second.

GPU Graphical Processing Unit.

HD High Definition.

MKL Intel Math Kernel Library.

NRMSE Normalized Root Mean Squared Error.

PCA Principal Component Analysis.

PC Principal Components.

RPCA Robust Principal Component Analysis.

rPCA Randomized Principal Component Analysis.

rRPCA Randomized Robust Singular Value Decomposition.

rSVD Randomized Singular Value Decomposition.

SVD Singular Value Decomposition.

Chapter 1

Introduction

"Truth is much too complicated to allow anything but approximations."
— John von Neumann

1.1 The Big Picture

Randomness is a fascinating and powerful concept, deeply embedded in nature. Certainly, the vast and amazing variety of life would be unlikely without the element of randomness (Monod, 1971). However, randomness also plays an indispensable role in science. The whole body of inferential statistics is based on the assumption of randomness as well as modern cryptography and computer simulations. Moreover, randomness can be used as an effective strategy for designing better algorithms. This is achieved by the deliberate introduction of randomness into computations (Motwani and Raghavan, 1995). Randomized algorithms have not only shown to outperform some of the best deterministic methods, they also have enabled the computation of previously infeasible problems. The Monte Carlo method, invented by Stan Ulam, John von Neumann and Nick Metropolis, is certainly one of the most prominent randomized methods in computational statistics as well as one of the 'best' algorithms of the 20th century (Cipra, 2000). In the area of optimization, a prominent example of a randomized algorithm is the stochastic gradient descent method, which is indispensable for large-scale machine learning (Bottou, 2010). Further, randomness plays an important role in neural networks, a

powerful learning technique in the area of machine learning (Scardapane and Wang, 2017).

Over the past two decades, randomness has also become popular as a computational strategy to compute low-rank approximations (see Section 1.3). In particular, modern data analysis and scientific computing largely relies on low-rank approximations, since low-rank matrices are ubiquitous throughout the social, physical, and biological sciences. However, in the era of 'big data', the emergence of massive data poses a significant computational challenge for traditional (deterministic) algorithms. This has forced a shift towards modern computational concepts to face these challenges. The basic idea of these randomized algorithms is to employ a degree of randomness as part of the logic in order to derive a smaller matrix from a high-dimensional input matrix, which captures the essential information of the original data matrix. This smaller matrix is then used to compute a near-optimal low-rank approximation, using a standard deterministic matrix factorization algorithm. The principal concept is sketched in Figure 1.1. The ideas behind this approach, are

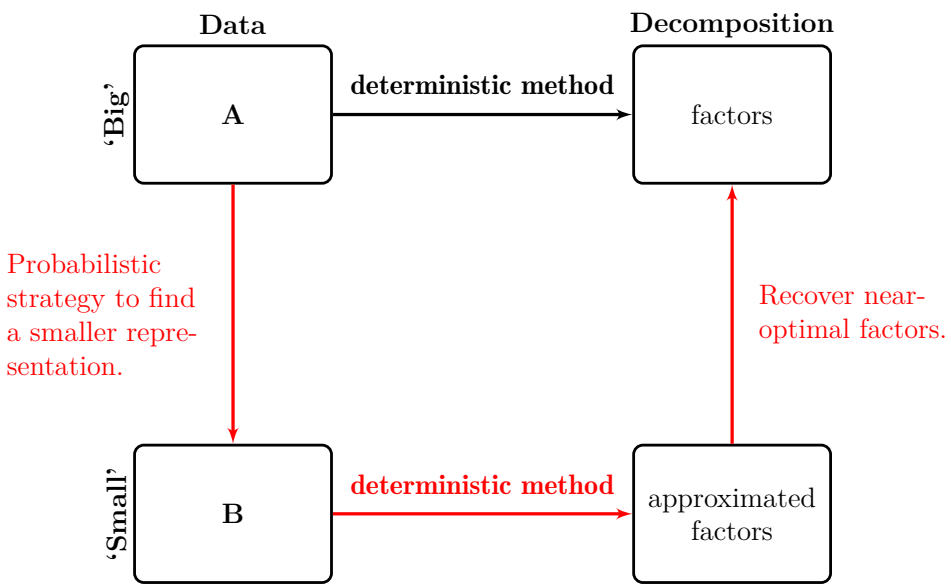


Fig. 1.1 First, randomness is used as computational strategy to derive a smaller matrix **B** from **A**. This smaller matrix can then be used to compute an approximate matrix decomposition. Finally, the near-optimal (high-dimensional) factors may be reconstructed.

originated in both theoretical computer science and applied mathematics. Initially, the focus of the early research was on computing the near-optimal low-rank singular value decomposition (SVD), which is likely to be one of the most important algorithms in machine learning, signal processing, and statistical computing. More recently, the field around randomized algorithms for low-rank matrix and data approximations has evolved into the broader field of randomized numerical linear algebra (RNLA). This is now an intense area of research, and comprehensive surveys of the field of research are provided by Mahoney (2011), Drineas and Mahoney (2016), Woodruff (2014), Halko et al. (2011b) and Martinsson (2016).

1.2 Some Notation and Preliminaries

In the following we give a brief overview of some notations used throughout this thesis. We have tried to remain as consistent as possible with terminology used in the area of matrix decompositions.

Scalars are denoted by lower case letters x, and vectors both in $\mathbb{R}^n$ and $\mathbb{C}^n$ are denoted as bold lower case letters $\mathbf{x} = [x_1, x_2, ..., x_n]$. The standard norm we use for vectors is the Euclidean norm

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{j=1}^{n} |x_j|^2},$$

denoted by the subscript 2. Both real $\mathbb{R}^{m\times n}$ and complex $\mathbb{C}^{m\times n}$ matrices are denoted by bold capitals $\mathbf{A}$, and its entry at row i^{th} and column j^{th} is denoted as A_{ij}. Often it is more convenient to refer to entries using the following notation $\mathbf{A}(i, j)$. This is, because this notation is convenient for matrix slicing, for instance, $\mathbf{A}(1 : i, :)$ extracts the first $1, 2, ..., i$ rows, and $\mathbf{A}(:, 1 : j)$ extracts the first $1, 2, ..., j$ columns. The transpose of a real matrix is denoted as $\mathbf{A}^\top$. More generally, the Hermitian transpose of a complex matrix is denoted as $\mathbf{A}^*$. Without loss of generality, we restrict most of the discussion in the following to real matrices. The spectral or operator norm of a matrix is defined as the largest singular value σ_{max} of $\mathbf{A}$, i.e., the square root of the largest eigenvalue λ_{max} of the

positive-semidefinite matrix $\mathbf{A}^\top\mathbf{A}$

$$\|\mathbf{A}\|_2 = \sqrt{\lambda_{max}(\mathbf{A}^\top\mathbf{A})} = \sigma_{max}(A) = \max_{x\neq 0} \frac{\|\mathbf{A}\mathbf{x}\|_2}{\|\mathbf{x}\|_2}.$$

The Frobenius norm is defined as the square root of the sum of the absolute squares of its elements, which is equal to the square root of the matrix trace of $\mathbf{A}^\top\mathbf{A}$

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |A_{ij}|^2} = \sqrt{\mathrm{trace}(\mathbf{A}^\top\mathbf{A})},$$

denoted by the subscript F. The column space (range) of $\mathbf{A}$ is denoted as $\mathrm{col}(\mathbf{A})$, and the row space as $\mathrm{row}(\mathbf{A})$. Further, $\mathsf{E}[\cdot]$ denotes the expected value of a random variable.

1.3 Low-Rank Approximations

Assume that a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ has intrinsic rank r. Then, in general, the objective of a low-rank matrix approximation is to find two smaller matrices

$$\underset{m\times n}{\mathbf{A}} \approx \underset{m\times r}{\mathbf{E}} \underset{r\times n}{\mathbf{F}}, \tag{1.1}$$

such that the matrix $\mathbf{E} \in \mathbb{R}^{m\times r}$ span the column space, and the rows of the matrix $\mathbf{F} \in \mathbb{R}^{r\times n}$ span the row space of $\mathbf{A}$. The factors $\mathbf{E}$ and $\mathbf{F}$ can then be used to summarize or to reveal some interesting structure in the data. Further, the factors can be used to efficiently store the large data matrix $\mathbf{A}$. Specifically, while $\mathbf{A}$ requires mn words of storage, $\mathbf{E}$ and $\mathbf{F}$ require only $mr + nr$ words of storage.

In practice most data matrices do not feature a precise rank r. Rather we are commonly interested in finding a rank k matrix $\mathbf{A}_k \in \mathbb{R}^{m\times n}$ which is as close as possible to an arbitrary input matrix $\mathbf{A}$ in the least-square sense. We refer to k as the target rank in the following. More formally, this can be expressed as a minimization problem, where the cost function measures the fit between $\mathbf{A}$ and $\mathbf{A}_k$, subject to the constraint that the

approximating matrix $\mathbf{A}_k$ has reduced rank k

$$\mathbf{A}_k := \underset{\mathbf{A}'_k : rank(\mathbf{A}'_k) \leq k}{\operatorname{argmin}} \|\mathbf{A} - \mathbf{A}'_k\|_2. \tag{1.2}$$

The Eckart-Young theorem (Eckart and Young, 1936) states that the low-rank singular value decomposition provides the optimal rank-k reconstruction of a matrix in the least-square sense, both in the spectral and Frobenius norms

$$\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top := \underset{\mathbf{A}'_k : rank(\mathbf{A}'_k) \leq k}{\operatorname{argmin}} \|\mathbf{A} - \mathbf{A}'_k\|. \tag{1.3}$$

Here, $\mathbf{U}_k$ and $\mathbf{V}_k$ are orthonormal matrices whose columns are the first k left and right singular vectors of $\mathbf{A}$, and $\mathbf{\Sigma}_k$ is a diagonal matrix containing the singular values in descending order. The reconstruction error in both the spectral and Frobenius norms is given by

$$\|\mathbf{A} - \mathbf{A}_k\|_2 = \sigma_{k+1} \quad \text{and} \quad \|\mathbf{A} - \mathbf{A}_k\|_F = \sqrt{\sum_{j=k+1}^{\min(m,n)} \sigma_j^2}. \tag{1.4}$$

Now, from this follows, that the best rank k approximation $\mathbf{A}_k$ can be found by projecting $\mathbf{A}$ onto the span of its top k left singular vectors

$$\mathbf{A}_k = \mathbf{U}_k \mathbf{U}_k^\top \mathbf{A}. \tag{1.5}$$

However, finding the best orthonormal basis $\mathbf{U}_k$ becomes computationally expensive with increasing dimensions of the input matrix. Instead, of using a deterministic algorithm to construct the orthonormal basis, it has been shown that the near-optimal orthonormal basis $\mathbf{Q} \in \mathbb{R}^{m \times k}$ can be efficiently constructed using randomized algorithms, while satisfying

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^\top \mathbf{A}\|_2 \leq (1 + \epsilon)\|\mathbf{A} - \mathbf{U}_k \mathbf{U}_k^\top \mathbf{A}\|_2, \tag{1.6}$$

where ϵ denotes some error.

1.4 Probabilistic Framework

In the following, we advocate the probabilistic framework formulated by (Halko et al., 2011b) in order to compute a near-optimal low-rank approximation. Specifically, this framework splits the computational task into two logical stages:

- **Stage A:** Construct a low dimensional subspace that approximates the range of $\mathbf{A} \in \mathbb{R}^{m \times n}$. Specifically, it is the aim to find a matrix $\mathbf{Q} \in \mathbb{R}^{m \times k}$ with orthonormal columns, where k denotes the target rank, such that $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^{\intercal}\mathbf{A}$ is satisfied.

- **Stage B:** Form a smaller matrix $\mathbf{B} \in \mathbb{R}^{k \times n}$ by restricting the high-dimensional input matrix to the low-dimensional space spanned by the near-optimal basis $\mathbf{Q}$. This smaller matrix can then be used to compute a desired low-rank approximation.

The first computational stage is where randomness comes actually into the play, while the second stage is purely deterministic. Let us consider the two stages in somewhat more detail, now.

Stage A: Computing a Near-Optimal Basis

Recall, the aim is to find a near-optimal basis $\mathbf{Q} \in \mathbb{R}^{m \times k}$ for the input matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ such that

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^{\top}\mathbf{A} \tag{1.7}$$

is satisfied. The desired target rank is denoted as k, and is assumed to be $k \ll (m, n)$. Specifically, $\mathbf{P} = \mathbf{Q}\mathbf{Q}^{\top}$ is a (linear) orthogonal projector. A projection operator corresponds to a linear subspace, and transforms any vector to its orthogonal projection on the subspace. This is illustrated in Figure 1.2, where a vector $\mathbf{x}$ is confined to the column space $\mathrm{col}(\mathbf{A})$.

Now, in order to efficiently construct such a orthogonal projector, we use the concept of random projections to sample the range (column space) of the input matrix $\mathbf{A}$. Specifically, a set of k random vectors $\{\omega_i\}_{i=1,\ldots,k}$ is drawn from a sub-Gaussian distribution. Now, note that probability theory guarantees that random vectors are linearly independent with high

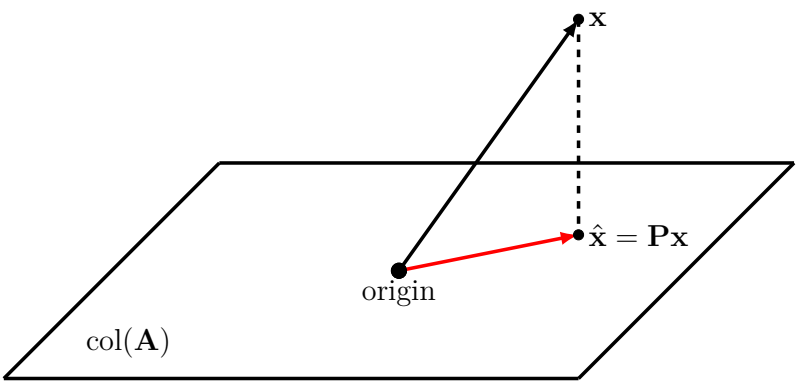


Fig. 1.2 Geometric illustration of the orthogonal projection operator $\mathbf{P}$. A vector $\mathbf{x} \in \mathbb{R}^m$ is restricted to the range of $\mathbf{A}$, where $\mathbf{P}\mathbf{x} \in \mathrm{col}(\mathbf{A})$.

probability. Hence, a new set of random projections $\{\mathbf{y}_i\}_{i=1,\ldots,k}$ can be computed to quickly sample the range of $\mathbf{A}$ as

$$\mathbf{y}_i = \mathbf{A}\omega_i \quad \text{for } i{=}1{,}2{,}\ldots{,}k\ . \tag{1.8}$$

Equation (1.8) can be efficiently executed in parallel as well as using matrix operations. Therefore, let us define the random test matrix $\mathbf{\Omega} \in \mathbb{R}^{m\times k}$, which is generated again from a sub-Gaussian distribution. Then, the sampling matrix $\mathbf{Y} \in \mathbb{R}^{m\times k}$ is simply obtained by post-multiplying the input matrix by the test matrix as

$$\mathbf{Y} = \mathbf{A}\mathbf{\Omega}. \tag{1.9}$$

Once $\mathbf{Y}$ is obtained, we only need to orthonormalize the columns in order to form a natural basis $\mathbf{Q} \in \mathbb{R}^{m\times k}$. This can be efficiently achieved using the QR-decomposition $\mathbf{Y} = \mathbf{Q}\mathbf{R}$. Then, it follows that Equation (1.7) is satisfied.

Stage B: Compute a Smaller Matrix

Now, given the near-optimal basis $\mathbf{Q}$, we aim to find a smaller matrix $\mathbf{B} \in \mathbb{R}^{k\times n}$. Therefor, we simply project the high-dimensional input matrix $\mathbf{A}$ to the low-dimensional space as

$$\mathbf{B} = \mathbf{Q}^\top \mathbf{A}. \tag{1.10}$$

Geometrically, this is a projection (i.e, a linear transformation) which takes points in a high-dimensional space into corresponding points in a low-dimensional space, illustrated in Figure 1.3. This process preserves

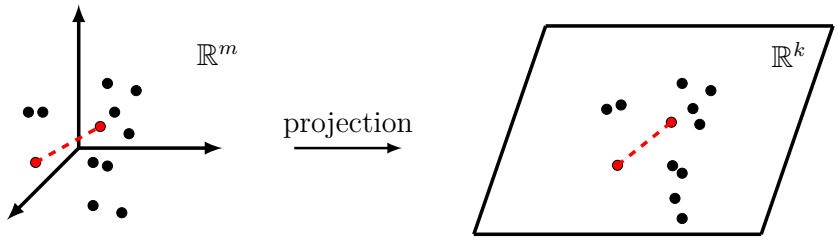


Fig. 1.3 Points in a high-dimensional space are projected into low-dimensional space, while the geometric structure is preserved in an Euclidean sense.

the geometric structure in an Euclidean sense, because inner products are preserved (Trefethen and Bau III, 1997). Further, due to the invariance of inner products the angles between vectors are preserved as well as their length $|\mathbf{Qa}|_2 = |\mathbf{a}|_2$. Substituting Equation (1.10) into (1.7) yields then the following low-rank approximation

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{Q}} \underset{k \times n}{\mathbf{B}}. \tag{1.11}$$

This decomposition is also referred to as the QB decomposition in the following. Subsequently, the smaller matrix $\mathbf{B}$ can be used to approximately compute a matrix decomposition using a traditional deterministic algorithm.

1.4.1 Computational Considerations

The outlined probabilistic framework can be used as a computational strategy to tackle the challenge of approximating massive matrices. The decomposition in Equation (1.11) has a time complexity of about $O(mnk)$. However, more importantly, the outlined procedure requires only two passes over the input matrix $\mathbf{A}$. By passes we refer to the number of sequential reads of the entire input matrix. This aspect is, in particular, crucial in the area of 'big data'. Specifically, we often face massive data

matrices which are to big to fit into fast memory, and the time to read the data from the hard-drive into the fast memory can be substantially more expensive then the theoretical costs of the actual algorithm would suggest. Hence, the probabilistic framework has a fundamental advantage in practice. Recently, Tropp et al. (2016) have introduced an interesting set of new single pass algorithms, reducing the communication costs even further.

Moreover, the computational steps required to compute the QB decomposition are simple to implement in any programing language which provides some numerical linear algebra routines. The procedure is robust, and allows by design to exploit modern computational architectures, e.g., multithreading or parallelized and distributed computing. It is also interesting to note that the actual quality of the random numbers does not play a significant role as long as the pseudo-random numbers appear to be 'random enough' in a certain sense (Halko et al., 2011b). Thus, the randomized algorithms described here are more reliable than some other Monte-Carlo methods.

In addition, the approximation error $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^\top \mathbf{A}\|_2$ can be controlled by introducing the two computational improvements as outlined in the following.

Oversampling.

Most real data matrices do not feature exact rank, which means that the singular values $\{\sigma_i\}_{i=k+1}^n$ of the input matrix $\mathbf{A}$ are non-zero. As a consequence the sampled matrix $\mathbf{Y}$ is likely to not span a good basis for the column space. However, this issue can be overcome by oversampling, i.e., using $l = k + p$ samples instead of just k in order to obtain the sample matrix $\mathbf{Y}$. Here, p denotes the number of additional samples and in most situations small values $p = \{5, 10\}$ are sufficient to obtain a good basis that is comparable to the best possible basis (Martinsson, 2016).

The intuition behind the oversampling scheme is the following. The sample matrix $\mathbf{Y}$ is a random variable, as it depends on the drawing of a random test matrix $\mathbf{\Omega}$. Specifically, oversampling decreases the variation in the singular value spectrum of the random test matrix, and subsequently improves the quality of the sample matrix. This is illustrated in Figure 1.4,

which shows the distribution of the singular value spectrum of a Gaussian test matrix $\mathbf{\Omega} \in \mathbb{R}^{100\times100}$. Now oversampling, i.e., stacking additional columns to the test matrix, substantially decreases the variation of the inverse of the smallest singular values.

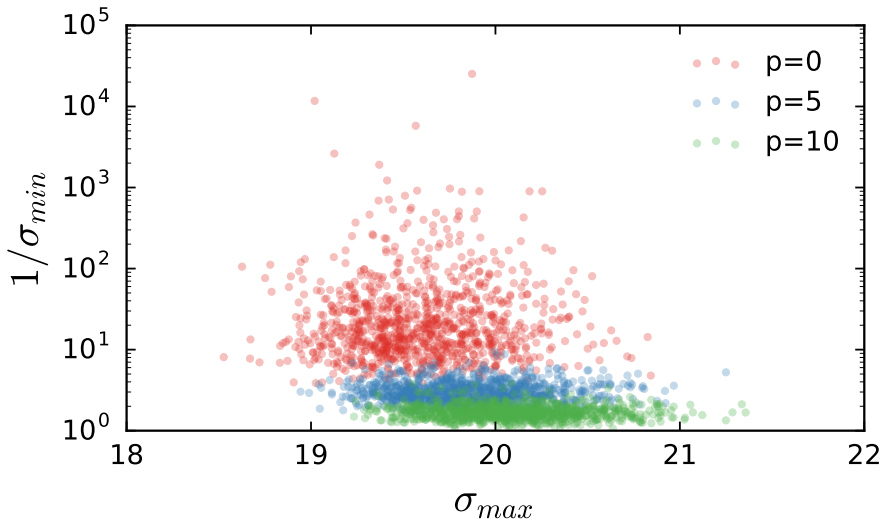


Fig. 1.4 Singular value spectrum of 1000 random Gaussian matrices of dimension 100×100. Adding random columns decreases the variation in the singular value spectrum.

Power Scheme

The second method to improve the quality of the basis $\mathbf{Q}$ involves the use of power sampling iterations (Gu, 2015; Halko et al., 2011b; Rokhlin et al., 2009). Instead of obtaining the sampling matrix $\mathbf{Y}$ directly, the data matrix $\mathbf{A}$ is first preprocessed as

$$\mathbf{A}^{(q)} = (\mathbf{A}\mathbf{A}^\top)^q\mathbf{A}, \tag{1.12}$$

where q is an integer specifying the number of power iterations. Let, $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, then it is simple to show that $\mathbf{A}^{(q)} = (\mathbf{A}\mathbf{A}^\top)^q\mathbf{A} = \mathbf{U}\mathbf{\Sigma}^{2q+1}\mathbf{V}^\top$. Here, $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices whose columns are the left, and right singular vectors of $\mathbf{A}$, and $\mathbf{\Sigma}$ is a diagonal matrix containing the singular values in descending order. Hence, for $q > 0$, the modified matrix $\mathbf{A}^{(q)}$ has a relatively fast decay of singular values even when the decay

in $\mathbf{A}$ is modest. This is illustrated in Figure 1.5, showing the singular values of a 50×50 matrix before (red) and after computing $q = \{1, 2, 3\}$ power iterations. Now, substituting Equation (1.12) into (1.9) yields an

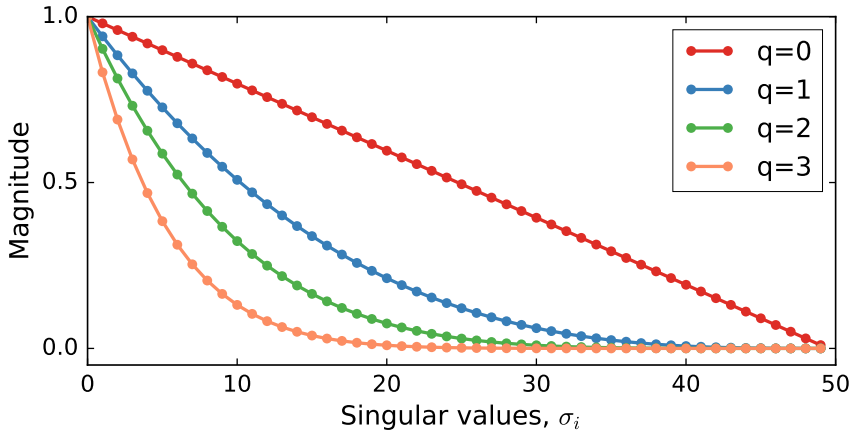


Fig. 1.5 Singular value spectrum of a matrix before and after the computation of power iterations.

improved sampling matrix

$$\mathbf{Y} = \mathbf{A}^{(q)}\mathbf{\Omega} = \Big((\mathbf{A}\mathbf{A}^\top)^q\mathbf{A}\Big)\mathbf{\Omega}. \tag{1.13}$$

The drawback, of course, is that q additional passes over the input matrix are required. However, when the singular values of the data matrix decay slowly, about $q = \{1, 2\}$ power iterations can considerably improve the approximation. In a practical implementation subspace iterations are used instead of power iterations for numerical stability. Some implementation details are discussed in more detail in Section 2.2.

1.4.2 Theoretical Performance

Both the concept of oversampling and the power scheme allow to control the quality of low-rank approximation. Martinsson (2016) provides the following description of the average case behavior of the outlined

probabilistic framework: [1]

$$\mathsf{E}\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^\top \mathbf{A}\|_2 \leq \left[1 + \sqrt{\frac{k}{p-1}} + \frac{e\sqrt{l}}{p} \cdot \sqrt{\min\{m,n\} - k}\right]^{\frac{1}{2q+1}} \sigma_{k+1}(\mathbf{A}).$$

Here it is assumed that the oversampling parameter $p \geq 2$. The operator E denotes the expectation with respect to a Gaussian test matrix $\mathbf{\Omega}$, and $\sigma_{k+1}(\mathbf{A})$ denotes the smallest possible error achievable with any basis matrix $\mathbf{Q}$. Thus, both oversampling and the computation of additional power iterations drive the approximation error down.

1.4.3 Test Matrices

An essential computational step of the above described probabilistic framework is the construction of the random test matrix $\mathbf{\Omega}$. Specifically, we seek a test matrix which insures that its randomly generated columns are linearly independent with high probability.

Dense Random Test Matrices

The most prominent choice is to construct a test matrix with independent identically distributed (i.i.d.) standard normal entries $\mathcal{N}(0,1)$. This is mainly due to the attractive theoretical properties of the Gaussian distribution. In practice, uniform random measurements are sufficient, while less expensive to generate. Thus, for the practical implementation of the algorithms we favor random test matrices with i.i.d. standard uniform entries $\mathcal{U}(-1,1)$.

The drawback is that matrix operations are becoming increasingly expensive for large dense matrices. However, BLAS (Basic Linear Algebra Subprograms) operations tend to be highly scalable, and computations can be substantially accelerated using parallel computing. In Chapter 5 we describe how graphics processing units (GPUs) can be utilized for this.

[1] Note, that this is a simplified version of one of the key theorems presented by Halko et al. (2011b), who provide a detailed error analysis of the outlined probabilistic framework.

Structured Random Test Matrices

Woolfe et al. (2008) proposed a more computationally efficient approach, exploiting the properties of structured random matrices. Specifically, it was shown that the computational costs can be reduced from $O(mnk)$ to $O(mn\log(k))$ using a subsampled random Fourier transform (SRFT) test matrix

$$\mathbf{\Omega} = \mathbf{RFD} \tag{1.14}$$

were $\mathbf{R} \in \mathbb{C}^{k\times n}$ draws k random rows (without replacement) from the identity matrix $\mathbf{I} \in \mathbb{C}^{n\times n}$. $\mathbf{F} \in \mathbb{C}^{n\times n}$ is the unnormalized discrete Fourier transform with the following entries $\mathbf{F}(j,v) = exp(-2\pi i(j-1)(v-1)/m)$ and $\mathbf{D} \in \mathbb{C}^{n\times n}$ is a diagonal matrix with independent random diagonal elements uniformly distributed on the complex unit circle. While the SRFT sensing matrix comes with nice theoretical properties, the improvement from $O(k)$ to $O(log(k))$ is not necessarily significant if the target rank k is small. Further, the efficient implementation of random structured test matrices is quite subtle in practice.

1.5 Overview and Contributions

As outlined, the efficient computation of low rank matrix approximations is an ubiquitous problem in machine learning and elsewhere. Massive data pose a computational challenge for traditional (deterministic) matrix algorithms, placing significant constraints on both memory and processing power. Recently, randomized matrix algorithms have been established as some of the most competitive methods to overcome some of these computational challenges. Randomized algorithms are robust, reliable and computationally efficient. Specifically, this thesis is based on the concepts of randomized matrix algorithms as introduced by Martinsson et al. (2011) and Halko et al. (2011b). The main contribution of this thesis is the development of new randomized algorithms, and their application for video processing and fluid dynamics as well as the generalization of the probabilistic framework to tensors. In the following we give a brief overview of each chapter and outline the contributions.

- **Chapter 2: Randomized Singular Value Decomposition.** This chapter reviews the concept of randomness as a computational strategy to accelerate the computations of the singular value decomposition (SVD) and principal component analysis (PCA). Further, it is demonstrated how the randomized SVD can be used to accelerate the computation of robust principal component analysis (RPCA). The main contribution of this Chapter is the corresponding R software package **rsvd**, which provides efficient implementations of the presented methods (Erichson, 2015) in R. The corresponding source code of the **rsvd** package is available via the GIT repository: https://github.com/Benli11/rSVD. Several numerical examples demonstrate the substantial computational savings by using the powerful concept of randomness. This includes examples of image compression, eigenfaces, and foreground/background decomposition as well as the evaluation of the computational performance on both dense and sparse matrices.

  This chapter is based on the paper by Erichson et al. (2016c) (under review with the Journal of Statistical Software), which is written in joint work with Sergey Voronin, Steven L. Brunton and J. Nathan Kutz.

- **Chapter 3: Randomized Dynamic Mode Decomposition.** The dynamic mode decomposition (DMD) is a modern dimensionality reduction technique designed to extract dynamic information from dynamical systems based on a sequence of snapshots (time series of data). However, the dynamic mode decomposition is computationally demanding, in particular, for high-dimensional data such as fluid flows or video feeds. This chapter presents a novel randomized algorithm for computing the dynamic mode decomposition. We show that the randomized accelerated algorithm can substantially reduce the computational costs, while being robust and obtaining near optimal decomposition results. The algorithm is compared with the deterministic algorithm to compute the DMD as well as with the previously introduced compressed DMD algorithm.

The accompanying software package **DMDpack** provides implementations of the outlined algorithms in Python. The source code is available via the GIT repository: https://github.com/Benli11/DMDpack.

This chapter is based on the paper by Erichson et al. (2016a) and Erichson et al. (2017) which are written in joint work with Steven L. Brunton and J. Nathan Kutz.

- **Chapter 4: Dynamic Mode Decomposition for Background Modeling.**
  Background modeling for foreground detection is one of the fundamental computational steps in many computer vision applications. While there are numerous methods for modeling the background of surveillance videos, a great demand remains for high-performance algorithms capable of real-time processing. In particular, the rapidly increasing resolution of sensors pose a computational burden for several traditional techniques. Here, we advocate the method of randomized dynamic mode decomposition (rDMD) for background modeling. While the principal application of the DMD is in the area of fluid dynamics, the method has been successfully used for background modeling of surveillance videos previously. We present several numerical results using standard benchmark datasets including synthetic and real surveillance video feeds.

  This chapter is based on the paper by Erichson and Donovan (2016) which is written in joint work with Carl Donovan as well as on the paper by Erichson et al. (2016a) which is written in joint work with Steven L. Brunton and J. Nathan Kutz.

- **Chapter 5: GPU Accelerated Randomized Algorithms.**
  The outlined probabilistic framework for computing low-rank matrix approximations is embarrassingly parallel. Here, we utilize graphics processing units (GPUs) to accelerate the computations. GPUs becoming increasingly popular for general-purpose high-performance computing. Specifically, we demonstrate the advantage of parallel computing for obtaining the singular value and dynamic mode decomposition.

The GPU accelerated algorithms are implementations as contribution to the Python software package **scikit-cuda**, which is authored and maintained by Lev E. Givon. Specifically, we have provided routines to compute the deterministic, randomized and compressed DMD as well as the randomized SVD. Further contributions are routines for computing the eigendecomposition, the QR decomposition, and a function for constructing the Vandermonde matrix in GPU memory. The source code is available via the GIT repository: https://github.com/lebedov/scikit-cuda.

- **Chapter 6: Randomized CP Decomposition**
  Classical matrix factorizations can become inadequate when dealing with tensors. This is, because reshaping multi-modal data into matrices can fail to reveal important structures in the data. Tensor decompositions can overcome this issue of information loss. The CANDECOMP/PARAFAC (CP) decomposition is particularly suitable for data-driven discovery since it expresses a tensor as a sum of rank-one tensors. This chapter presents a novel randomized algorithm for computing the CP Decomposition. In particular the concept of power iterations is crucial in order to achieve a near-optimal decomposition quality. Theorem 1 characterizes the average-case behavior of the randomized tensor algorithm. The proof is sketched in Appendix A. Further, the accompanying Python software package **ctensor** provides implementations of the standard and randomized CP decomposition using different optimization strategies. The corresponding source code is available via the GIT repository: https://github.com/Benli11/ctensor. The performance of this highly efficient algorithm is demonstrated using artificial and real world data.

  This chapter is based on the paper by Erichson et al. (2016b) (under review) which is written in joint work with Krithika Manohar, Steven L. Brunton and J. Nathan Kutz.

Chapter 2

Randomized Singular Value Decomposition

"Begin with the simplest examples."

— David Hilbert

Note: The work described in this chapter was carried out in collaboration with Professors J. Nathan Kutz and Steven L. Brunton of University of Washington and Dr. Sergey Veronin of the Tufts University. It is submitted to the Journal of Statistical Software under the title: 'Randomized Matrix Decompositions using R'. My contributions involve conceptualizing the project idea, implementing the routines, running the simulations as well as writing the original draft.

2.1 Introduction

Matrix decompositions are fundamental mathematical tools, extensively used in the area of machine learning, statistical computing, computer vision and engineering. The singular value decomposition is among the most ubiquitous and powerful methods for linear dimensionality reduction and data processing in the computational era. Indeed, it is the workhorse algorithm behind principal component analysis, linear discriminant analysis, and canonical correlation analysis. However, the emergence of 'big data' has severely challenged our ability to compute the singular value decomposition, placing significant constraints on both memory and processing power. At the same time, methods for dimensionality reduction are becoming increasingly important in order to deal with high-dimensional

data produced by modern sensors or social networks. Interestingly though, many high-dimensional signals have low intrinsic rank relative to the dimension of the ambient measurement space. This means these data carry redundant information, which are unnecessary to understand, or describe an underlying process. Prominent examples of high-dimensional data featuring a natural low-rank structure are, for instance, structured signals like audio, images or video footage (Davenport and Romberg, 2016). Recently, it has been impressively demonstrated that randomization can be utilized as a computational strategy to substantially ease the logistic and computational challenges involved in obtaining an approximate low-rank singular value decomposition. This is interesting, in particular, for massive matrices where traditional deterministic algorithms fail.

Motivation and Overview

In the following we demonstrate how the singular value decomposition and the principal component analysis can be embedded into the probabilistic framework presented in Chapter 1, Section 1.4. While, this has been discussed in great detail in theory and applications elsewhere, our motivation and contribution was the development of the R software package **rsvd**. This package implements the methods discussed in the following, and is as far as we are aware the first R package providing these randomized routines. Section 2.2 briefly reviews the singular value decomposition, followed by presenting a randomized algorithm for computing the SVD. Section 2.3 reviews the principal component analysis, and establishes the connection to the SVD. Then, a fast randomized algorithm for computing the near optimal dominant principal components is shown. Section 2.4 discusses how the randomized singular value decomposition can be used to accelerate the computation of the robust principal component analysis. The interface of the **rsvd** package and some implementation details are presented in Section 2.5. Section 2.6 presents numerical results using the **rsvd** package. Examples include image compression, eigenfaces, and foreground/background decomposition of video footage. Then, the computational performance is evaluated for both dense and sparse matrices. Finally, concluding remarks and a roadmap for further developments of the **rsvd** package are outlined in Section 2.7.

2.2 Singular Value Decomposition

The SVD provides a numerically stable matrix decomposition that can be used to obtain low-rank approximations, to compute the pseudo-inverses of non-square matrices, and to find the least-squares and minimum norm solutions of a linear model. For a comprehensive technical overview we refer to Golub and Van Loan (1996), Demmel (1997), and Watkins (2002).

2.2.1 Brief Historical Overview

While the origins of the SVD can be traced back to the late 19th century, the field of randomized matrix algorithms is relatively young. Figure 2.1 shows an incomplete time-line of some major developments of the singular value decomposition. Stewart (1993) gives an excellent historical review of the five mathematicians who developed the fundamentals of the SVD, namely Eugenio Beltrami (1835-1899), Camille Jordan (1838-1921), James Joseph Sylvester (1814-1897), Erhard Schmidt (1876-1959) and Hermann Weyl (1885-1955). The development and fundamentals of modern high-performance algorithms to compute the SVD is related to the seminal work of Golub and Kahan (1965) and Golub and Reinsch (1970).

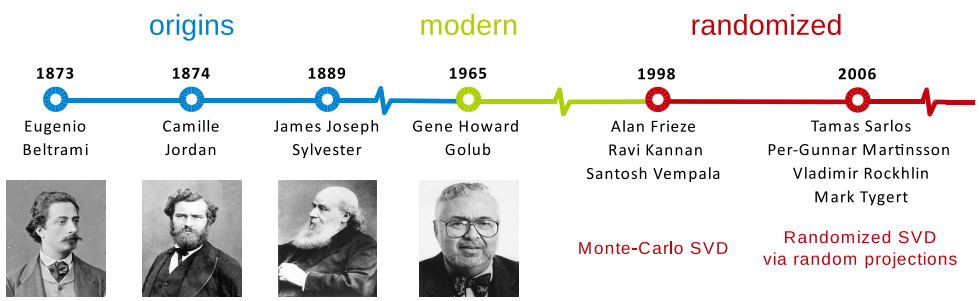


Fig. 2.1 A timeline of major singular value decomposition developments.

Modern singular value decomposition algorithms are largely based on Krylov methods. These methods are accurate and in particular powerful for approximating structured or sparse large-scale matrices (Martinsson, 2016). A modern and prominent state-of-the-art algorithm, based on the Lanczos algorithm, is the **PROPACK** SVD algorithm (Larsen, 1998). As well as the partial SVD (**svds**) algorithm based on the **ARPACK** software

package (Lehoucq et al., 1998), used in MATLAB or Python for sparse matrices.

Randomized matrix algorithms for computing low-rank matrix approximations have gained prominence over the past two decades. Frieze et al. (2004) introduced the 'Monte Carlo' SVD, a rigorous approach to efficiently compute the approximate low-rank SVD based on non-uniform row and column sampling. Sarlos (2006) and Martinsson et al. (2011) introduced a more robust approach based on random projections. Specifically, the properties of random vectors are exploited to efficiently build a subspace that captures the column space of a matrix. Woolfe et al. (2008) further improved the computational performance by leveraging the properties of highly structured matrices which enable fast matrix multiplications. Eventually, the seminal work by Halko et al. (2011b) unified and expanded the work on the randomized singular value decomposition (rSVD) and introduced state-of-the-art prototype algorithms to compute the near-optimal low-rank singular value decomposition.

2.2.2 Conceptual Overview

Given an arbitrary real [1] matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, where $m \geq n$ without loss of generality, we seek a decomposition, such that

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\top}. \tag{2.1}$$

The matrices $\mathbf{U} = [\mathbf{u}_1, ..., \mathbf{u}_m] \in \mathbb{R}^{m \times m}$ and $\mathbf{V} = [\mathbf{v}_1, ..., \mathbf{v}_n] \in \mathbb{R}^{n \times n}$ are orthonormal so that $\mathbf{U}^{\top}\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^{\top}\mathbf{V} = \mathbf{I}$. The first r left singular vectors in $\mathbf{U}$ provide a basis for the range and the first r right singular vectors in $\mathbf{V}$ a basis for the domain of the matrix $\mathbf{A}$, whereby r denotes the rank of the data matrix. The rectangular diagonal matrix $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ contains the corresponding non-negative singular values $\sigma_1 \geq ... \geq \sigma_r$, describing the spectrum of the data. If the rank r of the matrix $\mathbf{X}$ is smaller then the number of columns (i.e., $r < n$), then the last $m - r$ singular values $\{\sigma_i : i \geq r + 1\}$ are zero. The so called 'economic' or 'compact' SVD computes only the singular vectors corresponding to the non-zero

[1] Without loss of generality, the concept applies to complex matrices using the Hermitian transpose instead.

singular values $\mathbf{U} = [\mathbf{u}_1, ..., \mathbf{u}_r] \in \mathbb{R}^{m \times r}$ and $\mathbf{V} = [\mathbf{v}_1, ..., \mathbf{v}_r] \in \mathbb{R}^{n \times r}$ respectively. In many cases, the numerical rank $\bar{r}$ of the matrix is smaller than it's mathematical rank. That is, many of the last $\min(m, n) - \bar{r}$ singular values can be close to machine precision. Since the corresponding singular vectors are not in the span of the data, it is often desirable to compute only a reduced version of the SVD. The matrix can be well approximated by including only those singular vectors which correspond to singular values of a significant magnitude. This number k can be much smaller than $\min(m, n)$ depending on the value of $\bar{r}$. Choosing an optimal target rank k is highly dependent on the task, i.e., whether one is interested in a highly accurate reconstruction of the original data or in a very low dimensional representation of dominant features in the data. The low-rank SVD of rank k takes the form:

$$\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k = [\mathbf{u}_1, ..., \mathbf{u}_k] \mathrm{diag}(\sigma_1, \ldots, \sigma_k) [\mathbf{v}_1, ..., \mathbf{v}_k]^\top. \tag{2.2}$$

Nevertheless, computing the low-rank singular value decomposition is computationally demanding, and massive data pose a computational challenge for traditional algorithms. Specifically, the computational cost of computing the truncated SVD $\mathbf{A}_k$ using a deterministic algorithm for an $m \times n$ matrix is of the order $O(mn^2)$, from which the first k components can then be extracted to form $\mathbf{A}_k$. Modern SVD algorithms are largely based on Krylov methods. These methods are accurate and in particular powerful for approximating structured or sparse large-scale matrices (Martinsson, 2016). A modern and prominent state-of-the-art algorithm, based on the Lanczos algorithm, is the **PROPACK** SVD algorithm (Larsen, 1998). As well as the partial SVD (**svds**) algorithm based on the **ARPACK** software package (Lehoucq et al., 1998), used in MATLAB or Python for sparse matrices. These algorithms can be used to obtain an approximate rank k singular value decomposition at a cost of $O(mnk)$. While the randomized algorithm for computing the low-rank SVD comes with the same theoretical costs, it has several practical advantages. In particular, the randomized algorithm is more robust, i.e., the algorithm can better cope with ill-condition matrices. Further, the

randomized algorithm is easier to implement and embarrassingly parallel compared to Krylov methods.

2.2.3 Randomized Algorithm

Assume that we seek the low-rank SVD for a data matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, where $m \geq n$ without loss of generality, such that

$$\mathbf{A} = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top, \tag{2.3}$$

where k denotes the target rank. Now, instead of computing the singular value decomposition directly, we embed the SVD into the probabilistic framework presented in Chapter 1, Section 1.4. The principal concept is sketched in Figure 2.2 and 2.3. Specifically, we first compute the near-

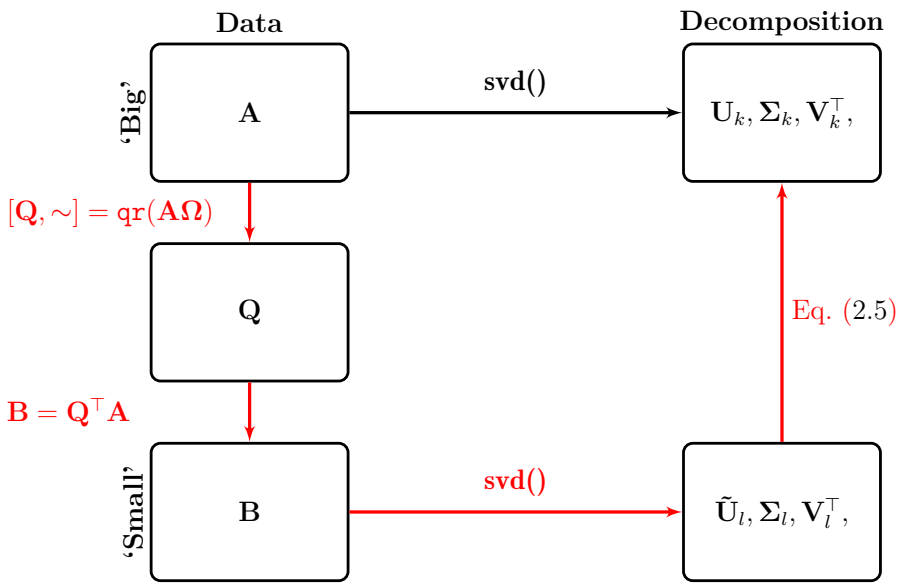


Fig. 2.2 Conceptual architecture of the randomized singular value decomposition. First, a natural basis is computed in order to derive the smaller matrix $\mathbf{B}$. Then, the SVD is efficiently computed using this smaller matrix. Finally, the left singular vectors $\mathbf{U}_k$ may be reconstructed from the approximate singular vectors $\tilde{\mathbf{U}}_l$ by the expression in Eq. (2.5).

optimal basis $\mathbf{Q} \in \mathbb{R}^{m \times l}$, and the relatively small (if $l \ll m, n$) matrix $\mathbf{B} \in \mathbb{R}^{l \times n}$. Note, that we allow for oversampling, i.e., $l = k + p$. Recall, we first generate a random test matrix $\mathbf{\Omega} \in \mathbb{R}^{n \times l}$ to sample the range

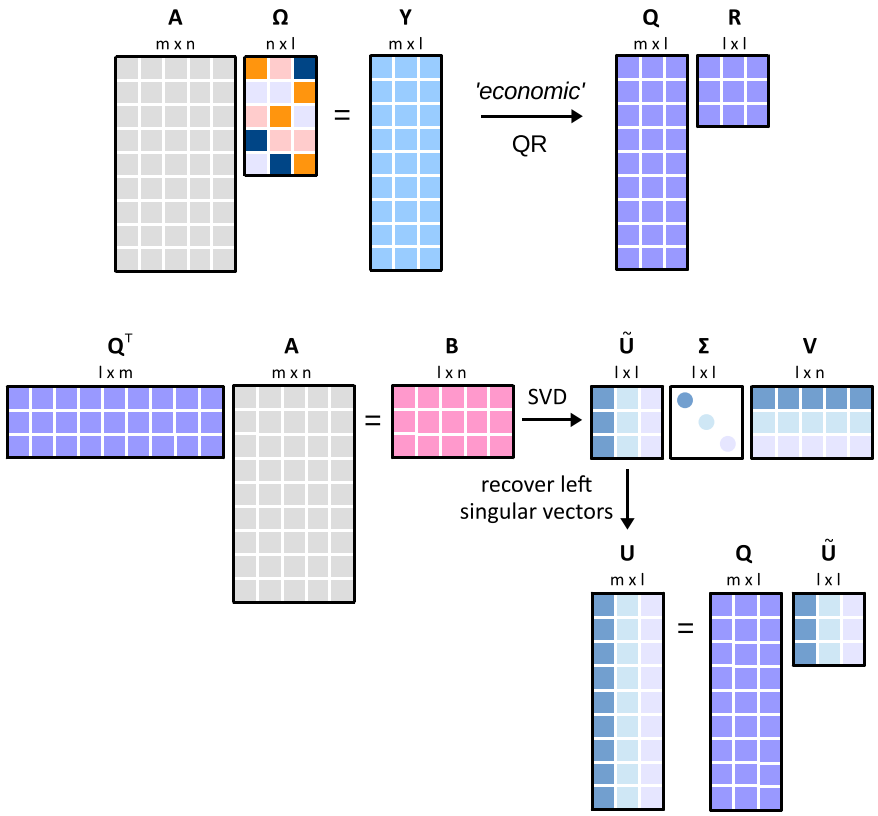


Fig. 2.3 Schematic architecture of the randomized singular value decomposition.

of the input matrix as $\mathbf{Y} = \mathbf{A\Omega}$. Then, orthonormalizing the columns of $\mathbf{Y}$ yields the natural basis. This can be efficiently achieved using the QR decomposition $\mathbf{Y} = \mathbf{QR}$. Next, the small matrix $\mathbf{B}$ is obtained by projecting the input matrix to the low-dimensional subspace as $\mathbf{B} = \mathbf{Q}^\top \mathbf{A}$. Once this is achieved, the approximate SVD is computed using a standard (deterministic) algorithm so that we attain the following decomposition

$$\mathbf{B} = \tilde{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^\top. \tag{2.4}$$

Thus, we efficiently obtain the first l right singular vectors $\mathbf{V} \in \mathbb{R}^{n\times l}$ as well as the corresponding singular values $\mathbf{\Sigma} \in \mathbb{R}^{l\times l}$. It then remains to recover the left singular vectors $\mathbf{U} \in \mathbb{R}^{m\times l}$ from the approximate left singular vectors $\tilde{\mathbf{U}} \in \mathbb{R}^{l\times l}$. This can be simply achieved by pre multiplying $\tilde{\mathbf{U}}$ by $\mathbf{Q}$ as follows

$$\mathbf{U} \approx \mathbf{Q}\tilde{\mathbf{U}}. \tag{2.5}$$

The justification for the randomized SVD can be sketched as follows

$$\begin{aligned} \mathbf{A} &\approx \mathbf{Q}\mathbf{Q}^\top\mathbf{A} \\ &\approx \mathbf{Q}\mathbf{B} \\ &\approx \mathbf{Q}\tilde{\mathbf{U}}\boldsymbol{\Sigma}\mathbf{V}^\top \\ &\approx \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top. \end{aligned} \tag{2.6}$$

Note, that if an oversampling parameter $p > 0$ has been specified, the desired rank k approximation is simply obtained by truncating the left and right singular vectors as well as the singular values.

Further, we have suggested that in addition to oversampling the accuracy of the natural basis $\mathbf{Q}$ can be improved by the use of power sampling iterations (Gu, 2015; Halko et al., 2011b; Rokhlin et al., 2009). Recall, the improved sampling matrix is computed as $\mathbf{Y} = \Big((\mathbf{A}\mathbf{A}^\top)^q\mathbf{A}\Big)\boldsymbol{\Omega}$. Now, this enforces a more rapid decay of singular values in the pre processed matrix sampled from. A direct implementation of the power iteration scheme could be implemented as follows

(1) $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$
(2) **for** $j = 1, \ldots, q$
(3) $\quad\mathbf{Y} = \mathbf{A}^\top\mathbf{Y}$
(4) $\quad\mathbf{Y} = \mathbf{A}\mathbf{Y}$
(5) **end for**

However, this direct implementation of the method is numerically unstable due to potential round-off errors. Instead, we improve the numerical stability of the procedure by orthogonalizing the sampling matrix in between each computational step. This leads to the following improved scheme of subspace iterations

(1) $\mathbf{Y} = \mathbf{A\Omega}$
(2) **for** $j = 1, \ldots, q$
(3) $\quad [\mathbf{Q}, \sim] = \mathtt{qr}(\mathbf{Y})$
(4) $\quad \mathbf{Z} = \mathbf{A}^\top \mathbf{Q}$
(5) $\quad [\mathbf{Q}, \sim] = \mathtt{qr}(\mathbf{Z})$
(6) $\quad \mathbf{Y} = \mathbf{AQ}$
(7) **end for**

Rokhlin et al. (2009) has proposed a slightly modified approach, which is avoiding the repeated computation of the QR decomposition. Here the pivoted LU decomposition is used instead, which is slightly less expensive to obtain. This scheme of normalized power iterations can be sketched as follows

(1) $\mathbf{Y} = \mathbf{A\Omega}$
(2) **for** $j = 1, \ldots, q$
(3) $\quad [\mathbf{L}, \sim] = \mathtt{lu}(\mathbf{Y})$
(4) $\quad \mathbf{Z} = \mathbf{A}^\top \mathbf{L}$
(5) $\quad [\mathbf{L}, \sim] = \mathtt{lu}(\mathbf{Z})$
(6) $\quad \mathbf{Y} = \mathbf{AL}$
(7) **end for**

Algorithm 1 presents an implementation which allows for oversampling as well as the computation of additional subspace iterations. In Algorithm 1 we compute the full SVD of $\mathbf{B}$, so it follows that $\|\mathbf{A} - \mathbf{QB}\| = \|\mathbf{A} - \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top\|$, and $\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top$. Martinsson (2016) presents the following simplified error bound quantifying the performance of the algorithm

$$\mathsf{E}\|\mathbf{A} - \mathbf{A}_k\|_2 \leq \left[1 + \sqrt{\frac{k}{p-1}} + \frac{e\sqrt{k+p}}{p} \cdot d\right]^{\frac{1}{2q+1}} \sigma_{k+1}, \tag{2.7}$$

where $d = \sqrt{\min(m,n) - k}$. Thus an increasing number of power iterations q drives the approximation error down, i.e., the bound approaches the theoretically optimal value of σ_{k+1} with increasing q.

Algorithm 1 A randomized SVD algorithm.

```
function [U, Σ, V] = rsvd(A, k, p, q)
    l = k + p                          slight oversampling
    Ω = rnorm(n, l)                    generate sampling matrix
    Y = AΩ                             draw range samples
    for j = 1, ..., q                  perform optional subspace iterations
        [Q, ~] = qr(Y)
        Z = A^T Q
        [Q, ~] = qr(Z)
        Y = AQ
    end for
    [Q, ~] = qr(Y)                     form orthonormal samples matrix
    B = Q^T A                          project to smaller space
    [Ũ, Σ, V] = svd(B)                 compact SVD of smaller matrix B
    U = QŨ                             recover left singular vectors
    U = U(:, 1 : k)                    extract k components
    Σ = Σ(1 : k, 1 : k)                extract k components
    V = V(:, 1 : k)                    extract k components
```

2.3 Principal Component Analysis

Originally formulated by Pearson (1901), principal component analysis (PCA)[2] still plays an important role in modern statistics due to its simple geometric interpretation. Specifically, it is widely used for feature extraction and visualization of big datasets comprising many interrelated variables. A classical statistical text on PCA is Jolliffe (2002), while more modern views and extensions are presented by Hastie et al. (2009), Murphy (2012) and Izenman (2008). Further, we want to point out the excellent review by Abdi and Williams (2010) and the recent seminal paper on linear dimensionality reduction by Cunningham and Ghahramani (2015).

[2] Also commonly known as Hotelling transform, Karhunen Loève, or proper orthogonal decomposition (POD).

2.3.1 Conceptual Overview

The essential idea of PCA is to find a new set of uncorrelated variables that retain most of the information (total variation) present in the data. Figure 2.4 illustrates this for a two-dimensional example. The upper plot shows some fairly correlated data. The arrows indicate the principal directions of the data. It can be seen that the first arrow points into the direction which explains most of the variance, while the second arrow is orthogonal to the first one. Together, they span a new coordinate system so that the first axis accounts for most of the variance and the second for the remaining variance in the data. The lower plot shows the original data in this new coordinate system, represented by a new set of uncorrelated variables the so called principal components. The values of these new variables are called principal component scores or coordinates. The histograms indicate that most information (variation) is now captured by just the first principal component (PC).

To be more formal, assume a data matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ with n observations and p variables (column-wise, mean-centered). Then, the principal components can be expressed as a linear combination

$$\mathbf{z}_i = \mathbf{X}\mathbf{w}_i, \tag{2.8}$$

where $\mathbf{z}_i \in \mathbb{R}^n$ denotes the i'th principal component and the i'th principal direction is represented by the vector $\mathbf{w}_i \in \mathbb{R}^p$, where the elements of $\mathbf{w}_i = (w_1, ..., w_p)^\top$ are the corresponding principal component coefficients or weights.

In summary, we seek that the first principal component explains most of the total variation in the data. Subsequent PCs are required to be orthogonal to the previous components, and capturing the remaining variance in descending order. Mathematically, this problem can be formulated either as a least square minimization or as a variance maximization problem (Cunningham and Ghahramani, 2015). The two views are illustrated in Figure 2.5. Geometrically, the relationship between the two approaches can be explained by utilizing the Pythagorean theorem (Jolliffe, 2002). Specifically, the total variance equals the sum of the explained and unexplained variance, as illustrated in Figure 2.6. Thus, both views are

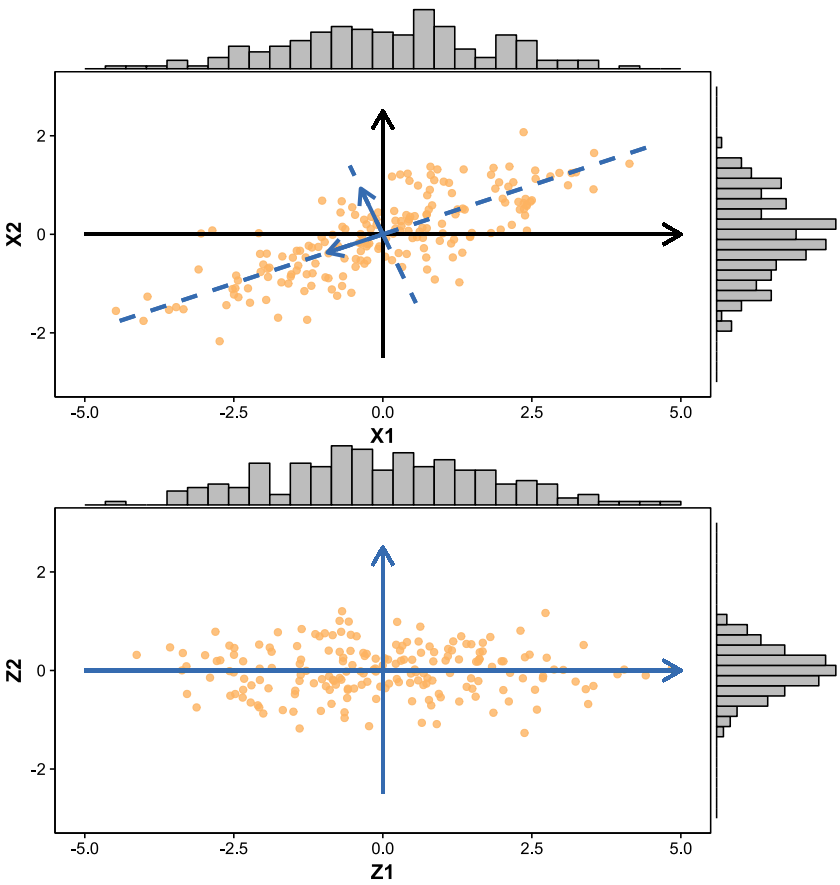


Fig. 2.4 Illustration of PCA seeking to find a new set of uncorrelated variables. The top plot shows the data and its two principal directions are shown. The bottom plot shows the new principal component variables, indicating that the first component accounts for most of the variation in the data.

equivalent and boil down to an eigenvalue problem (Murphy, 2012). Here, we follow the latter approach, i.e., maximizing the variance of the first principal component subject to the normalization constraint $\|\mathbf{w}\|_2^2 = 1$ as follows

$$\mathbf{w}_1 = \underset{\|\mathbf{w}\|_2^2=1}{\operatorname{argmax}} \, \mathsf{VAR}(\mathbf{X}\mathbf{w}) \propto \underset{\|\mathbf{w}\|_2^2=1}{\operatorname{argmax}} \, \|\mathbf{X}\mathbf{w}\|_2^2, \tag{2.9}$$

where VAR denotes the variance of a random variable. The last term can then be expanded as

$$\mathbf{w}_1 = \underset{\|\mathbf{w}\|_2^2=1}{\operatorname{argmax}} \, \mathbf{w}^\top (\mathbf{X}^\top \mathbf{X}) \mathbf{w}. \tag{2.10}$$

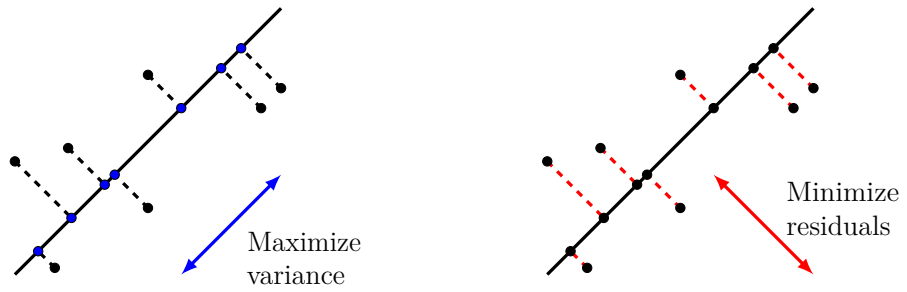


Fig. 2.5 Principal component analysis can be formulated either as a variance maximization or as a least square minimization problem. Both views are equivalent.

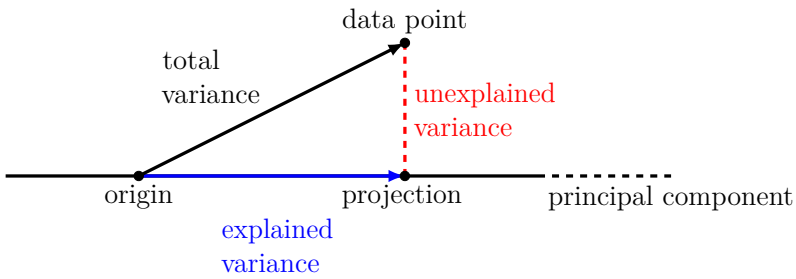


Fig. 2.6 Geometrical relationship between the two views of PCA. From the Pythagorean theorem it follows that the principal components can be obtained by either maximizing the variance or minimizing the unexplained variance (squared residuals).

Considering that we constrained $\|\mathbf{w}\|_2^2 = \mathbf{w}^\top \mathbf{w} = 1$ to be a unit vector, Eq. (2.10) can be rewritten as

$$\mathbf{w}_1 = \underset{\mathbf{w}}{\operatorname{argmax}} \frac{\mathbf{w}^\top \mathbf{C} \mathbf{w}}{\mathbf{w}^\top \mathbf{w}}, \tag{2.11}$$

where $\mathbf{C}$ is a symmetric positive definite matrix, e.g., the covariance or correlation matrix. Specifically, the sample covariance matrix is defined as

$$\mathbf{C} = \frac{1}{n-1} \mathbf{X}^\top \mathbf{X}. \tag{2.12}$$

Finally, the method of Lagrange multipliers can be used to solve the problem

$$\mathcal{L}(\mathbf{w}_1, \lambda_1) = \max_{\mathbf{w}_1, \lambda_1} (\mathbf{w}_1^\top \mathbf{C} \mathbf{w}_1 - \lambda_1 (\mathbf{w}_1^\top \mathbf{w}_1 - 1)), \tag{2.13}$$

which leads to the well known eigenvalue problem

$$\mathbf{C}\mathbf{w}_1 = \lambda_1 \mathbf{w}_1. \tag{2.14}$$

Hence, the first principal direction for the mean centered matrix $\mathbf{X}$ is given by the dominant eigenvector $\mathbf{w}_1$. More generally, the principal directions of the covariance matrix $\mathbf{C}$ are the columns of the eigenvector matrix $\mathbf{W} \in \mathbb{R}^{p \times p}$ and the corresponding eigenvalues λ are the diagonal elements of $\mathbf{\Lambda} \in \mathbb{R}^{p \times p}$. Interestingly, the eigenvalues express exactly the amount of variation explained by the principal components

$$\mathbf{C}\mathbf{W} = \mathbf{\Lambda}\mathbf{W}. \tag{2.15}$$

More compactly, we can compute the principal components $\mathbf{Z} \in \mathbb{R}^{n \times p}$ as

$$\mathbf{Z} = \mathbf{X}\mathbf{W}. \tag{2.16}$$

Hence, the matrix $\mathbf{W}$ can also be interpreted as a projection matrix that maps the original observations to the new coordinates in the eigenspace. Since the eigenvectors have unit norm, the projection should be purely rotational without any scaling; thus, the matrix $\mathbf{W}$ is also denoted as a rotation matrix. However, we want to stress that the term 'loading' in general refers to the scaled eigenvectors

$$\mathbf{L} = \mathbf{W}\mathbf{\Lambda}^{0.5} \tag{2.17}$$

which, in some situations, provide a more insightful interpretation of the principal components.[3] The loading matrix $\mathbf{L} \in \mathbb{R}^{n \times p}$ has the properties:

- The squared column sums equal the eigenvalues.
- The squared row sums equal the amount of a variable's variance.

In practice, we are often interested in a useful low-dimensional representation to reveal the coherent structure of the data. However, the number of principal components to retain is subtle and often domain

[3]Technically, the eigenvectors can be seen as direction cosines, while the corresponding eigenvalues describe the magnitude. By construction, the loading matrix aggregates both.

specific. Many different heuristics, like the scree plot, have been proposed (Jolliffe, 2002). A mathematically more refined approach is the method of the optimal hard threshold for singular values, which was recently introduced by Gavish and Donoho (2014).

Note that the analysis of variables, which are measured in different units, can be misleading and cause undesirable interpretations. This is because the eigenvectors are not scale invariant. Thus, it is favorable to use the correlation matrix instead of the covariance matrix in general.

The singular value decomposition to compute PCA. In practice the explicit computation of the covariance or correlation matrix for massive datasets can be expensive. A more computationally efficient approach to compute the principal components is the singular value decomposition, which avoids the (often) costly computation of the Gram matrix $\mathbf{X}^\top\mathbf{X}$. Specifically, the eigenvalue decomposition of the inner and outer dot product of $\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$ can be related to the singular value decomposition as follows

$$\mathbf{X}^\top\mathbf{X} = (\mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^\top)(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top) = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\top \tag{2.18a}$$

$$\mathbf{X}\mathbf{X}^\top = (\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top)(\mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^\top) = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top \tag{2.18b}$$

where the eigenvalues $\boldsymbol{\Lambda}$ are equal to the squared singular values $\boldsymbol{\Sigma}^2$, i.e., $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^2$. The eigenvectors of $\mathbf{X}\mathbf{X}^\top$ are given by the left singular vectors $\mathbf{U}$ and the eigenvectors of $\mathbf{X}^\top\mathbf{X}$ are given by the right singular vectors $\mathbf{V}$ of the matrix $\mathbf{X}$. Thus, from the SVD of $\mathbf{X}$ we recover the rotation matrix $\mathbf{W}$ and eigenvalues $\boldsymbol{\Lambda}$ of the covariance matrix $\mathbf{C}$ as $\boldsymbol{\Lambda} = \frac{1}{n-1}\boldsymbol{\Sigma}^2$ and $\mathbf{W} = \mathbf{V}$. Moreover, the principal component scores can be computed as

$$\mathbf{Z} = \mathbf{X}\mathbf{W} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top\mathbf{W} = \mathbf{U}\boldsymbol{\Sigma}. \tag{2.19}$$

2.3.2 Randomized Algorithm

The information in the data is often explained by just the first few dominant principal components, and thus the randomized singular value decomposition provides an efficient and fast algorithm for computing the first k principal components. This approach is denoted randomized

principal component analysis (rPCA) as introduced by Rokhlin et al. (2009), and Halko et al. (2011a). See also Szlam et al. (2014) for some interesting implementation details. However, instead of using Algorithm 1 directly, we use a slightly modified Algorithm. Instead of computing the SVD of $\mathbf{B}$, the eigendecomposition of the smaller $\mathbf{B}\mathbf{B}^\top$ matrix is computed. The concept is depicted in Figure 2.7.

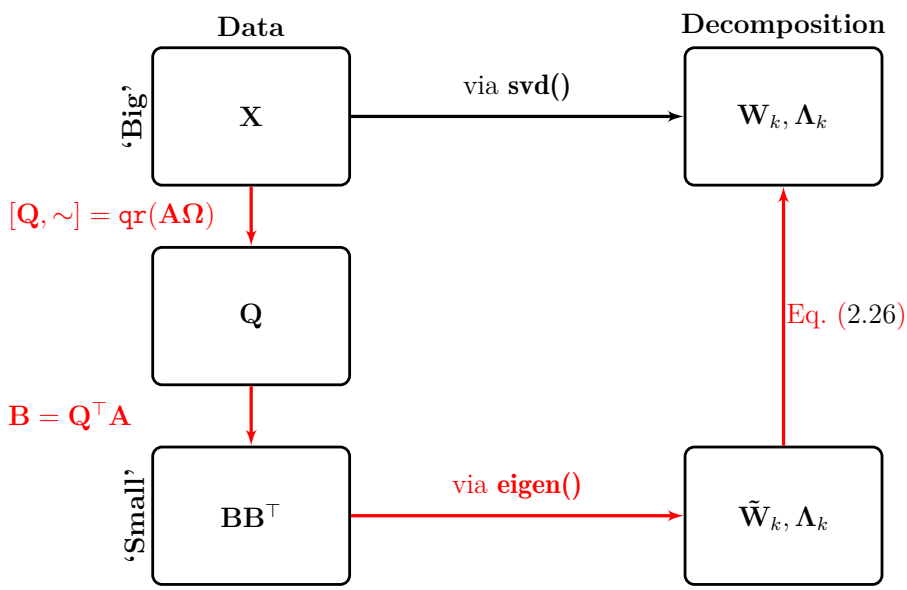


Fig. 2.7 Conceptual architecture of the randomized principal component analysis. The data are first compressed via right multiplication by a sampling matrix $\mathbf{\Omega}$, and a natural basis $\mathbf{Q}$ is formed. Then, the smaller matrix $\mathbf{B}$ is derived. Next, the eigenvalue decomposition of the outer product $\mathbf{B}\mathbf{B}^\top$ is computed. Finally, the rotation matrix $\mathbf{W}_k$ may be reconstructed from the compressed rotation matrix $\tilde{\mathbf{W}}_k$ by the expression in Eq. (2.5).

Specifically, for use in PCA, we compute the approximate randomized low-rank eigenvalue decomposition of $\mathbf{X}^\top\mathbf{X}$ given the rectangular mean centered matrix $\mathbf{X}$. Note that if $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, then

$$\mathbf{X}^\top\mathbf{X} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top \approx \mathbf{V}_k\mathbf{\Sigma}_k^2\mathbf{V}_k^\top, \tag{2.20}$$

and

$$\mathbf{X}\mathbf{X}^\top = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top \approx \mathbf{U}_k\mathbf{\Sigma}_k^2\mathbf{U}_k^\top. \tag{2.21}$$

Performing such an eigendecomposition for an $m \times n$ matrix with many columns would be expensive, since $\mathbf{X}^\top\mathbf{X}$ is $n \times n$. On the other hand, the first k singular vectors and values of $\mathbf{X}$ are approximately captured by

$$\mathbf{B} = \mathbf{Q}^\top\mathbf{X}, \tag{2.22}$$

where $\mathbf{Q}$ is such that

$$\mathbf{Q}\mathbf{Q}^\top\mathbf{X} \approx \mathbf{X}. \tag{2.23}$$

The $l \times n$ matrix $\mathbf{B}$ could still be large, so instead we can work with the $l \times l$ matrix $\mathbf{B}\mathbf{B}^\top$. Notice that if $\mathbf{B} = \tilde{\mathbf{U}}\boldsymbol{\Sigma}\mathbf{V}^\top$, which we do not compute, then

$$\mathbf{B}\mathbf{B}^\top = \tilde{\mathbf{U}}\boldsymbol{\Sigma}^2\tilde{\mathbf{U}}^\top. \tag{2.24}$$

Now, suppose that we instead construct the eigendecomposition of

$$\mathbf{B}\mathbf{B}^\top = \tilde{\mathbf{W}}\boldsymbol{\Lambda}\tilde{\mathbf{W}}^\top. \tag{2.25}$$

Then, we note that from $\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$ it follows that $\mathbf{V}^\top = \boldsymbol{\Sigma}^{-1}\mathbf{U}^\top\mathbf{A}$. Further, we have that $\tilde{\mathbf{U}} = \tilde{\mathbf{W}}$, and thus the eigenvectors (right singular vectors) are approximately recovered via

$$\mathbf{W} = \mathbf{V} = \mathbf{B}^\top\tilde{\mathbf{W}}\boldsymbol{\Lambda}^{-0.5}. \tag{2.26}$$

The procedure is summarized in Algorithm 2.

2.4 Robust Principal Component Analysis

2.4.1 Conceptual Overview

In the previous two sections we have summarized two closely related methods for dimensionality reduction, namely the singular value decomposition and principal component analysis. In Section 2.6, we show that these two methods can be used for noise reduction/removal, if the data are corrupted by independent identically distributed (i.i.d.) Gaussian noise, and if the signal-to-noise ratio is sufficiently large. However, in many practical applications we face data with arbitrarily corrupted observations,

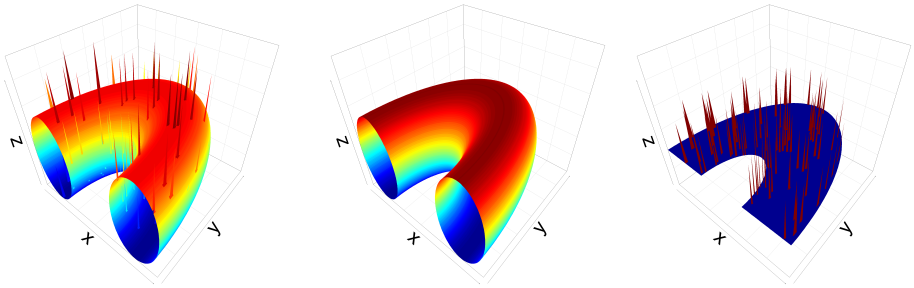
Algorithm 2 A randomized PCA algorithm.

```
function [Λ, W] = reigen(X, k, p, q)
    l = k + p                                  slight oversampling
    Ω = rnorm(n, l)                            generate sampling matrix
    Y = XΩ                                     draw range samples
    for j = 1, ..., q                          perform optional subspace iterations
        [Q, ~] = qr(Y)
        Z = X^T Q
        [Q, ~] = qr(Z)
        Y = XQ
    end for
    [Q, ~] = qr(Y)                             form orthonormal samples matrix
    B = Q^T X                                  project to smaller space
    BBt ← BB^T                                 compute outer product
    BBt ← 0.5 * (BBt + BBt^T)                  ensure symmetry
    [Λ, W̃] = eigen(BBt)                        compact eigendecomposition
    W = B^T W̃ Λ^(-0.5)                         recover right singular vectors
    W = W(:, 1 : k)                            extract k components
    Λ = Λ(1 : k, 1 : k)                        extract k components
```

e.g., shadows in images or moving objects in videos. In this case the resulting low-rank approximations can be poor, even if only a very few observations are corrupted. Figure 2.8 presents a toy example of such a possible situation. Subplot (a) shows a grossly corrupted data matrix as the superposition of a half torus (b) and spike noise (c). Specifically, the corrupted data matrix here is superimposed of a low-rank component ($r = 3$) and a sparse component. The question arises, whether a corrupted data matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ can be robustly separated into a low-rank matrix $\mathbf{L} \in \mathbb{R}^{m \times n}$ and sparse matrix $\mathbf{S} \in \mathbb{R}^{m \times n}$

$$\mathbf{A} = \mathbf{L} + \mathbf{S} \tag{2.27}$$

so that $\mathbf{S}$ captures the perturbations. Indeed, Candès et al. (2011) proved that it is possible to exactly separate such a data matrix into both its low-rank and sparse components, under rather broad assumptions. This is achieved by solving a convenient convex optimization problem, called principal component pursuit (PCP). The objective is to minimize a weighted combination of the nuclear norm $\|\cdot\|_* := \sum_i \sigma_i$ (sum of the



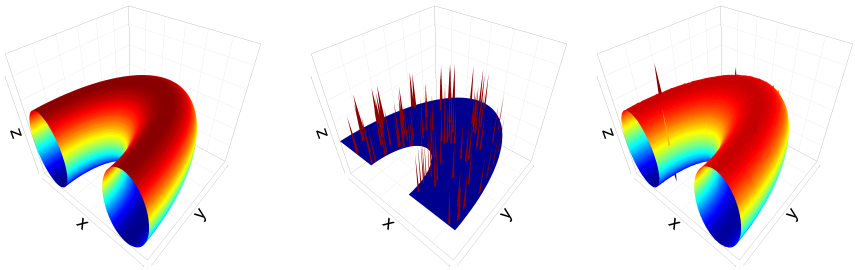
(a) Grossly corrupted torus. (b) Original half torus. (c) Sparse (spiked) noise.

Fig. 2.8 Toy example of a grossly corrupted data matrix. Subplot (a) shows the perturbed torus as a superposition of the half torus (b) and spike noise (c).

singular values) and and the ℓ_1 norm $\|\cdot\|_1 := \sum_{ij} |m_{ij}|$ as follows

$$\min_{\mathbf{L},\mathbf{S}} \|\mathbf{L}\|_* + \gamma\|\mathbf{S}\|_1 \quad \text{subject to } \mathbf{A} - \mathbf{L} - \mathbf{S} = 0. \tag{2.28}$$

The arbitrary balance parameter, $\gamma > 0$, is typically chosen to be $\gamma = 1/\sqrt{\max(n, m)}$. Hence, the key to achieving such a matrix decomposition is ℓ_1 optimization. The PCP concept is mathematically sound and has been applied successfully to images and videos (Wright et al., 2009). More generally the concept of robustly separating corrupted matrices is denoted as robust principal component analysis (RPCA). The method achieves an exact separation of the corrupted half torus into the low-rank and sparse components, shown in Figure 2.9. The reconstruction error is nearly negligible, while in comparison the low-rank SVD approximation contains substantial defects. Its remarkable ability to separate high-dimensional matrices into low-rank and sparse component makes robust principal component analysis an invaluable tool for data science.



(a) Low-rank component **L**. (b) Sparse component **S**. (c) SVD approximation.

Fig. 2.9 Subplot (a) and (b) show the separation of the grossly corrupted torus in Figure 2.8 using robust PCA. The low-rank component captures the original torus faithfully, and the reconstruction error is as low as 0.0003%. In contrast, the approximation performance of the the low-rank SVD is poor, shown in subplot (c). The reconstruction error is about 1.81%.

However, its biggest challenge is computational efficiency, especially given the iterative nature of the optimization required. Bouwmans et al. (2016b) have identified more then 30 related algorithms to the original PCP approach, aiming to overcome the computational complexity and to generalize the original algorithm.

2.4.2 Randomized Algorithm

The inexact augmented Lagrange multiplier method (Lin et al., 2011) is a popular choice, due to its favorable computational properties. This method essentially formulates the following Lagrangian function

$$\mathcal{L}(\mathbf{L},\mathbf{S},\mathbf{Y},\mu) = \|\mathbf{L}\|_* + \gamma\|\mathbf{S}\|_1 + \langle \mathbf{Y}, \mathbf{A}-\mathbf{L}-\mathbf{S}\rangle + \frac{\mu}{2}\|\mathbf{A}-\mathbf{L}-\mathbf{S}\|_F^2 \quad (2.29)$$

where μ is a positive scalar and the matrix $\mathbf{Y}$ is the Lagrange multiplier. We omit here a detailed discussion of the computational steps required to obtain the decomposition, and refer instead to Lin et al. (2011). However, the singular value decomposition lies at the heart of the algorithm, and accounts for the majority of the computational load. Now, replacing the deterministic by the randomized SVD algorithm leads to substantial computational savings. Further, the randomized algorithm has the interesting

property that it introduces a natural regularization step through the oversampling parameter p. Thus, the convergence is often more smooth.

2.5 The rsvd Package

The presented randomized matrix algorithms are becoming increasingly popular and implementations are now available in a variety of programming languages, and machine learning libraries (IBM Reseach Division, 2014; Liutkus, 2014; Okanohara, 2011; Pedregosa et al., 2011; Voronin and Martinsson, 2015). However, as far as we are aware, there has been no previous package for the programming language R. The **rsvd** package aims to fill the gap, and is available on the Comprehensive R Archive Network (CRAN). Specifically, the package provides the following core functions:

- Randomized singular value decomposition: **rsvd()**.
- Randomized principal component analysis: **rpca()**.
- Randomized robust principal component analysis: **rrpca()**.

For installation instructions visit the GIT repository https://github.com/Benli11/rSVD. See also the package manual, as all package functions are fully documented.

The interface of the **rsvd()** and **rpca()** functions is designed similar to the corresponding R base functions **svd()** and **prcomp()**. The latter function (based on the singular value decomposition) is the preferred routine for computing the principal component analysis in R. This is because, using the SVD is computationally more efficient in general than using the eigenvalue decomposition, like the **princomp()** function does (Venables and Ripley, 2002). The **rpca()** function can be used as a plug-in function for the **prcomp()** functions to benefit of the fast randomized algorithm, while providing the familiar summary statistics and plots. In addition, also fancy plot functions are provided to visualize the results of the principal component analysis. The **rrpca()** functions computes the robust principal component analysis using the augmented Lagrange multiplier method Lin et al. (2011). This function is designed so that the user can choose between either the randomized or deterministic SVD algorithm. All functions are based on the underlying LAPACK

software package (Anderson et al., 1999). These routines are numerical stable and highly accurate (full double precision).

2.5.1 The rsvd() Function

The base function for computing the SVD in R is the function **svd()**. This function provides an interface to the underlying LAPACK SVD routines (Anderson et al., 1999). These routines are known to be numerical stable and highly accurate, but computationally demanding. Specifically, the computational time required to approximate large-scale data is tremendous. In many applications, the full SVD is not necessary; only the truncated factorization is required. Thus, an alternative for constructing low-rank approximations of large-scale data are partial singular value decomposition algorithms. As far as we are aware, only the **svd** (Korobeynikov and Larsen, 2016) and **Rspectra** (Qiu et al., 2016) packages provide competitive functions for computing the partial singular value decomposition in R. The former package provides a wrapper for the **PROPACK** SVD algorithm (Larsen, 1998). The latter package is inspired by the software package **ARPACK** (Lehoucq et al., 1998) and provides a fast partial SVD and eigendecomposition algorithm. Both packages are in particular powerful for computing the singular value decomposition of large and sparse or structured matrices.

The function **rsvd** provides a efficient routine for computing the partial singular value decomposition using Algorithm 1. Specifically, the randomized singular value decomposition function **rsvd()** gains a very significant speedup when one seeks a low-rank approximation with a small rank relative to the ambient matrix dimension. The interface of the **rsvd()** functions follows the base **svd()** function

```
obj <- rsvd(A, k = NULL, nu = NULL, nv = NULL, p = 10, q = 1,
sdist = "unif", vt = FALSE)
```

Thus, it can be simply used as plug-in function. The first mandatory argument **A** passes the $m \times n$ input data matrix. The second argument **k** sets the target rank, and it is assumed that **k** is smaller then the ambient dimensions of the input matrix $k \leq min(m, n)$. If no target rank is specified, then the SVD is computed using the deterministic algorithm.

Similar to the **svd()** function the arguments **nu** and **nv** can be used to specify the number of left and right singular vectors to be computed. Most important are the two tuning parameters **p** and **q** in order to control the accuracy of the algorithm. The former parameter is used to oversample the basis, and is set by default to **p=10**. This setting guarantees a good basis with high probability in general. The parameter **q** can be used to compute additional power iterations (subspace iterations). By default this parameter is set to **q=1** which shows a good performance in our numerical experiments. The default values show a optimal trade-off between speed and accuracy in standard situations. If the singular value spectrum of the input matrix is slowly decaying, more power iterations are desirable. However, in practice we have not encountered a situation which requires more then three subspace iterations ($q > 3$). Further, the **rsvd()** routine allows to choose between a Gaussian and uniform test matrix. The different options can be selected via the optional argument **sdist=c("normal", "uniform")**. While there is no significant practical difference in terms of accuracy, the generation of uniform samples is slightly more computationally efficient. The resulting model object, **obj**, is itself a list. It contains the following components:

- **d**: k-dimensional vector containing the singular values.
- **u**: $m \times k$ matrix containing the left singular vectors.
- **v**: $n \times k$ matrix containing the right singular vectors. Note, that **v** is not returned in its transposed form as it often in other programing languages.

More details are provided in the corresponding documentation, see **?rsvd**.

2.5.2 The rpca() Function

The base function for computing the PCA in R is the function **prcomp()**, which is based on the singular value decomposition. This function is more efficient in general then using the eigenvalue decomposition, like the **princomp()** function does (Venables and Ripley, 2002).

The **rpca()** function provides an efficient routine for computing the dominant principal components using either Algorithm 1 or Algorithm 2.

The interface of the **rpca()** function is natural and follows the **prcomp()** function

```
obj <- rpca(A, k = NULL, center = TRUE, scale = TRUE,
loading = FALSE, retx = FALSE, svdalg = "auto",
p = 10, q = 1, ...)
```

The first mandatory argument **A** passes the $m \times n$ input data matrix. The second argument **k** sets the target rank, and it is assumed that **k** is smaller then the ambient dimensions of the input matrix $k \leq min(m, n)$. If no target rank is specified, then the PCA is computed using a deterministic algorithm, i.e., it is using the same algorithm as the **prcomp()** function does. By default it is not assumed that the input matrix is centered or scaled. This is done internally, and can be controlled via the arguments **center** and **scale**. By default both parameters are set to true, so that the analysis is based on the correlation matrix. The covariance matrix can be used instead, by setting **scale = FALSE**. The argument **loading** can be set to **TRUE** in order to unit scale the eigenvectors, i.e., scale the eigenvectors by the square root of the eigenvalues. The argument **retx** is indicating whether the rotated variables should be returned. This is required, for instance, to display the biplot. The argument **svdalg** allows the user to choose between the underlying Algorithms 1 and 2. From a computational perspective, the latter algorithm is slightly favorable for large matrices. By default the **rpca()** routine automatically chooses between the algorithms, and if **k** is larger then $k > min(n, m)/1.5$, then the deterministic SVD algorithm **svd()** is used. This is, because the randomized algorithms have no computational advantage over the deterministic SVD algorithm, if the target rank is relatively large compared to the ambient dimensions of the input matrix. The resulting model object, **obj**, is a list and contains the following components:

- **rotation**: $n \times k$ matrix containing the rotations (eigenvectors).
- **eigvals**: k-dimensional vector containing the eigenvalues.
- **sdev**: k-dimensional vector containing the standard deviations of the principal components, i.e., the square root of the eigenvalues.
- **x**: if **retx=TRUE**, then a $m \times k$ matrix containing the scores (rotated data) is returned.

- **center, scale**: the numeric centering and scalings used (if any).

Utility functions

Like **prcomp()**, the **rpca()** routine comes with various generic functions that can be used to summarize and display the model information.

The **summary()** function provides information about the explained variance, standard deviations, proportion of variance as well as the cumulative proportion of the computed principal components. These results can also be informatively displayed using the corresponding **plot()** function. In addition to the standard plot functions, also pretty plots are provided using **ggplot** (Wickham, 2009).

The **print()** function can be used to print the rotations (eigenvectors).

Further a correlation plot can be displayed using the **ggcorplot()** function as well as a biplot **ggbiplot()**.

2.5.3 The rrpca() Function

Robust principal component analysis (RPCA) is a method for the robust separation of a data matrix into a low-rank component **L** and a sparse component **S**. The **rrpca()** function is implementing a randomized routine based on the inexact augmented Lagrange multiplier method (IALM). Specifically, it is replacing the deterministic by the randomized SVD. While the randomized algorithm is faster, it also has the effect of implicitly regularizing the approximation. The interface of the **rrpca()** function is as follows

```
obj <- rrpca(A, k = NULL, lamb = NULL, gamma = 1.25,
rho = 1.5, maxiter = 50, tol = 0.001, svdalg = "auto",
p = 10, q = 1, trace = FALSE, ...)
```

The first mandatory argument **A** passes the $m \times n$ input data matrix. The second argument **k** sets the target rank, and it is assumed that **k** is smaller then the ambient dimensions of the input matrix $k \leq min(m, n)$. If, no target rank is specified, then **k** is set equal to 2. The arguments **lamb**, **gamma** and **rho** are tuning parameters. These parameters are described in detail in (Lin et al., 2011), and we recommend using the default values.

The arguments **maxiter** and **tol** can be used to control the maximum numbers of iterations and the desired approximation accuracy. The argument **trace** can be set to **TRUE** to print the progress of the iterative process. Further the argument **svdalg** can be used to chose between the deterministic and the randomized SVD algorithm, i.e., **svdalg = c("rsvd", "svd")**. The resulting model object, **obj**, is a list and contains the following components:

- **L**: $m \times n$ matrix containing the low-rank component.
- **S**: $m \times n$ matrix containing the sparse component.
- **k**: integer denoting the target rank used for the final iteration.
- **err**: vector which contains the Frobenious error achieved in each iteration.

For more details we refer to the documentation, see **?rrpca**.

2.6 Numerical Results

In this section, we demonstrate the usage of randomized matrix algorithms within the programing language R. We illustrate the performance of our routines using several standard examples and compare the results to the corresponding deterministic R functions. Section 2.6.1 starts with a classic example showing how the randomized singular value decomposition function **rsvd()** can be used for image compression. Section 2.6.2 demonstrates the randomized principal component analysis routine **rpca()** for computing eigenfaces. Section 2.6.3 shows how the randomized robust principal component analysis function **rrpca()** can be used for foreground/background separation of videos. Finally, Section 2.6.4 investigates the performance of the **rsvd()** function, showing speed-ups ranging from 5 to 150 times.

A workstation with an Intel Xeon CPU E5-2620 2.4GHz, 32GB DDR3 memory, and the operating-system Ubuntu 16.04 LTS is used for all following computations and the **microbenchmark** package is utilized for evaluating the computational runtime of the algorithms (Mersmann et al., 2015).

2.6.1 SVD Example: Image Compression

The singular value decomposition can be used to obtain a low-rank approximation to high-dimensional data. Image compression is a simple and illustrative example of this.[4] Although images often feature a high-dimensional ambient space, the underlying structure can be represented by a very sparse model. This means that most natural images can be faithfully recovered from a relatively small set of basis functions. For demonstration a 1600×1200 grayscale image is used in the following. A grayscale image may be thought of as a real-valued matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, where m and n are the number of pixels in the vertical and horizontal directions, respectively. To compress the image we must first decompose the image. The singular vectors and values provide a hierarchical representation of the image in terms of a new coordinate system defined by dominant correlations within the image. Thus, the number of singular vectors used for approximation poses a trade-off between the compression rate (i.e., the number of singular vectors to be stored) and the image details.

We use the **rsvd()** function to compute the first $k = 100$ singular values and vectors. Specifically, we use the default settings for the oversampling parameter $p = 10$, while varying the number of power iterations. Figure 2.10 presents the results. First, we see that the truncated deterministic **svd()** function achieves an error as low as 0.121. Indeed, the result illustrates that in general natural images feature a very compact representation. Here, the normalized root mean squared error (NRMSE) is used to quantify the error, which is a common measure for the reconstruction quality of images (Fienup, 1997).

Next, we see that the **rsvd()** routines achieves near-optimal approximations. By visual inspection no significant difference can be seen between (b) and (d,e,f). However, it can be seen that the quality slightly suffers by excluding the computation of at least one subspace iteration (c). Using the default tuning parameters, i.e., $p = 10$, and $q = 1$ the reconstruction error is about 0.125. However, the randomized algorithm is substantially faster. Specifically, we gain a speedup by a factor of about 7.

[4] Among the many strategies to compress or denoise images, the singular value decomposition is one prominent tool, although it is certainly not the most effective.

The results are summarized in Table 2.1. In addition, here the median elapsed run-time for other fast SVD algorithms in R are shown. The **svds()** function of the **Rspectra** package achieves a speedup of about 4.2, while the **propack.svd()** function only achieves a marginal speedup of about 1.1. Note, that the computational gain of the randomized algorithm becomes more pronounced with increased matrix dimension, e.g., images with higher resolution.

Table 2.1 Summary of algorithms runtime and errors. The randomized routines achieve a substantial speedup, while attaining similar reconstruction errors with $q \geq 1$.

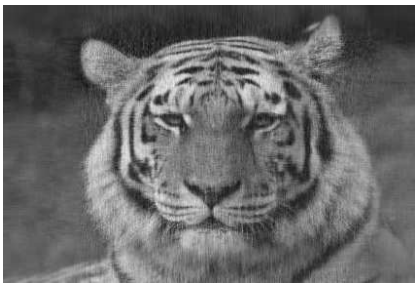
Method	Parameters	Time (s)	Speedup	Error
svd()		1.05	-	0.121
propack.svd()	**neig=100**	0.94	1.11	0.121
svds()	**k=100**	0.25	4.21	0.121
rsvd()	**k=100, q=0**	0.08	12.80	0.163
	k=100, q=1	0.15	7.05	0.125
	k=100, q=2	0.22	4.86	0.122
	k=100, q=3	0.29	3.62	0.121

Figure 2.11 shows the corresponding singular values and it can be seen that the randomized algorithm faithfully captures the true singular values with $q = \{1, 2\}$ subspace iterations. However, without computing subspace iterations, the singular values decay slightly.

The singular value decomposition is also a numerically reliable tool for extracting a desired signal from noisy data. The central idea is that the small singular values mainly represent the noise, while the dominant singular values represent a filtered (less noisy) signal. Thus, the low-rank SVD approximation can be used to denoise data matrices. To demonstrate this concept, the original image is first corrupted with additive white noise. Then, the underlying image (signal) is approximated using again the first $k = 100$ singular vectors, illustrated in Figure 2.12. The NRMSE indicates an improvement by about 15%.

(a) Original image.

(b) SVD.
(NRMSE=0.121)



(c) rSVD using $q = 0$.
(NRMSE=0.163)

(d) rSVD using $q = 1$.
(NRMSE=0.125)

(e) rSVD using $q = 2$.
(NRMSE=0.122)

(f) rSVD using $q = 3$.
(NRMSE=0.121)

Fig. 2.10 Image compression using the SVD. Subplot a) shows the original image. Subplot (b) shows the reconstructed image using the dominant $k = 100$ singular vectors obtained via the deterministic SVD algorithm. The following subplots (c,d,e,f) show the approximation results using the randomized SVD algorithm. The reconstruction quality using $q = 1$ subspace iterations is insignificant compared to the deterministic SVD algorithm.

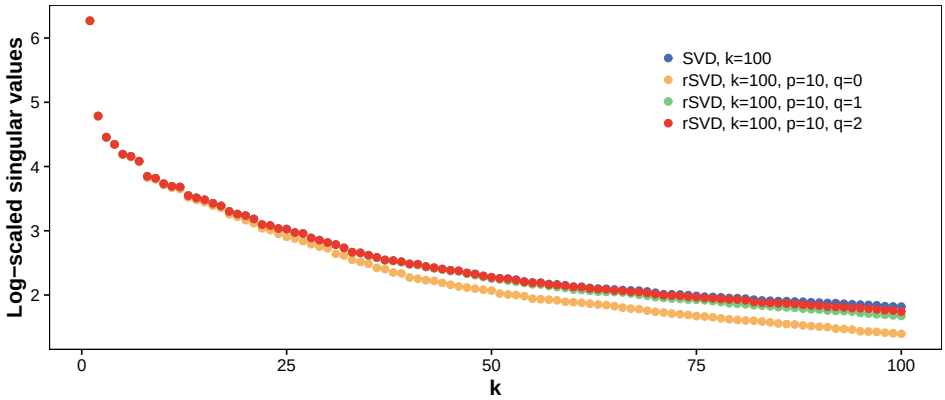


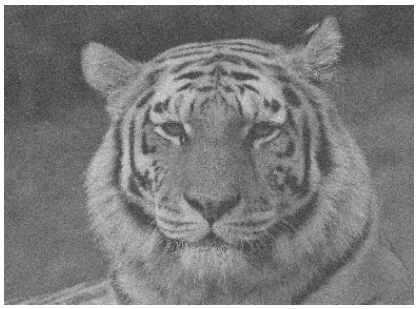
Fig. 2.11 Dominant log-scaled singular value spectrum of the image 'tiger'.

2.6.2 PCA Example: Eigenfaces

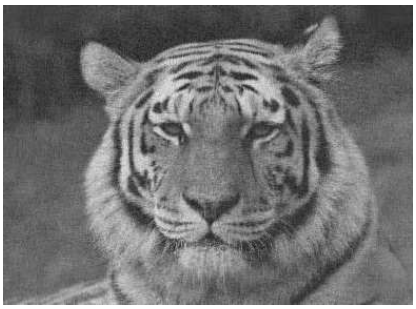
One of the most striking demonstrations of PCA are eigenfaces, first studied by Kirby and Sirovich (1990). The aim is to extract the most dominant correlations between different faces from a large set of facial images. Specifically, the resulting columns of the rotation matrix (i.e., the eigenvectors) represent 'shadows' of the faces, the so-called eigenfaces. Specifically, the eigenfaces reveal both inner face features (e.g., eyes, nose, mouth) and outer features (e.g., head shape, hairline, eyebrows). These features can then be used for facial recognition and classification as first shown by Turk and Pentland (1991).

In the following we use the downsampled cropped Yale face database B (Georghiades et al., 2001). The dataset comprises 2410 grayscale images of 38 different people, cropped and aligned. For computational convenience the 96×84 faces images are stored as column vectors. We approximate the first $k = 20$ dominant eigenfaces using the **rpca()** function. Here, the analysis is performed on the correlation matrix by setting the argument **scale = TRUE**. The summary (using **summary()**) is as follows

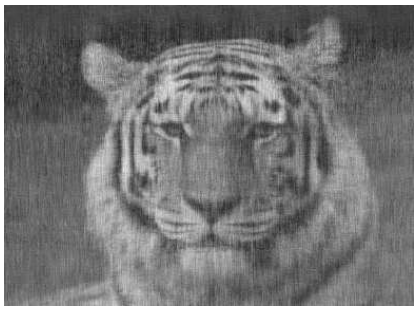
```
R>                          PC1        PC2      PC3    ...
R> Explained variance     2901.539  2706.699  388.053  ...
R> Standard deviations      53.866    52.026   19.699  ...
R> Proportion of variance    0.360     0.336    0.048  ...
R> Cumulative proportion     0.360     0.695    0.744  ...
```



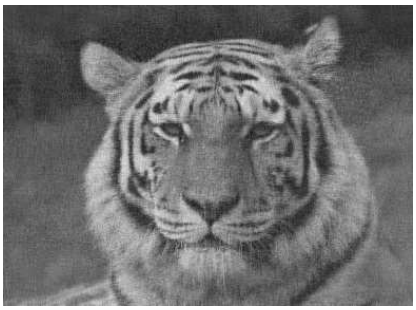
(a) Noisy image.
(NRMSE=0.351)



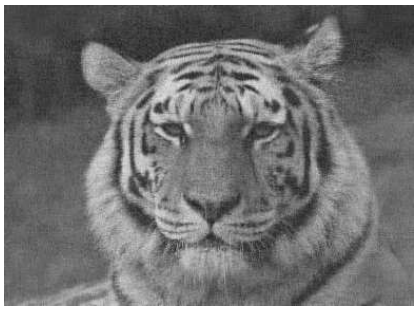
(b) SVD.
(NRMSE=0.200)



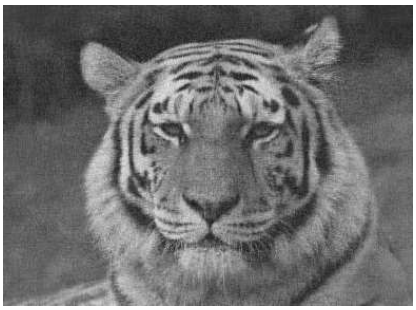
(c) rSVD using $q = 0$.
(NRMSE=0.268)



(d) rSVD using $q = 1$.
(NRMSE=0.207)



(e) rSVD using $q = 2$.
(NRMSE=0.203)



(f) rSVD using $q = 3$.
(NRMSE=0.202)

Fig. 2.12 Image denoising using the singular value decomposition. Both the deterministic and randomized SVD reduce the error by about 15%.

```
R> Eigenvalues              2901.539  2706.699  388.053  ...
```

Just the first 3 PCs explain about 74% of the total variation in the data, while the first 20 PCs explain more then 88%. The summary can be visualized, as shown in Figure 2.13.

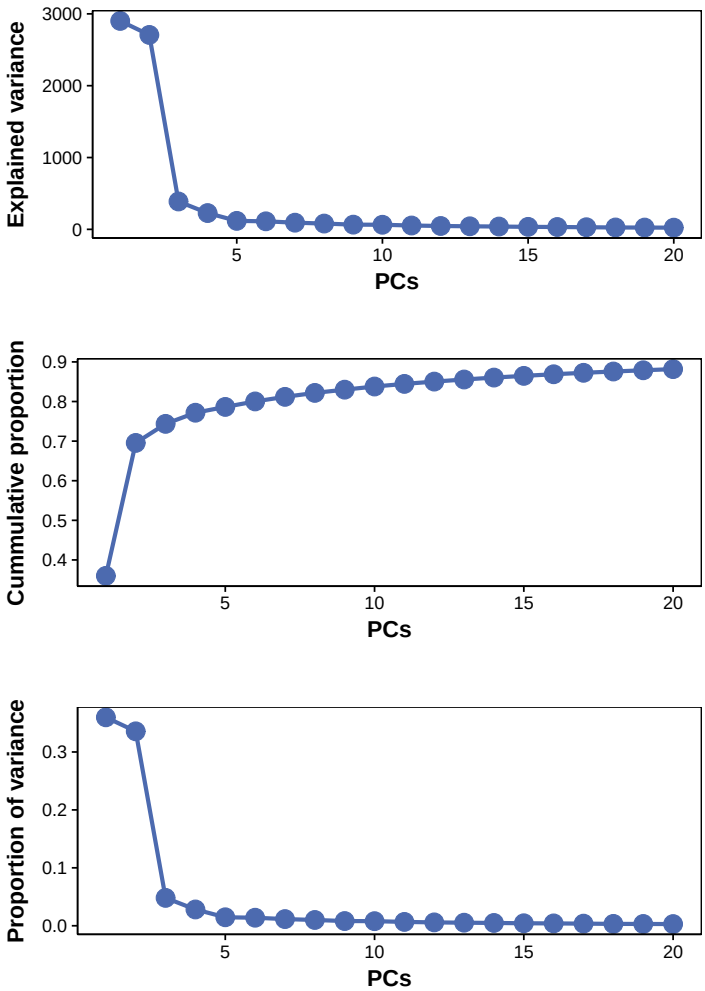
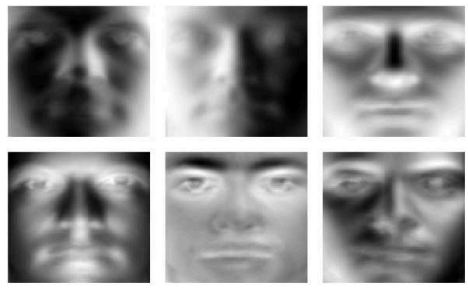


Fig. 2.13 The upper plot shows the explained variance (eigenvalues) in decaying order. The middle plot shows the cumulative proportion of the explained variance, and the lower plot shows the proportion explained by the principal components.

Finally, the eigenvectors can be visualized as eigenfaces, e.g., the first eigenvector (eigenface) is displayed as an image. Figure 2.14 shows the first six dominant eigenfaces, computed with both the **prcomp()** and the **rpca()** function. The eigenfaces encode the facial features as well as the

illumination. To better contextualize the key features different color maps can be used. The first eigenface characterizes the facial shapes, while the second shows illumination variations. The third and fifth eigenface feature the nose and eyebrows. Note that the color maps for some of the eigenfaces are flipped. This is because the signs of the columns of the rotation matrix are arbitrary and differ between different PCA implementations.



(a) Eigenfaces.

(b) Colored eigenfaces.

(c) Randomized eigenfaces.

(d) Colored randomized eigenfaces.

Fig. 2.14 Dominant six eigenfaces computed with both the **prcomp()** and **rpca()** functions. Subplot (a) and (c) show the grayscale and (b) and (c) the colored eigenfaces using the colormap **cm.colors(255)**. Note that some of the randomized eigenfaces are sign flipped.

Figure 2.15 shows the standard deviations (eigenvalues) for the **prcomp()** and **rpca()** function. The randomized algorithm faithfully approximates the eigenvalues. The corresponding computational timings and errors are summarized in Table 2.2. The randomized algorithm achieves a speedup of about 10, while attaining a similar approximation error.

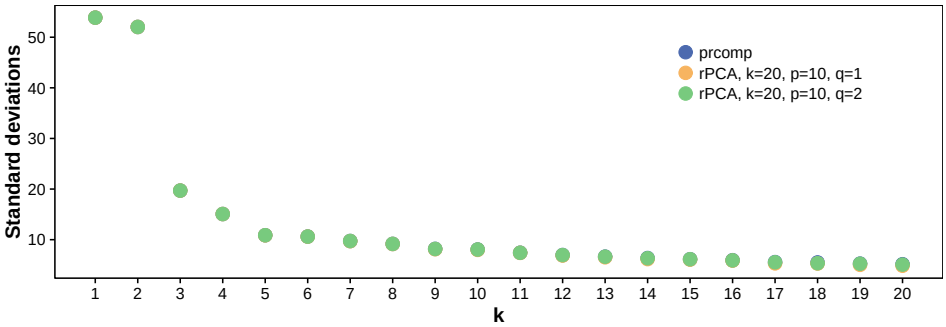


Fig. 2.15 Standard deviations for the deterministic and randomized PCA algorithm.

Table 2.2 Computational time for the deterministic and randomized PCA algorithm. The randomized algorithms achieves an about 10 fold speed-up, while attaining near-optimal results.

Method	Parameters	Time (s)	Speedup	Error
prcomp()		19.13	-	0.228
rpca()	**k=20, q=1**	1.90	10.05	0.232
	k=20, q=2	2.11	9.08	0.229

2.6.3 Robust PCA Example: Foreground/Background Separation

In the following we demonstrate the randomized RPCA algorithm for video foreground/background separation, a problem studied widely in the computer vision community (Bouwmans and Zahzah, 2014; Bouwmans et al., 2016b). Background modeling is a fundamental task in computer vision to detect moving objects in a given video stream from a static camera. The key idea is that dynamic pixels in successive video frames are considered foreground objects, whereas static pixels are considered part of the background. Thus the foreground can be found in a video by removing the background. However, background estimation is a challenging task due to the presence of foreground objects and variability in the background itself, e.g., waving trees, water fountains or illumination changes. One way to tackle this challenge is to exploit the low-rank structure of the background, while considering foreground objects as outliers.

A solution to this problem is provided by robust principal component analysis, which separates a matrix into a low-rank (background) and sparse component (representing the activity in the scene).[5] An example surveillance video (Goyette et al., 2012) containing 200 grayscale frames of size 176×144 is used in the following. Each frame is stored as a flattened column vector of the input data matrix, and we assume that the background is the low-rank component of this matrix. This assumption is reasonable since the camera is fixed and the background only gradually changes over time.

The **rrpca()** function returns two arrays: **L** and **S**. The first array contains the low-rank component and the latter the sparse component. Thus, the results can be illustrated by displaying an example frame of both arrays, shown in Figure 2.16. It clearly can be seen that the algorithm treats the foreground objects as sparse components, representing outlying entries in the data matrix. Thus the algorithm faithfully separates the video into its two components. Figure 2.17 shows the convergence

(a) Video frame. (b) Low-rank component **L**. (c) Sparse component **S**.

Fig. 2.16 Foreground/background separation of a video using **rrpca()**. Subplot (a) shows the actual video frame, which is separated into its two components. The low-rank component represents the background, and the sparse component captures the foreground objects.

performance of the **rrpca()** function using both the deterministic and the randomized SVD algorithm. Without subspace iterations, the randomized algorithm converges after about 22 iterations. Using the deterministic SVD algorithm requires slightly fewer iterations, however, each iteration is substantially more expensive. The timings are shown in Table 2.3.

[5] A general drawback of RPCA methods is that they rely on one or more tuning parameters, although, the default values are suitable in general.

Interestingly, the convergence profile of the randomized algorithm can be often smoother. This is due to the implicit regularization of the oversampling parameter p. Moreover, we stress that the randomized RPCA algorithm only achieves accurate results if the singular value spectrum is rapidly decaying, i.e., the data must exhibit low-rank structure.

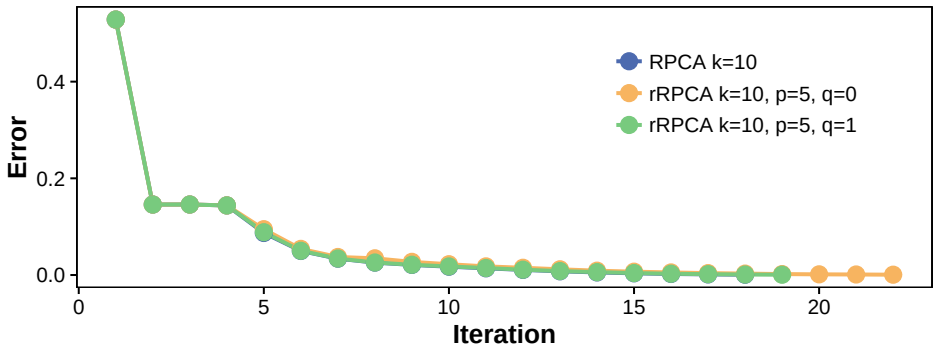


Fig. 2.17 Convergences of the **rrpca()** function using both the deterministic and randomized SVD algorithm.

Table 2.3 Computational time for the deterministic and randomized RPCA algorithm. Note, that the deterministic algorithm is induced by setting **svdalg="svd"**.

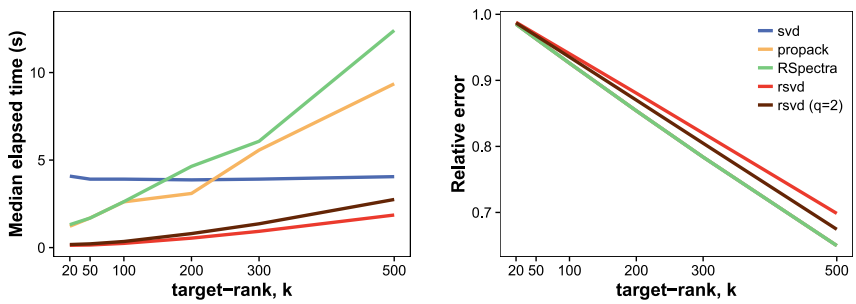
Method	Parameters	Time (s)	Iterations	Error
rrpca()	**k=10, svdalg="svd"**	14.34	18	6.18e-04
rrpca()	**k=10, p=5, q=0**	6.47	22	6.68e-04
	k=10, p=5, q=1	7.32	19	5.99e-04

2.6.4 Computational Performance

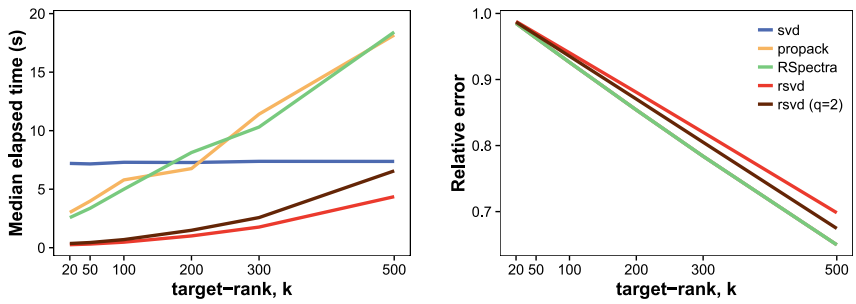
The time complexity of classic deterministic SVD algorithms is $O(mn^2)$, where it is assumed that $m \geq n$. Modern partial SVD algorithms reduce the time complexity to $O(mnk)$ (Demmel, 1997). Randomized SVD, as presented here, comes also with asymptotic costs of $O(mnk)$. The key difference, however, is that the randomized SVD algorithm (without subspace iterations) requires only two passes over the input matrix. Each

additional subspace iteration requires one more pass over the data matrix. Hence, from a practical point of view, the algorithm is computationally more efficient.

Figure 2.18 shows the computational evaluation of the base **svd()**, as well as the **propack.svd()** from the **svd** package, the **svds()** from the **Rspectra** package, and the **rsvd()** function. Here two different dense random matrices are presented. In each situation the singular values decaying linearly from 1 to 0.001. The computational time and the relative reconstruction errors are computed over a sequence of different target-ranks k. In particular, the **rsvd()** function achieves a substantial speedup



(a) Dense matrix of size 2000×2000.



(b) Dense matrix of size 5000×2000.

Fig. 2.18 Computational performance of singular value decomposition routines for dense matrices in R. The left column shows the median computational time and the right column shows the relative reconstruction error over varying target-ranks k. The **rsvd()** outperforms in terms of the computational time. Further, the error of the **rsvd()** algorithm can be controlled by the parameter p and q.

for very low-dimensional approximations of dense matrices. The advantage of the randomized algorithm becomes pronounced with an increased

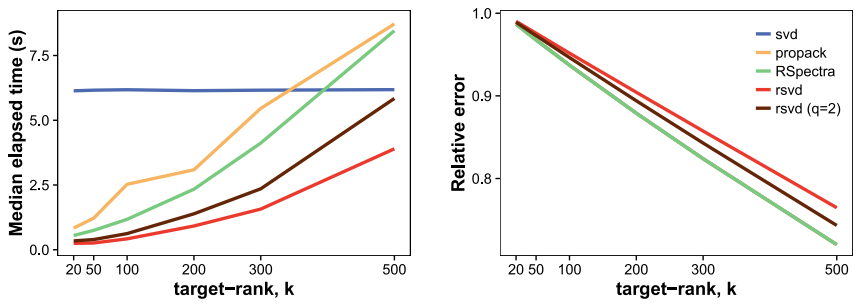
matrix dimension. Hence, the randomized SVD algorithm enables the fast decomposition of large dense data matrices, while achieving competitive reconstruction errors. It is shown that the reconstruction error can be improved by performing more subspace iterations. This allows the user to control the trade-off between computational time and accuracy.

Figure 2.19 shows the computational evaluation on a sparse matrix with only 5% non-zero elements. The two partial SVD algorithms are specifically designed for large sparse and structured matrices, and show here a significant computational improvement over the dense matrix of the same dimension. Although, the **rsvd()** algorithm is still competitive, the advantage of the **propack.svd()** and **svds()** functions is notable with about 1% non-zero elements for computing the dominant singular values and vectors.

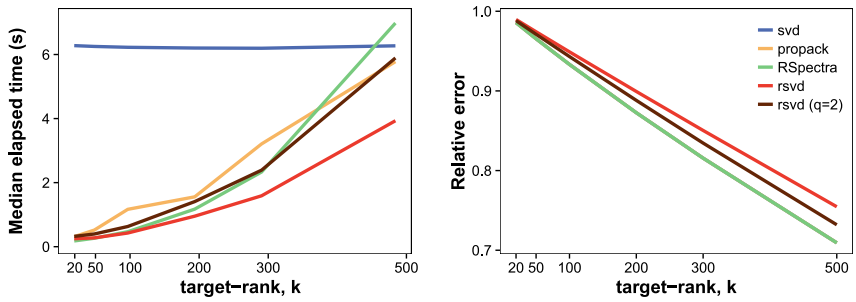
2.7 Conclusion

Due to the tremendous increase of high-dimensional data produced by modern sensors and social networks, data methods for dimensionality reduction are becoming increasingly important. However, despite modern computer power, massive datasets pose a tremendous computational challenge for traditional algorithms. Probabilistic algorithms can substantially ease the logistic and computational challenges in obtaining approximate matrix decompositions. This advantage becomes pronounced with an increasing matrix dimension. Randomized algorithms are feasible for even massive matrices where traditional deterministic algorithms fail. The randomized singular value decomposition is the most prominent and ubiquitous randomized algorithm. Its popularity is due to the strong theoretical error bounds and the advantage that the error can be controlled by oversampling and subspace iterations.

The R package **rsvd** provides computational efficient randomized routines for the singular value decomposition, principal component analysis and robust principal component analysis. The package is particularly useful for computing the approximate low rank decompositions when the rank k is substantially smaller than the matrix dimensions. The routines are intuitive and the performance evaluation shows that the randomized



(a) Sparse matrix of size 5000×2000, with 5% non-zero entries.



(b) Very sparse matrix of size 5000×2000, with 1% non-zero entries.

Fig. 2.19 Computational performance of singular value decomposition routines for sparse matrices in R. The left column shows the median computational time and the right column shows the relative reconstruction error over varying target-ranks k. The two partial SVD algorithms **propack.svd()** and **svds()** are competitive for computing the dominant singular values and vectors of large sparse or structured matrices.

algorithm provides an efficient framework to reduce the computational demands of the traditional (deterministic) algorithms. Substantial speedups are gained over other fast (partial) SVD and PCA implementations in R, while achieving a competitive reconstruction error.

The applications of the randomized matrix algorithms are ubiquitous and can be utilized for all methods relying on the computation of (generalized) eigenvalue problems. Future developments of the **rsvd** package will apply this concept to compute linear discriminant analysis, principal component regression, and canonical correlation analysis. Another important direction is to provide more efficient routines for large-scale sparse matrices using the **Matrix** package (Bates and Maechler, 2016).

Chapter 3

Randomized Dynamic Mode Decomposition

"Important thing in science is not so much to obtain new facts as to discover new ways of thinking about them."

— Sir William Bragg

Note: The work described in this chapter was carried out in collaboration with Professors J. Nathan Kutz and Steven L. Brunton of University of Washington. My contributions involve conceptualizing the project idea, implementing the routines, running the simulations as well as writing the original draft.

3.1 Introduction

The dynamic mode decomposition (DMD) is a modern dimensionality reduction technique, originally introduced in the field of fluid dynamics by Schmid (2010) and Rowley et al. (2009). Specifically, the method attempts to extract dynamic information from dynamical systems based on a sequence of snapshots (time series of data). Therefore, a decomposition in space and time is created, as schematically depicted in Figure 3.1. In contrast to traditional dimensionality reduction methods like principal component analysis, the DMD finds a set of modes (components) which contain the spatial structure, but are not orthogonal. While the imposed orthogonality restrictions of PCA are mathematically convenient, they are not always empirically plausible. Indeed, PCA often fails to accurately

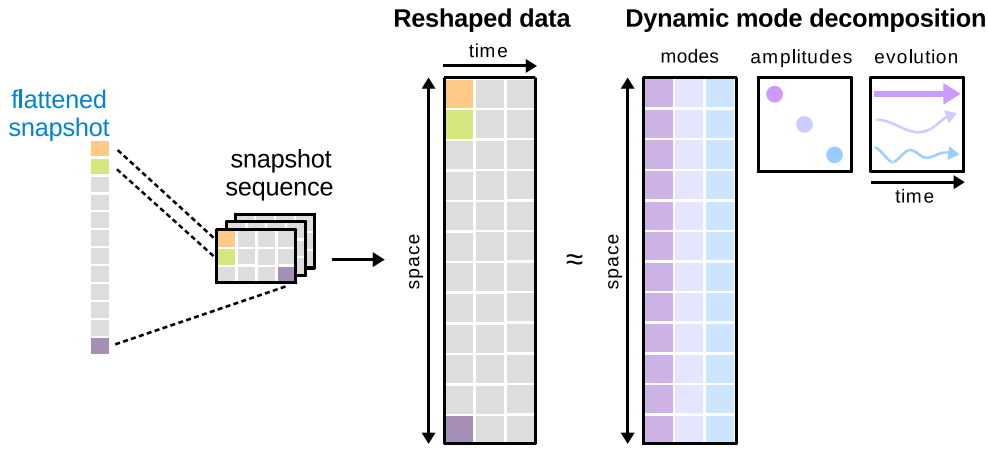


Fig. 3.1 Illustration of the dynamic mode decomposition for a given snapshot sequence describing a dynamical system.

capture the temporal behaviors in time series data. Thus, DMD is capable of providing a set of more physically meaningful modes which are associated with a damped sinusoidal behavior in time. This is possible because the DMD integrates both the Fourier transforms and the singular value decomposition.

Brief Historical Overview

The origins of the dynamic mode decomposition can be traced back to Koopman (1931). However, it had limited early impact since computers had not yet been invented and only analytic results, for simple problems, could be obtained. The work of Koopman was revived by Mezić and co-workers starting in 2004 (Mezić and Banaszuk, 2004), when both a deep theoretical understanding of dynamical systems theory and modern computers were available. In short, Koopman's theory is a dynamical systems tool that provides complete information about a nonlinear dynamical system via an associated infinite-dimensional linear operator. Specifically, it provides a theoretical characterization that is readily interpretable in terms of standard methods of dynamical systems. The dynamic mode decomposition is a special case of Koopman's theory where the so-called *Koopman observables* are just the state space itself (Kutz et al., 2016a). Schmid (2010) proposed the DMD architecture for modeling complex flows, and the connection with Koopman's theory

was only made theoretically rigorous by the subsequent work of Rowley et al. (2009). The connection between the works of Mezić, Schmid and Rowley and their co-workers between 2004-2010 laid the foundations for DMD as a transformative mathematical architecture.

Innovations and Applications

In the last few years alone, the dynamic mode decomposition has seen tremendous development in both theory and application. In theory, DMD has seen innovations around:

- **Multi-resolution analysis:** Kutz et al. (2016b) proposed the multi-resolution DMD, which integrates techniques used in multiresolution analysis. This enables the extraction of DMD modes and eigenvalues from data sets containing multiple timescales.
- **Sparsity:** Jovanović et al. (2014) proposed sparsity promoting techniques as procedures to select DMD modes. Specifically, the approach uses a ℓ_1 norm penalty to identify a smaller set of important DMD modes.
- **Control theory:** Proctor et al. (2016) proposed the DMD with control which is, in particular, designed for data obtained from input output systems. Specifically, this approach is able to disambiguate between the underlying dynamics and the effects of actuation.
- **Compressive architectures:** Brunton et al. (2015) proposed compressive sampling strategies for computing the DMD. They showed that the DMD modes and eigenvalues can be extracted from heavily subsampled data.

See also Kutz et al. (2016a) for a comprehensive overview of innovations around the dynamic mode decomposition. In addition to continued progress in fluid dynamics, the DMD has also been applied to several new domains including Neuroscience (Brunton et al., 2016), Epidemiology (Proctor and Echhoff, 2015), and video processing (Erichson and Donovan, 2016; Erichson et al., 2016a; Grosek and Kutz, 2014; Kutz et al., 2015).

Motivation and Overview

Computing the dynamic mode decomposition is computational challenging, in particular, for high-dimensional data like fluid flows. In order to ease the computational load the following two probabilistic strategies have been proposed previously:

(a) Erichson and Donovan (2016), and later Bistrian and Navon (2016) have presented an algorithm utilizing the randomized singular value decomposition (rSVD). While this approach is robust to noise, and reliable, only the computation of the SVD is accelerated. Subsequent computational steps involved in computing the DMD remain relatively expensive. Hence, this approach is not entirely qualified to handle massive data.

(b) Brunton et al. (2015), and later Erichson et al. (2016a) have presented the compressed dynamic mode decomposition, an algorithm which forms a smaller (sketched) sequence of snapshots from a small number of random linear combinations of the rows of the high-dimensional sequence of snapshots. Subsequently, the approximate dynamic modes and eigenvalues are obtained from the sketched representation. This approach achieves substantial speedups over (a), and has been shown to be successful for high-performance computing in the area of fluid flows, and video; however, the algorithm is less reliable, and is relatively sensitive to noise.

In the following, we propose a novel randomized algorithm for computing the near-optimal dynamic mode decomposition. Specifically, we embed the DMD into the probabilistic framework presented in Chapter 1. This new algorithm achieves considerable speedups over (a) as well as it is more robust, and reliable then (b). Specifically, the approximation error can be controlled via oversampling and additional power iterations. This allows the user to choose a suitable trade-off between computational time and accuracy.

Section 3.2 briefly reviews the deterministic algorithm for computing the dynamic mode decomposition. Section 3.3 outlines the compressed dynamic mode decomposition algorithm. Then, Section 3.4 presents

the randomized algorithm for computing the DMD. Section 3.5 gives an overview of the **DMDpack** package which provides implementations of the discussed algorithms in Python. Section 3.6 presents numerical results using the **DMDpack** package. Finally, concluding remarks and further research directions are outlined in Section 3.7.

3.2 Deterministic DMD

3.2.1 Conceptual Overview

Given a sequence of snapshots $\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^m$, separated in time by a constant step Δt, the dynamic mode decomposition aims to find the eigenvectors and eigenvalues of the time-independent linear map $\mathbf{M} : \mathbb{R}^m \to \mathbb{R}^m$ which approximately relates a given snapshot $\mathbf{x}_t$ with a subsequent one $\mathbf{x}_{t+1}$ as

$$\mathbf{x}_{t+1} = \mathbf{M}\mathbf{x}_t + \mathbf{e}_t, \tag{3.1}$$

where $\mathbf{e}$ denotes the residual (Schmid, 2010). Specifically, the eigenvalues λ_j and eigenvectors ϕ_j of $\mathbf{M}$ characterize the system dynamics. The ultimate goal in the DMD algorithm is to optimally construct the matrix $\mathbf{M}$ so that the error between the observed and approximated snapshot sequence is minimal in a least-square sense. Of course, the approximation holds only over the sampling window where $\mathbf{M}$ is constructed, but the approximate solution can be used to not only make future state predictions, but also to decompose the dynamics into various time-scales since the ω_j are prescribed.

There are two methods for constructing the linear map $\mathbf{M}$. Originally, the DMD were computed using an Arnoldi-like algorithm, which treats the linear map as a companion matrix (Rowley et al., 2009). This approach is, in particular, useful for theoretical analysis due to its connection with Krylov methods. The modern formulation of the DMD algorithm is based on the singular value decomposition (SVD). This approach is computationally more robust to noise in the data and to numerical errors.

We advocate here the SVD based algorithm, which is described in more detail below.

3.2.2 Deterministic Algorithm

Following Tu et al. (2014), the deterministic algorithm for computing the dynamic mode decomposition proceeds by first separating the snapshot sequence $\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^m$ into two overlapping sets of data

$$\mathbf{X}_L = \begin{bmatrix} | & | & & | \\ \mathbf{x}_0 & \mathbf{x}_1 & \cdots & \mathbf{x}_{n-1} \\ | & | & & | \end{bmatrix}, \quad \mathbf{X}_R = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \\ | & | & & | \end{bmatrix}. \tag{3.2}$$

$\mathbf{X}_L \in \mathbb{R}^{m \times n}$ and $\mathbf{X}_R \in \mathbb{R}^{m \times n}$ are called the left and right snapshot matrix, respectively. Next, we reformulate Equation (3.1) in matrix notation

$$[\mathbf{x}_1 | \mathbf{x}_2 | \cdots | \mathbf{x}_n] \approx [\mathbf{M}\mathbf{x}_0 | \mathbf{M}\mathbf{x}_1 | \cdots | \mathbf{M}\mathbf{x}_{n-1}], \tag{3.3}$$

and more compact we have

$$\mathbf{X}_R \approx \mathbf{M}\mathbf{X}_L. \tag{3.4}$$

In order to find an estimate for the matrix $\mathbf{M}$ we face the following least-squares problem

$$\hat{\mathbf{M}} = \underset{\mathbf{M}}{\operatorname{argmin}} \, \|\mathbf{X}_R - \mathbf{M}\mathbf{X}_L\|_F^2. \tag{3.5}$$

This is a well-studied problem, and an estimator for $\hat{\mathbf{M}} \in \mathbb{R}^{m \times m}$ is given by

$$\hat{\mathbf{M}} = \mathbf{X}_R \mathbf{X}_L^{\dagger}. \tag{3.6}$$

Here, the Moore-Penrose pseudoinverse, denoted as †, produces a regression that is optimal in a least-square sense. In practice, this estimator may be intractable to compute, in particular, when the input data are high-dimensional, i.e., m is large. However, for a sufficiently large time series of data, we can assume that the snapshots eventually become linearly dependent, and thus feature a low-rank structure. Hence, the

DMD algorithm circumvents the computation of $\hat{\mathbf{M}}$ by considering a rank-reduced representation $\tilde{\mathbf{M}}$. This is achieved by using the similarity transform, i.e., projecting the linear map on the dominant left singular vectors. Typically, we aim to make use of the low-rank structure of the data as well. Thus, we project the linear map only onto the subspace spanned by the dominant $k \leq min(n, m)$ left singular vectors. This yields the relatively small matrix $\tilde{\mathbf{M}} \in \mathbb{R}^{k \times k}$. However, choosing the target rank k is subjective and often problem dependent.

The DMD algorithm proceeds by first computing the SVD of $\mathbf{X}_L$

$$\mathbf{X}_L = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top. \tag{3.7}$$

If $k \leq min(n, m)$ we truncate the SVD, so that we yield $\mathbf{U} \in \mathbb{R}^{m \times k}$, $\mathbf{\Sigma} \in \mathbb{R}^{k \times k}$ and $\mathbf{V} \in \mathbb{R}^{n \times k}$. Note, that the computation of the SVD is the computational bottleneck of the algorithm. The computation of the SVD might be intractable for a massive snapshot matrix. The computational load can be reduced using modern partial SVD algorithm, which allow to compute the first k dominant singular vectors and values only. For instance, SVD algorithms based on Krylov methods or the randomized SVD are reasonable choices. The latter approach, using the randomized SVD for computing the DMD was proposed by Erichson and Donovan (2016), and later by Bistrian and Navon (2016). $\tilde{\mathbf{M}}$, the $k \times k$ projection of the full matrix $\hat{\mathbf{M}}$ onto the dominant left singular vectors, is computed as

$$\begin{aligned} \hat{\mathbf{M}} &= \mathbf{X}_R \mathbf{X}_L^\dagger = \mathbf{X}_R \mathbf{V} \mathbf{\Sigma}^{-1} \mathbf{U}^\top \\ \Longrightarrow \quad \tilde{\mathbf{M}} &= \mathbf{U}^\top \hat{\mathbf{M}} \mathbf{U} = \mathbf{U}^\top \mathbf{X}_R \mathbf{V} \mathbf{\Sigma}^{-1}. \end{aligned} \tag{3.8}$$

The DMD modes and eigenvalues, containing the spatial and temporal information, are then obtained by computing the eigendecomposition of the linear map as

$$\tilde{\mathbf{M}}\mathbf{W} = \mathbf{W}\mathbf{\Lambda}, \tag{3.9}$$

where columns of $\mathbf{W} \in \mathbb{C}^{k \times k}$ are eigenvectors ϕ_j and $\mathbf{\Lambda} \in \mathbb{C}^{k \times k}$ is a diagonal matrix containing the corresponding eigenvalues λ_j. The continuous-time eigenvalues are given by $\omega_j = \log(\lambda_j)/\Delta t$. Finally, we

may reconstruct the eigendecomposition of $\hat{\mathbf{M}}$ from $\mathbf{W}$ and $\mathbf{\Lambda}$. Specifically, it follows from the similarity transform, that the eigenvalues of $\hat{\mathbf{M}}$ are given by $\mathbf{\Lambda}$. Further, the DMD modes (eigenvectors) are given by columns of $\mathbf{\Phi} \in \mathbb{C}^{m \times k}$

$$\mathbf{\Phi} = \mathbf{X}_R \mathbf{V} \mathbf{\Sigma}^{-1} \mathbf{W}. \tag{3.10}$$

Note that Equation (3.10) from Tu et al. (2014) differs from the formula $\mathbf{\Phi} = \mathbf{U}\mathbf{W}$ from Schmid (2010), although these will tend to converge if $\mathbf{X}_L$ and $\mathbf{X}_R$ have the same column spaces. Finally, the left snapshot matrix $\mathbf{X}_L$ can be approximately reconstructed by noting that the snapshots can be represented as a linear combination (Kutz et al., 2016a)

$$\mathbf{x}_t \approx \sum_{j=1}^{k} b_j \phi_j \lambda_j^t. \tag{3.11}$$

Since the amplitudes b_i are time independent $\mathbf{x}_0$ reduces to

$$\mathbf{x}_0 \approx \sum_{i=1}^{k} b_i \phi_i = \mathbf{\Phi}\mathbf{b}. \tag{3.12}$$

The parameter vector $\mathbf{b} \in \mathbb{C}^k$ can be estimated by using simply the method of least squares. To summarize, the dynamic mode decomposition yields the following low-rank factorization of the left snapshot sequence

$$\boldsymbol{X}_L \approx \mathbf{\Phi}\mathbf{B}\mathcal{V} = \begin{pmatrix} \phi_{11} & \phi_{1p} & \cdots & \phi_{1k} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{i1} & \phi_{ip} & \cdots & \phi_{ik} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{m1} & \phi_{mp} & \cdots & \phi_{mk} \end{pmatrix} \begin{pmatrix} b_1 & & & & \\ & \ddots & & & \\ & & b_p & & \\ & & & \ddots & \\ & & & & b_k \end{pmatrix} \begin{pmatrix} 1 & \lambda_1 & \cdots & \lambda_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_p & \cdots & \lambda_p^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_k & \cdots & \lambda_k^{n-1} \end{pmatrix}, \tag{3.13}$$

where $\mathbf{B} \in \mathbb{C}^{k \times k}$ is a diagonal matrix of the amplitudes, and $\mathcal{V} \in \mathbb{C}^{k \times n-1}$ is the Vandermonde matrix of the eigenvalues. The Vandermonde matrix describes the temporal evolution of the DMD modes. Specifically, each DMD mode, retrieved by the DMD, corresponds to a single (distinct) frequency. The computational steps are summarized in Algorithm 3.

Algorithm 3 Dynamic Mode Decomposition.
Given a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ containing the snapshots, and a target rank k, this procedure computes the approximate low-rank dynamic mode decomposition, where columns of $\mathbf{\Phi} \in \mathbb{C}^{m \times k}$ are the DMD modes, $\mathbf{b} \in \mathbb{C}^k$ are the amplitudes, and $\mathcal{V} \in \mathbb{C}^{k \times n}$ is the Vandermonde matrix describing the temporal evolution. It is required that $m \geq n$, integer $k \geq 1$.

function $[\mathbf{\Phi}, \mathbf{b}, \mathcal{V}] = \texttt{dmd}(\mathbf{X}, k)$

(1)	$\mathbf{X}_L, \mathbf{X}_R = \mathbf{X}$	Left/right snapshot sequence.
(2)	$\mathbf{U}, \mathbf{\Sigma}, \mathbf{V} = \texttt{svd}(\mathbf{X}_L, k)$	Truncated SVD.
(3)	$\tilde{\mathbf{M}} = \mathbf{U}^\top \mathbf{X}_R \mathbf{V} \mathbf{\Sigma}^{-1}$	Least squares fit.
(4)	$\mathbf{W}, \mathbf{\Lambda} = \texttt{eig}(\tilde{\mathbf{M}})$	Eigenvalue decomposition.
(5)	$\mathbf{\Phi} \leftarrow \mathbf{X}_R \mathbf{V} \mathbf{\Sigma}^{-1} \mathbf{W}$	Compute full-state modes $\mathbf{\Phi}$.
(6)	$\mathbf{b} = \texttt{lstsq}(\mathbf{\Phi}, \mathbf{x}_0)$	Comp. amplitudes with init. cond. $\mathbf{x}_0$.
(7)	$\mathcal{V} = \texttt{vander}(\texttt{diag}(\mathbf{\Lambda}))$	Vandermonde matrix (optional).

Remark 1. An alternative to the predefined target-rank k is the recent hard-thresholding algorithm of Gavish and Donoho (2014). This method can be combined with step 2 to automatically determine the optimal target-rank.

3.3 Compressed DMD

3.3.1 Conceptual Overview

Brunton et al. (2015) proposed the compressed dynamic mode decomposition to overcome of the computational challenges of computing the dynamic mode decomposition. Specifically, they showed that the dominant DMD modes and eigenvalues can be obtained from massively undersampled or compressed data. The method was originally devised to reconstruct high-dimensional, full-resolution DMD modes from sparse, and undersampled measurements by leveraging compressed sensing. However, it was quickly realized that if full-state measurements are available, many of the computationally expensive steps in DMD may be computed on a compressed representation of the data, providing dramatic computational savings. While Brunton et al. (2015) embedded their algorithm into the theory of compressed sensing, later Erichson et al. (2016a) noticed

that the algorithm can be motived using the concept ot matrix sketching as well. The procedure is sketched in Figure 3.2.

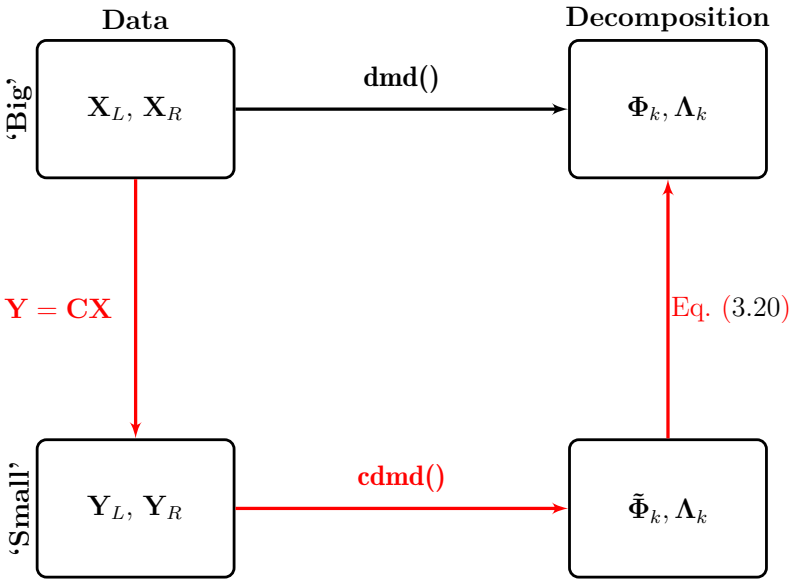


Fig. 3.2 Conceptual architecture of the compressed dynamic mode decomposition (cDMD). First, a smaller matrix $\mathbf{Y} = \mathbf{CX}$ is computed using a random test matrix $\mathbf{C}$. Then, the DMD is efficiently computed using this smaller (low-dimensional) snapshot matrices. Finally, the dominant DMD modes $\mathbf{\Phi}_k$ may be reconstructed from the approximate modes $\tilde{\mathbf{\Phi}}_k$ by the expression in Eq. (3.20).

Matrix sketching forms a smaller matrix, a so called sketch, from a small number of random linear combinations of the rows of the high-dimensional input matrix (Liberty, 2013).[1] While, compressed sensing provides a theoretical framework to reconstruct high-dimensional signals from low-dimensional measurement (Baraniuk, 2007; Candès and Wakin, 2008; Donoho, 2006), matrix sketching provides a framework to construct low-dimensional representations from high-dimensional data (Woodruff, 2014). The latter case is often the more relevant in the context of the DMD. This is because the full data are available. Hence, we favor matrix sketching, because this framework poses less restrictive assumptions on the input matrix than the compressed sensing framework.

[1]Note, that this is somewhat different from the probabilistic framework outlined in Chapter 1, which forms a small number of random linear combinations of the columns of the input matrix.

3.3.2 Compressed Algorithm

The compressed DMD algorithm starts by constructing a smaller matrix, illustrated in Figure 3.3. Specifically, the sketch is formed from a small

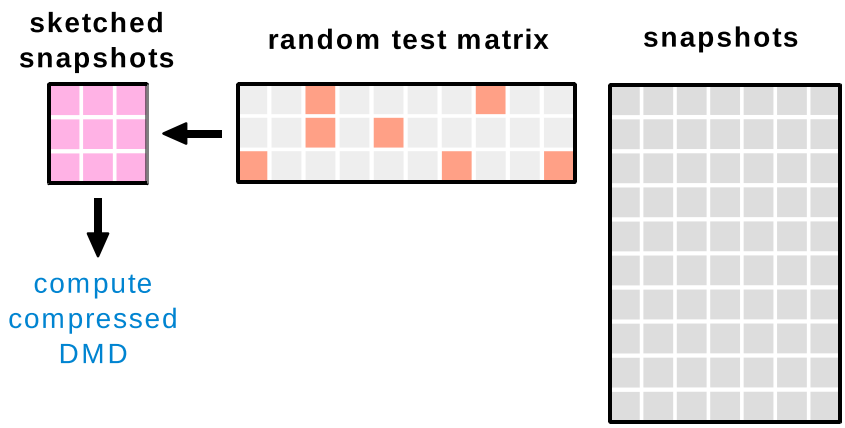


Fig. 3.3 Video compression using a sparse measurement matrix. The compressed matrix faithfully captures the essential spectral information of the video.

number of random linear combinations of the rows of the left and right snapshot matrices as follows

$$\mathbf{Y}_L = \mathbf{C}\mathbf{X}_L, \quad \mathbf{Y}_R = \mathbf{C}\mathbf{X}_R. \tag{3.14}$$

$\mathbf{C} \in \mathbb{R}^{c \times m}$ is a random test matrix, and c denotes the number of samples. Instead of sketching the left and right snapshot sequence individually, it is computationally more efficient to compute a sketch of $\mathbf{X}$ directly. Then, the resulting sketch can be split into the two low-dimensional overlapping sets of data

$$\mathbf{Y}_L = \begin{bmatrix} | & | & & | \\ \mathbf{y}_0 & \mathbf{y}_1 & \cdots & \mathbf{y}_{n-1} \\ | & | & & | \end{bmatrix}, \quad \mathbf{Y}_R = \begin{bmatrix} | & | & & | \\ \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_n \\ | & | & & | \end{bmatrix}. \tag{3.15}$$

$\mathbf{Y}_L \in \mathbb{R}^{c \times n}$ and $\mathbf{Y}_R \in \mathbb{R}^{c \times n}$. If m is large, then the random test matrix $\mathbf{C}$ becomes expensive to construct as well as the computation of the sketch is costly. However, if the information are uniformly distributed across the input data, then the low-dimensional sequences of snapshots can be simply constructed by randomly selecting rows, which leads to an

computationally efficient algorithm. Note, that in order to compute the first k dominant DMD modes and eigenvalues the sampling parameter c is required to be larger then k. However, the disadvantage of the method is that there is no good heuristic to chose an optimal number of samples c. But, since random row sampling is cheap we chose in practice (depended on the problem) simply a reasonable large value for c. Once the compressed snapshot matrices are obtained, the algorithm proceeds similar to the previous described deterministic DMD algorithm.

First, the truncated singular value decomposition of the compressed left snapshot matrix is computed

$$\mathbf{Y}_L = \mathbf{U}_\mathbf{Y} \mathbf{\Sigma}_\mathbf{Y} \mathbf{V}_\mathbf{Y}^\top, \tag{3.16}$$

where the matrices $\mathbf{U} \in \mathbb{R}^{c \times k}$, and $\mathbf{V} \in \mathbb{R}^{n \times k}$ are the truncated left and right singular vectors. The diagonal matrix $\mathbf{\Sigma} \in \mathbb{R}^{k \times k}$ has the corresponding singular values as entries. Here k is the target rank of the truncated SVD approximation. Note that the subscript $\mathbf{Y}$ is included to explicitly denote computations involving the compressed data $\mathbf{Y}$. Then, the linear map $\mathbf{M}_\mathbf{Y}$ which approximately relates the left and right snapshot matrix is computed similar to the previously described deterministic algorithm. Specifically, we compute the low-dimensional model projected onto the left singular vectors

$$\tilde{\mathbf{M}}_\mathbf{Y} = \mathbf{U}_\mathbf{Y}^\top \hat{\mathbf{M}}_\mathbf{Y} \mathbf{U}_\mathbf{Y} \tag{3.17a}$$

$$= \mathbf{U}_\mathbf{Y}^\top \mathbf{Y}_R \mathbf{V}_\mathbf{Y} \mathbf{\Sigma}_\mathbf{Y}^{-1}. \tag{3.17b}$$

Since this is a similarity transform, the eigenvectors and eigenvalues can be obtained from the eigendecomposition of $\tilde{\mathbf{M}}_\mathbf{Y} \in \mathbb{R}^{k \times k}$

$$\tilde{\mathbf{M}}_\mathbf{Y} \mathbf{W}_\mathbf{Y} = \mathbf{W}_\mathbf{Y} \mathbf{\Lambda}_\mathbf{Y}, \tag{3.18}$$

where columns of $\mathbf{W}_\mathbf{Y} \in \mathbb{C}^{k \times k}$ are eigenvectors $\boldsymbol{\phi}_j$ and $\mathbf{\Lambda}_\boldsymbol{Y} \in \mathbb{C}^{k \times k}$ is a diagonal matrix containing the corresponding eigenvalues λ_j. The similarity transform implies that $\mathbf{\Lambda} \approx \mathbf{\Lambda}_\boldsymbol{Y}$. The compressed DMD modes

$\mathbf{\Phi_Y} \in \mathbb{C}^{c\times k}$ are consequently given by

$$\mathbf{\Phi_Y} = \mathbf{Y}_R \mathbf{V_Y} \mathbf{\Sigma_Y}^{-1} \mathbf{W_Y}. \tag{3.19}$$

Finally, the full DMD modes $\mathbf{\Phi} \in \mathbb{C}^{m\times k}$ are recovered using

$$\mathbf{\Phi} = \mathbf{X}_R \mathbf{V_Y} \mathbf{\Sigma_Y}^{-1} \mathbf{W_Y}. \tag{3.20}$$

Note that the compressed DMD modes in Equation (3.20) make use of the full data $\mathbf{X}_R$ as well as the linear transformations obtained using the compressed data $\mathbf{Y}_L$ and $\mathbf{Y}_R$. The expensive SVD on $\mathbf{X}_L$ is bypassed, and it is instead performed on $\mathbf{Y}_L$. The computational steps are summarized in Algorithm 4.

Algorithm 4 Compressed Dynamic Mode Decomposition.
Given a matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$ containing the snapshots, this procedure computes the approximate low-rank dynamic mode decomposition, where columns of $\mathbf{\Phi} \in \mathbb{C}^{m\times k}$ are the DMD modes, $\mathbf{b} \in \mathbb{C}^k$ are the amplitudes, and $\mathcal{V} \in \mathbb{C}^{k\times n}$ is the Vandermonde matrix describing the temporal evolution. The procedure can be controlled by the two parameters k and c, the target rank and the number of samples respectively. It is required that $m \geq n$, integer $k, c \geq 1$ and $k \ll n$ and $c \geq k$.

function $[\mathbf{\Phi}, \mathbf{b}, \mathcal{V}] = \texttt{cdmd}(\mathbf{X}, k, c)$

(1)	$\mathbf{X}_L, \mathbf{X}_R = \mathbf{X}$	Left/right snapshot sequence.
(2)	$\mathbf{C} = \texttt{rand}(c, m)$	Draw $c \times m$ test matrix.
(3)	$\mathbf{Y}_L, \mathbf{Y}_R = \mathbf{CD}$	Sketch input matrix.
(4)	$\mathbf{U}, \mathbf{\Sigma}, \mathbf{V} = \texttt{svd}(\mathbf{Y}_L, k)$	Truncated SVD.
(6)	$\tilde{\mathbf{M}} = \mathbf{U}^\top \mathbf{Y}_R \mathbf{V} \mathbf{\Sigma}^{-1}$	Least squares fit.
(7)	$\mathbf{W}, \mathbf{\Lambda} = \texttt{eig}(\tilde{\mathbf{M}})$	Eigenvalue decomposition.
(8)	$\mathbf{\Phi} \leftarrow \mathbf{X}_R \mathbf{V} \mathbf{\Sigma}^{-1} \mathbf{W}$	Compute full-state modes $\mathbf{\Phi}$.
(9)	$\mathbf{b} = \texttt{lstsq}(\mathbf{\Phi}, \mathbf{x}_0)$	Comp. amplitudes with init. cond. $\mathbf{x}_0$.
(10)	$\mathcal{V} = \texttt{vander}(\texttt{diag}(\mathbf{\Lambda}))$	Vandermonde matrix (optional).

Remark 2. The computational performance heavily depends on the measurement matrix used to construct the compressed matrix. For a practical implementation sparse or single pixel measurements (random row selection) are favored, see Erichson et al. (2016a) for a detailed discussion.

3.4 Randomized DMD

3.4.1 Conceptual Overview

The compressed dynamic mode decomposition presented in the previous section is a highly computationally efficient algorithm to compute the dynamic modes. However, the algorithm is less precise[2], and relatively sensitive to noise. In the following we present a novel randomized algorithm, which is as computationally efficient, but more robust. The advantage is that the approximation error can be controlled via both oversampling and power iterations. The key difference to the compressed DMD algorithm lies in the concept for obtaining the smaller (low-dimensional) sequence of snapshots. Therefor, we utilize the probabilistic framework introduced in Chapter 1. Hence, instead of premultiplying the data matrix by a measurement matrix $\mathbf{C} \in \mathbb{R}^{c \times m}$, we now postmultiply the data matrix by a random test matrix $\mathbf{\Omega} \in \mathbb{R}^{n \times l}$ in order to sample the range of high-dimensional sequence of snapshots. Then, a natural basis is computed and the data are projected to low-dimensional space. The idea is depicted schematically in Figure 3.4.

3.4.2 Randomized Algorithm

Given a sequence of snapshots $\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^m$, we first compute the near-optimal basis $\mathbf{Q} \in \mathbb{R}^{m \times l}$. Then, we project the data onto the low-dimensional space, so that we obtain the low-dimensional sequence of snapshots $\mathbf{b}_0, \mathbf{b}_1, ..., \mathbf{b}_n \in \mathbb{R}^l$. Here, $l = k + p$, where k denotes the desired target rank, and p the oversampling parameter. Specifically, we first sample the range as

$$\mathbf{Y} = \mathbf{X}\mathbf{\Omega}, \tag{3.21}$$

where $\mathbf{\Omega}$ is the random test matrix. Then, the orthonormal basis $\mathbf{Q} \in \mathbb{R}^{m \times l}$ is obtained via the QR-decomposition $\mathbf{Y} = \mathbf{QR}$, such that

$$\mathbf{X} \approx \mathbf{Q}\mathbf{Q}^\top \mathbf{X}$$

[2]Note that random sampling introduces a slight error.

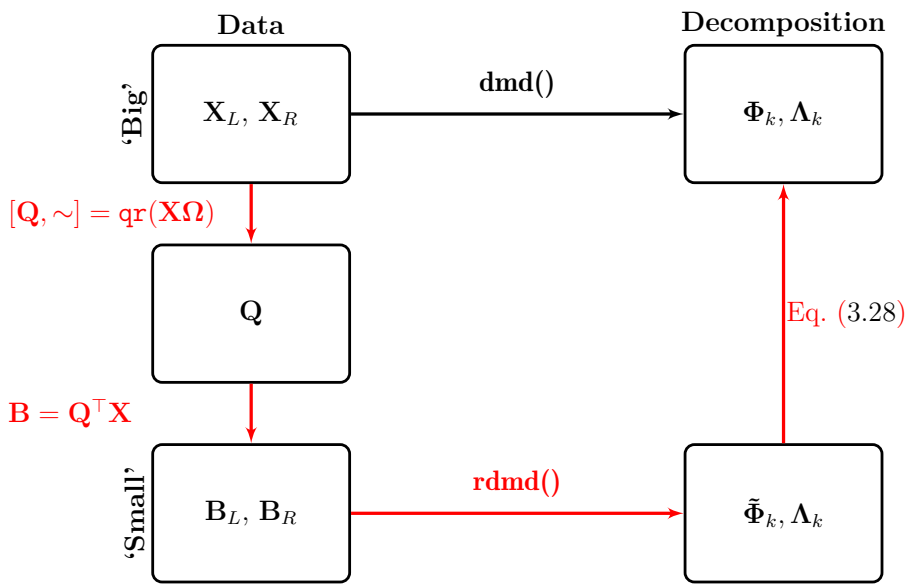


Fig. 3.4 Conceptual architecture of the randomized dynamic mode decomposition. First, a natural basis is computed in order to derive the smaller snapshot matrices $\mathbf{B}_L$ and $\mathbf{B}_R$. Then, the DMD is efficiently computed using this smaller matrices. Finally, the dominant DMD modes $\mathbf{\Phi}_k$ may be reconstructed from the approximate DMD modes by the expression in Equation (3.28).

is satisfied. Finally, $\mathbf{X}$ is projected to low-dimensional space

$$\mathbf{B} = \mathbf{Q}^\top\mathbf{X},$$

where $\mathbf{B} \in \mathbb{R}^{l\times n}$.

Once the low-dimensional matrix $\mathbf{B}$ is obtained, the standard procedure to obtain the DMD modes can be applied. Therefore, we first separate the data into two overlapping sets of data

$$\mathbf{B}_L = \begin{bmatrix} | & | & & | \\ \mathbf{b}_0 & \mathbf{b}_1 & \cdots & \mathbf{b}_{n-1} \\ | & | & & | \end{bmatrix}, \quad \mathbf{B}_R = \begin{bmatrix} | & | & & | \\ \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_n \\ | & | & & | \end{bmatrix}, \tag{3.22}$$

where $\mathbf{B}_L \in \mathbb{R}^{l\times n}$ and $\mathbf{B}_R \in \mathbb{R}^{l\times n}$. Then, the estimator for the linear map $\mathbf{M_Y}$ is defined as

$$\hat{\mathbf{M}}_{\mathbf{B}} = \mathbf{B}_R\mathbf{B}_L^{\dagger} \tag{3.23a}$$

$$= \mathbf{B}_R\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^{\top}, \tag{3.23b}$$

The pseudo-inverse $\mathbf{B}_L^{\dagger}$ is computed using the SVD:

$$\mathbf{B}_L = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\top}, \tag{3.24}$$

where the matrices $\mathbf{U} \in \mathbb{R}^{k\times k}$, and $\mathbf{V} \in \mathbb{R}^{n\times k}$ are the truncated left and right singular vectors. The diagonal matrix $\mathbf{\Sigma} \in \mathbb{R}^{k\times k}$ has the corresponding singular values as entries. Here k is the target-rank of the truncated SVD approximation to $\mathbf{B}_L$. As in the standard DMD algorithm, we typically do not compute the matrix $\hat{\mathbf{M}}_{\mathbf{B}}$, but instead compute the low-dimensional model projected onto the left singular vectors

$$\tilde{\mathbf{M}}_{\mathbf{B}} = \mathbf{U}^*\hat{\mathbf{M}}\mathbf{U} \tag{3.25a}$$

$$= \mathbf{U}^*\mathbf{B}_R\mathbf{V}\mathbf{\Sigma}^{-1}. \tag{3.25b}$$

Subsequently, the eigenvectors and eigenvalues can be obtained from the eigendecomposition of $\tilde{\mathbf{M}}_{\mathbf{B}}$

$$\tilde{\mathbf{M}}_{\mathbf{B}}\mathbf{W}_{\mathbf{B}} = \mathbf{W}_{\mathbf{B}}\mathbf{\Lambda}, \tag{3.26}$$

where columns of $\mathbf{W}_{\mathbf{B}}$ are eigenvectors $\boldsymbol{\phi}_j$ and $\mathbf{\Lambda}_{\boldsymbol{B}}$ is a diagonal matrix containing the corresponding eigenvalues λ_j. The approximate randomized DMD modes are consequently given by

$$\mathbf{\Phi}_{\mathbf{B}} = \mathbf{B}_R\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{W}_{\mathbf{B}}. \tag{3.27}$$

Finally, the near-optimal high-dimensional DMD modes are recovered as

$$\mathbf{\Phi} = \mathbf{Q}\mathbf{\Phi}_{\mathbf{B}} = \mathbf{Q}\mathbf{B}_R\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{W}_{\mathbf{B}}. \tag{3.28}$$

Algorithm 5 is summarizing the computational steps.

Algorithm 5 Randomized Dynamic Mode Decomposition.
Given a snapshot matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$, this algorithm computes the near-optimal low-rank dynamic mode decomposition, where k denotes the target rank. The approximation quality can be controlled via the oversampling parameter p, and the number of power iterations denoted by the parameter q.

function $[\mathbf{\Phi}, \mathbf{b}, \mathcal{V}] = \texttt{rdmd}(\mathbf{X}, k, p, q)$

(1)	$l = k + p$	Slight oversampling.
(2)	$\mathbf{\Omega} = \texttt{rand}(n, l)$	Generate random matrix.
(3)	$\mathbf{Y} = \mathbf{X}\mathbf{\Omega}$	Compute sampling matrix.
(4)	**for** $j = 1, \ldots, q$	Subspace iterations (optional).
(5)	$[\mathbf{Q}, \sim] = \texttt{qr}(\mathbf{Y})$	
(6)	$[\mathbf{Z}, \sim] = \texttt{qr}(\mathbf{X}^\top \mathbf{Q})$	
(7)	$\mathbf{Y} = \mathbf{X}\mathbf{Z}$	
(8)	**end for**	
(9)	$[\mathbf{Q}, \sim] = \texttt{qr}(\mathbf{Y})$	Orthonormalize sampling matrix.
(10)	$\mathbf{B} = \mathbf{Q}^\intercal \mathbf{X}$	Project matrix to smaller space.
(11)	$\mathbf{B}_L, \mathbf{B}_R = \mathbf{B}$	Left/right small snapshot sequence.
(12)	$\mathbf{U}, \mathbf{\Sigma}, \mathbf{V} = \texttt{svd}(\mathbf{B}_L, k)$	Truncated SVD.
(13)	$\tilde{\mathbf{M}} = \mathbf{U}^\top \mathbf{B}_R \mathbf{V} \mathbf{\Sigma}^{-1}$	Least squares fit.
(14)	$\mathbf{W}, \mathbf{\Lambda} = \texttt{eig}(\tilde{\mathbf{M}})$	Eigenvalue decomposition.
(15)	$\mathbf{\Phi} \leftarrow \mathbf{Q}\mathbf{B}_R \mathbf{V} \mathbf{\Sigma}^{-1} \mathbf{W}$	Compute full-state modes $\mathbf{\Phi}$.
(16)	$\mathbf{b} = \texttt{lstsq}(\mathbf{\Phi}, \mathbf{x}_0)$	Comp. amplitudes with init. cond. $\mathbf{x}_0$.
(17)	$\mathcal{V} = \texttt{vander}(\texttt{diag}(\mathbf{\Lambda}))$	Vandermonde matrix (optional).

Justification

In the following we outline some justification for the proposed algorithm. Let us assume that we have an orthogonal basis $\mathbf{Q}$ so that

$$\mathbf{X}_L \approx \mathbf{Q}\mathbf{Q}^\top \mathbf{X}_L \tag{3.29a}$$

$$\mathbf{X}_R \approx \mathbf{Q}\mathbf{Q}^\top \mathbf{X}_R, \tag{3.29b}$$

is satisfied. Now, we show that the eigendecomposition of $\tilde{\mathbf{M}}$ is approximately equivalent to the eigendecomposition of $\tilde{\mathbf{M}}_{\mathbf{B}}$. Starting with

Equation (3.23a), we have that

$$\mathbf{W_B}\boldsymbol{\Lambda_B}\mathbf{W_B}^\top = \hat{\mathbf{M}}_\mathbf{B} = \mathbf{B}_R\mathbf{B}_L^\dagger = \mathbf{B}_R\mathbf{V}\boldsymbol{\Sigma}^{-1}\tilde{\mathbf{U}}^\top. \tag{3.30}$$

Now, note that we defined the projections to the low-dimensional subspace as $\mathbf{B}_L := \mathbf{Q}^\top\mathbf{X}_L$ and $\mathbf{B}_R := \mathbf{Q}^\top\mathbf{X}_R$. Hence, we yield the following equivalent expression of Equation (3.30) by substitution

$$\mathbf{W_B}\boldsymbol{\Lambda_B}\mathbf{W_B}^\top = (\mathbf{Q}^\top\mathbf{X}_R)(\mathbf{Q}^\top\mathbf{X}_L)^\dagger = (\mathbf{Q}^\top\mathbf{X}_R)\mathbf{V}\boldsymbol{\Sigma}^{-1}\tilde{\mathbf{U}}^\top. \tag{3.31}$$

Next, pre and post multiplying by the basis $\mathbf{Q}$ and $\mathbf{Q}^\top$ gives

$$\mathbf{Q}\mathbf{W_B}\boldsymbol{\Lambda_B}\mathbf{W_B}^\top\mathbf{Q}^\top = \mathbf{Q}(\mathbf{Q}^\top\mathbf{X}_R)(\mathbf{Q}^\top\mathbf{X}_L)^\dagger\mathbf{Q}^\top = \mathbf{Q}(\mathbf{Q}^\top\mathbf{X}_R)\mathbf{V}\boldsymbol{\Sigma}^{-1}\tilde{\mathbf{U}}^\top\mathbf{Q}^\top. \tag{3.32}$$

We note that $\mathbf{Q}\mathbf{Q}^\top\mathbf{X}_R \approx \mathbf{X}_R$ and that $\mathbf{V}\boldsymbol{\Sigma}^{-1}\tilde{\mathbf{U}}^\top\mathbf{Q}^\top \approx \mathbf{X}_L^\dagger$. Thus, it follows that

$$\hat{\mathbf{M}} = \mathbf{W}\boldsymbol{\Lambda}\mathbf{W}^\top \approx \mathbf{Q}\mathbf{W_B}\boldsymbol{\Lambda_B}\mathbf{W_B}^\top\mathbf{Q}^\top, \tag{3.33}$$

where $\mathbf{W} \approx \mathbf{Q}\mathbf{W_B}$ and $\boldsymbol{\Lambda} \approx \boldsymbol{\Lambda_B}$. It is then easy to verify that the same result holds for $\tilde{\mathbf{M}}_\mathbf{B} = \tilde{\mathbf{U}}^\top\hat{\mathbf{M}}_\mathbf{B}\tilde{\mathbf{U}}$, i.e., the linear map $\hat{\mathbf{M}}_\mathbf{B}$ projected onto the left singular vectors. Specifically, we have

$$\tilde{\mathbf{U}}^\top\mathbf{W_B}\boldsymbol{\Lambda_B}\mathbf{W_B}^\top\tilde{\mathbf{U}} = \tilde{\mathbf{U}}^\top\hat{\mathbf{M}}_\mathbf{B}\tilde{\mathbf{U}} = \tilde{\mathbf{U}}^\top\mathbf{B}_R\mathbf{B}_L^\dagger = \tilde{\mathbf{U}}^\top\mathbf{B}_R\mathbf{V}\boldsymbol{\Sigma}^{-1}. \tag{3.34}$$

Then we define $\tilde{\mathbf{W}}_\mathbf{B} := \tilde{\mathbf{U}}^\top\mathbf{W_B}$, and similar to the previous reasoning it follows that

$$\tilde{\mathbf{M}} = \mathbf{W}\boldsymbol{\Lambda}\mathbf{W}^\top \approx \mathbf{Q}\tilde{\mathbf{U}}\tilde{\mathbf{W}}_\mathbf{B}\boldsymbol{\Lambda_B}\tilde{\mathbf{W}}_\mathbf{B}^\top\tilde{\mathbf{U}}^\top\mathbf{Q}^\top. \tag{3.35}$$

Thus, we see that the eigenvalue decomposition of the small linear map $\tilde{\mathbf{M}}_\mathbf{B}$ can be used to approximate the eigenvalues and eigenvectors of the linear map $\tilde{\mathbf{M}}$ which relates the high-dimensional snapshots.

3.4.3 Blocked Randomized Algorithm

When dealing with massive fluid flows too big to read into the fast memory, the extension to distributed and parallel computing might be

inevitable. In particular, it might be necessary to distribute the data across processors which have no access to a shared memory to exchange information. Voronin and Martinsson (2015) proposed a blocked scheme to compute the QB decomposition in parallel. In the following, we briefly outline this scheme as it is beneficial for computing the dynamic mode decomposition as well. The basic idea is that a given high-dimensional sequence of snapshots $\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^m$ is subdivided into n smaller blocks along the rows. The submatrices can then be sent off to be processed in n independent streams of calculations. Here, n is assumed to be a power of two, and zero padding can be used in order to divide the data into blocks of the same size. Now, assume for simplicity that we aim to distribute the snapshot sequence across $n = 2$ processors. Using matrix notation, we have

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{bmatrix}. \tag{3.36}$$

Next, given a target-rank k, the QB decomposition (Equation 1.11) is computed on each block so that we yield

$$\mathbf{X} \approx \begin{bmatrix} \mathbf{Q}_1\mathbf{B}_1 \\ \mathbf{Q}_2\mathbf{B}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1 & 0 \\ 0 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \end{bmatrix}. \tag{3.37}$$

Now, the two smaller matrices $\mathbf{B}_i \in \mathbb{R}^{k \times n}$ can be collected and stacked together as

$$\mathbf{K}_1 = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \end{bmatrix}. \tag{3.38}$$

Subsequently, we compute the the QB decomposition of $\mathbf{K}_1 \in \mathbb{R}^{2k \times n}$ and obtain

$$\mathbf{K}_1 = \mathbf{Q}_{12}\mathbf{B}_{12}. \tag{3.39}$$

The small matrix $\mathbf{B}_{12} \in \mathbb{R}^{k \times n}$ can then be used to compute the dynamic mode decomposition as described above. Once the approximate dynamic modes $\mathbf{\Phi_B} \in \mathbb{R}^{k \times k}$ are obtained, the high-dimensional modes are reconstructed as follows

$$\mathbf{\Phi} \approx \begin{bmatrix} \mathbf{Q}_1 & 0 \\ 0 & \mathbf{Q}_2 \end{bmatrix} \mathbf{Q}_{12}\mathbf{\Phi_B}, \tag{3.40}$$

where $\mathbf{\Phi} \in \mathbb{R}^{m \times k}$. This scheme can be simply generalized to split the input matrix into more then just $n = 2$ blocks. In Python this algorithms can be implemented using the message passing interface (MPI), see for instance, Dalcin et al. (2005).

3.5 The DMDpack Package

We provide implementations of the presented deterministic and probabilistic dynamic mode decomposition algorithms in Python. Specifically, the **DMDpack** package provides the following core functions:

- Dynamic Mode Decomposition: **dmd()**.
- Compressed Dynamic Mode Decomposition: **cdmd()**.
- Randomized Dynamic Mode Decomposition: **rdmd()**.
- Randomized Singular Value Decomposition: **rsvd()**.

For installation instructions visit the GIT repository https://github.com/Benli11/DMDpack. See also the package manual, as all package functions are fully documented. In addition, the package comes with a set of test functions.

The interface of the functions are designed in **NumPy** style, the fundamental package for scientific computing with Python. All functions are based on routines provided by the **NumPy** and **SciPy** library. In particular these scientific libraries, provide several linear algebra functions, which rely on the LAPACK software package (Anderson et al., 1999). Moreover, **SciPy** provides by default the Intel Math Kernel Library (Intel MKL) accelerated high performance implementation of BLAS and LAPACK. In the following we will briefly introduce the **dmd()**, **cdmd()** and **rdmd()** functions. Note that the design of the interfaces is similar; the only difference are additional arguments, which can be passed to control the probabilistic algorithms.

3.5.1 The dmd() Function

The standard function for computing the DMD using the deterministic algorithm is the **dmd()** function, as outlined in Algorithm 3. The basic interface is

```
F, b, V, omega <- dmd(A, dt = 1, k=None,
  return_amplitudes=False, return_vandermonde=False, ...)
```

The first mandatory argument **A** passes the $m \times n$ snapshot matrix. The columns of the input matrix are assumed to contain the flattened snapshots, ordered in time. The algorithm is then separating the snapshot matrix into the left and right snapshot matrix using a pointer. The second argument specifies the time difference between the snapshots. This is used to rescale the DMD eigenvalues, and by default set to $\Delta t = 1$. The third argument **k** sets the target rank. If no target rank is provided the full dynamic mode decomposition is computed, i.e., all n dynamic modes are returned. Otherwise, the low-rank DMD is computed by truncating the singular value decomposition. Note, that with the additional argument **svd ={"rsvd", "partial", "truncated"}** the randomized singular value decomposition or a modern partial SVD algorithm can be selected. These algorithms are more efficient for computing the low-rank DMD than computing the truncated SVD. The arguments **return_amplitudes** and **return_vandermonde** can be used to compute in addition to the dynamic modes and eigenvalues the amplitudes and the Vandermonde matrix. By, default these are not computed. The function finally returns the following objects:

- **F**: $m \times n$ or $m \times k$ matrix containing the dynamic modes.
- **b**: (optional) n or k-dimensional vector which has the amplitudes, corresponding to the modes, as its entries.
- **V**: (optional) $n \times n$ or $k \times n$ Vandermonde matrix.
- **omega**: n or k-dimensional vector which contains the (scaled) DMD eigenvalues.

3.5.2 The cdmd() Function

The **cdmd()** function allows to compute the dynamic mode decomposition using the compressed algorithm. It follows Algorithm 4, and the basic interface is

```
F, b, V, omega <- cdmd(A, dt = 1, k=None, c=None,
  sdist="spixel", sf=0.9, return_amplitudes=False,
  return_vandermonde=False,  ...)
```

The first three arguments are similar to the **dmd()** function. The only difference is that this algorithm requires a target rank **k**, and it is supposed that $k \ll n$. Next, the argument **c** determines the number of samples used to sketch the input matrix. It is required that $c > k$. The disadvantage of this algorithm is that there is no clear guidance how to chose a good value. However, numerical results show that only a small number of samples relative to the ambient dimension of the snapshot matrix are required. By, default no explicit random test matrix is constructed. Rather c random rows are uniformly selected. This is computationally efficient, and achieves good results as long as the information are uniformly distributed across the matrix. When dealing with fluid flows or videos, this assumption is most often met. The argument **sdist** can alternatively set to **"unif"** in order to construct a dense random test matrix. However, this can become computational expensive if the dimension m is large. A third option is to construct a spare random test matrix using **"sparse"**. This option is a good trade-off between the dense test matrix and random row sampling. Using routines for fast sparse matrix multiplications allows to efficiently compute the sketch, while capturing more information then row sampling does. The sparsity factor can be controlled via the argument **"sf"**.

3.5.3 The rdmd() Function

The third function to compute the dynamic mode decomposition is the novel randomized DMD algorithm. It follows Algorithm 5, and the basic interface of the function is

```
F, b, V, omega <- rdmd(A, dt = 1, k=None, p=10, q=1,
  return_amplitudes=False, return_vandermonde=False, ...)
```

Again, the first three arguments are similar to the **dmd()** function. The argument **k**, is however, required to be $k \ll n$. The algorithm achieves a good performance gain, in particular, if $k < \frac{\min\{m,n\}}{2}$. The random test matrix is constructed by sampling from the uniform distribution. A future version will provide the option to use structured random test matrices as well. The arguments **p** and **q** are used to control the accuracy of the algorithm. The former parameter is used to oversample the basis, and is set by default to **p=10**. This setting guarantees a good basis with high

probability in general. The parameter **q** can be used to compute additional power iterations (subspace iterations). By default this parameter is set to **q=1** which shows a good performance in our numerical experiments. The default values show an optimal trade-off between speed and accuracy in standard situations. If the singular value spectrum of the input matrix is slowly decaying, more power iterations are desirable. However, in practice we have not encountered a situation which requires more then three subspace iterations ($q > 3$).

3.6 Numerical Results

In the following we present some numerical results demonstrating and comparing the performance of the proposed randomized algorithm for computing the dynamic mode decomposition. All computations are performed on a standard notebook with Intel Core i7-5500U 2.4GHz, and 8GB DDR3 memory.

3.6.1 Numerical Results

As a canonical example we use a fluid flow behind a cylinder (Noack et al., 2003). Specifically, the data are constructed as a sequence of 151 snapshots of fluid vorticity fields behind a stationary cylinder on an equispaced 449×199 grid.[3] The flow features a periodically shedding wake structure at Reynolds number $Re = 100$, and is inherently low-rank. Figure 3.5 shows three example snapshots of the fluid flow.

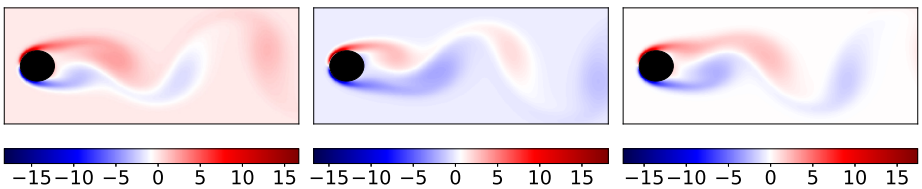


Fig. 3.5 Snapshots of the fluid flow behind a cylinder at time points $t = \{1, 50, 100\}$.

[3] The data are obtained by solving the two-dimensional Navier-Stokes equations using the publicly available code at https://github.com/cwrowley/ibpm, based on the immersed boundary projection method in a fast multi-domain solver (Colonius and Taira, 2008; Taira and Colonius, 2007).

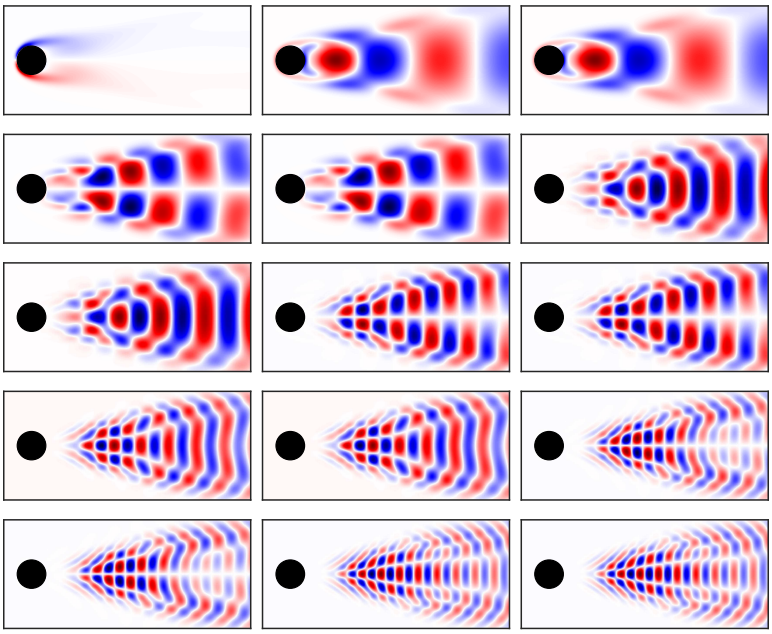
In the following we are interested to evaluate the different DMD algorithms in absence and in presence of additive white noise. In particular the latter case shows distinct differences between the algorithms.

Noise-Free Case

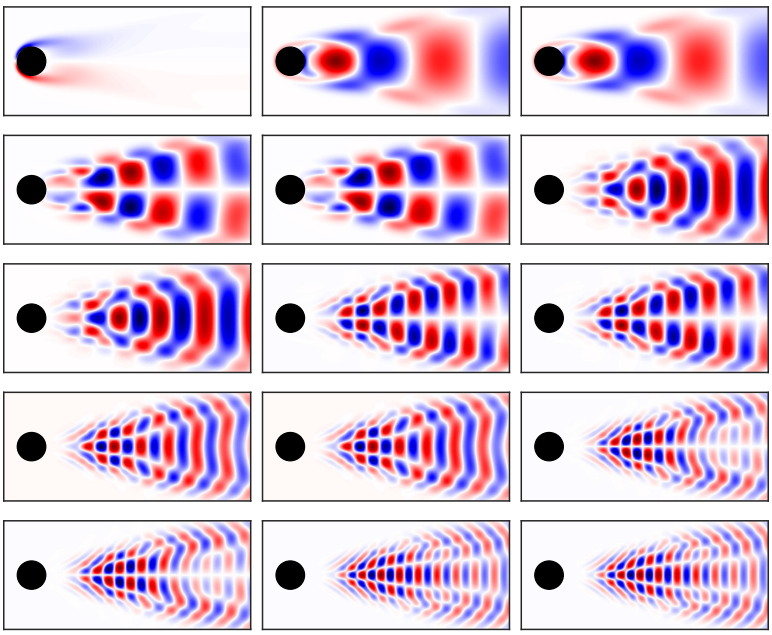
First, we compute the $k = 15$ dominant DMD modes and eigenvalues of the fluid flow. Here, we compute the randomized DMD using the recommended default settings for the oversampling parameter $p = 10$. However, knowing that the data are inherently low-rank we omit the computation of additional power iterations. Figure 3.6 shows the dominant DMD modes computed using both the deterministic and the randomized DMD algorithm. Note, that the data are not mean centered. Hence, the first mode does not change over time, and corresponds to the zero frequency DMD eigenvalue. The randomized algorithm faithfully reveals the coherent structures, while requiring considerably less computational resources. Specifically, a 6 fold speedup is achieved over the deterministic algorithm. The reconstruction error of the deterministic algorithm is $5.11e-03$, while the randomized algorithm achieves an error as low as $5.17e-03$. The dominant DMD eigenvalues are shown in Figure 3.7, and it can be seen that the deterministic DMD eigenvalues are faithfully captured by the randomized DMD eigenvalues as well as by the compressed DMD algorithm. Table 3.1 summarizes the results. The proposed randomized DMD algorithm outperforms the compressed DMD algorithm. The main advantage of the randomized DMD algorithm is that the approximation quality can be controlled via oversampling and power iterations. Further, the randomized algorithm is more precise, i.e., the standard deviation (SD) of the reconstruction error over 100 runs is smaller.

Table 3.1 Computational performance of the deterministic and probabilistic DMD algorithms (target rank is $k = 15$) in absence of noise. The results are averaged over 100 runs.

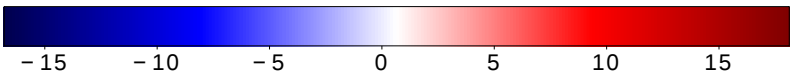
Method	**Parameters**	**Time (s)**	**Speedup**	**Error**	**SD**
Deterministic DMD	-	1.32	-	5.11e-03	-
Compressed DMD	**c=500**	0.28	4.7	8.07e-03	1.18e-03
Randomized DMD	**p=10, q=0**	0.21	6.3	5.17e-03	7.02e-05



(a) Deterministic DMD modes.



(b) Randomized DMD modes.



(c) Colorbar.

Fig. 3.6 Dominant 15 dynamic modes of a fluid flow behind a cylinder. By visual inspection there are no distinct differences between the DMD modes computed using the deterministic and randomized algorithm.

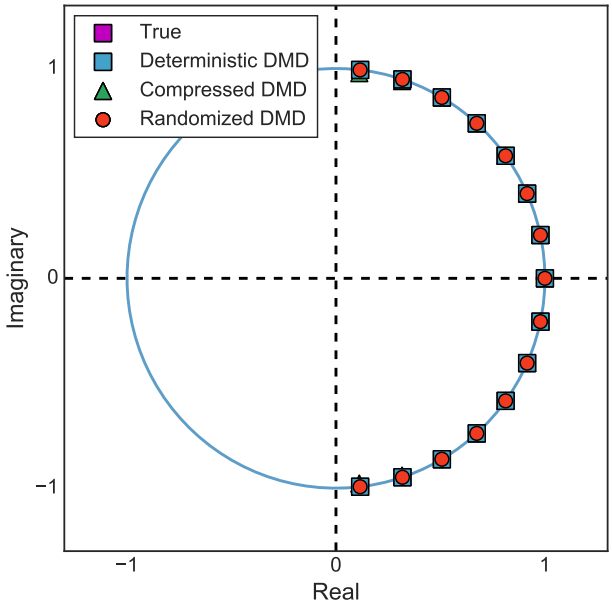
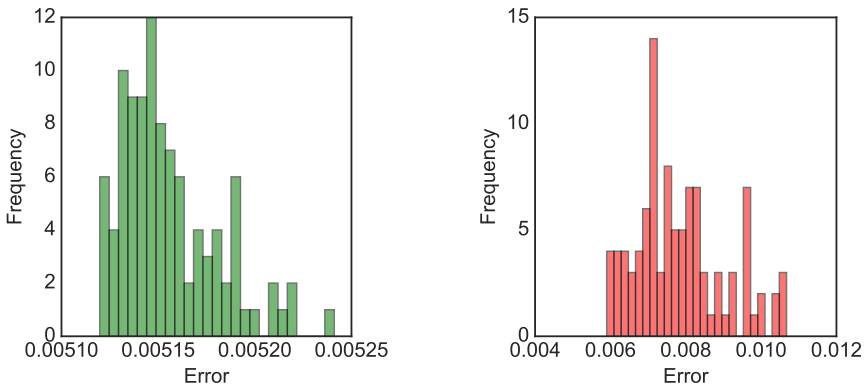


Fig. 3.7 DMD eigenvalues are faithfully captured by the randomized DMD eigenvalues.

Figure 3.8 shows a histogram of the relative reconstruction errors over 100 runs, which further contextualize the higher precision and accuracy of the randomized DMD algorithm compared to the compressed algorithm.



(a) Randomized DMD. (b) Compressed DMD.

Fig. 3.8 Relative error over 100 runs. The randomized algorithm shows to be more precise and accurate.

Noisy Case

Next, the analysis of the same flow is repeated in presence of noise. Specifically, the fluid flow is perturbed with additive white noise using a signal-to-noise (SNR) ratio of 10. Here, we compute the randomized DMD using two addition power iterations $q = 2$. This is because the added noise prevents the singular values from a vast decay. Figure 3.9 shows the first 60 singular values of the fluid flow in absence and presence of noise. The characteristic frequencies of flow oscillations occur in pairs, reflecting the complex-conjugate pairs of eigenvalues that define sine and cosine temporal dynamics. However, it can be seen that this structure is clearly effected by the additive white noise. Only the first fife complex-conjugate pairs remain in order. Figure 3.11 shows the

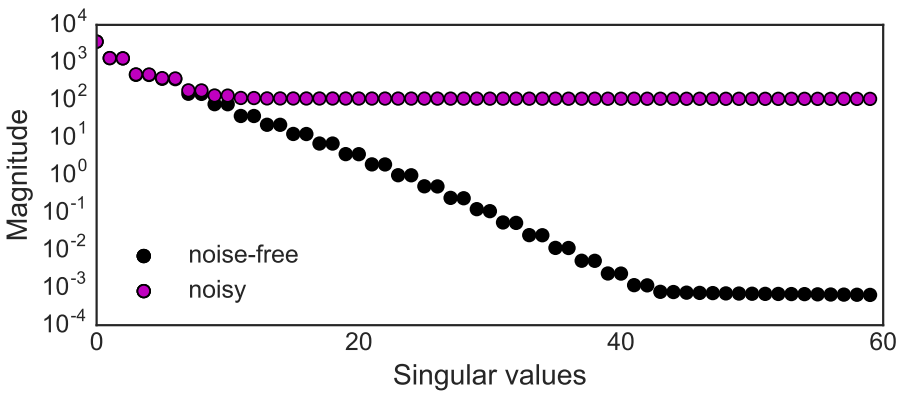


Fig. 3.9 Singular values of the fluid flow behind a cylinder in absence and presence of additive white noise (SNR=10). The complex-conjugate pairs reflecting the physics of the cylinder wake.

dominant DMD modes computed using both the deterministic and the randomized DMD algorithm. Here, the performance of the randomized algorithm is slightly worse than the deterministic algorithm. While, both algorithms are able to recover the structure of the first nine modes, the deterministic algorithm provides a better description of the remaining modes. Figure 3.10 shows the DMD eigenvalues, computed by using the different algorithms. Indeed, the performance of the different algorithms is distinct in the presence of noise. The deterministic algorithm performs best, capturing the first 11 eigenvalues, which are describing the physics of the fluid flow. The randomized DMD algorithm captures faithfully the

first 9 eigenvalues. The compressed DMD algorithm performs slightly worse, using the sampling parameter $c = 500$, however, the accuracy could be improved using a larger sampling parameter, e.g., $c = 5000$. But, this comes with higher computational costs. Table 3.2 summarizes the results. Again, the randomized DMD algorithm features the best trade-off between computational performance and approximation quality. The results show that the randomized DMD algorithm is more robust, and not as sensitive as the cDMD algorithm is to noise in the data. Further, the standard deviation (SD) of the reconstruction error over 100 runs indicates that the randomized algorithm is the more reliable algorithm.

Table 3.2 Computational performance of the deterministic and probabilistic DMD algorithms (target rank is $k = 15$) in presence of white noise (SNR=10). The results are averaged over 100 runs.

Method	**Parameters**	**Time (s)**	**Speedup**	**Error**	**SD**
Deterministic DMD	-	1.32	-	7.99e-02	-
Compressed DMD	**c=500**	0.28	4.7	1.84e-01	7.57e-03
Randomized DMD	**p=10, q=2**	0.35	3.7	8.43e-02	1.36e-03

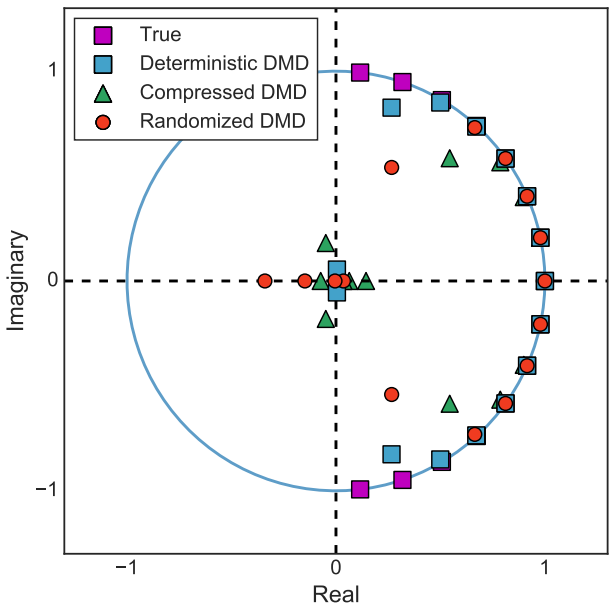
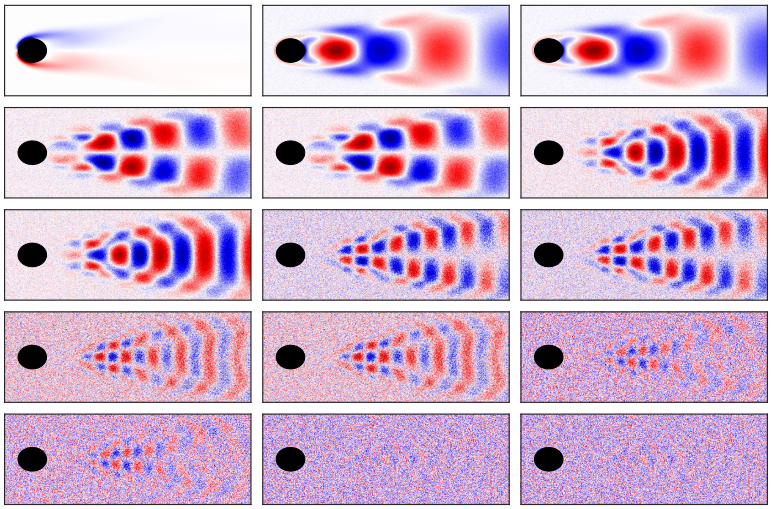
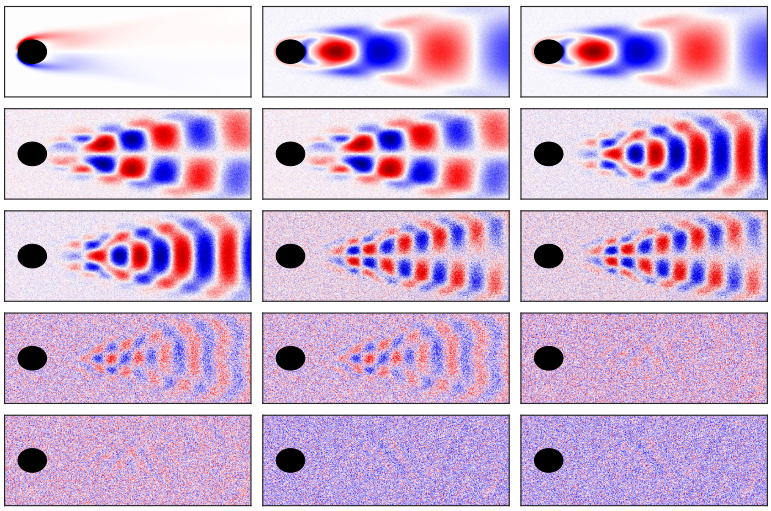


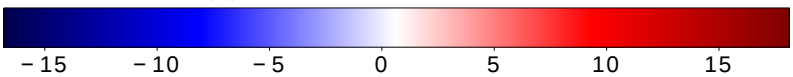
Fig. 3.10 DMD eigenvalues captured in presence of additive white noise (SNR=10). The deterministic algorithm achieves the best results. Randomized DMD, however, shows to be more robust than compressed DMD.



(a) Deterministic DMD modes.



(b) Randomized DMD modes.



(c) Colorbar.

Fig. 3.11 Dominant 15 dynamic modes of a fluid flow behind a cylinder in presence of additive white noise (SNR=10). By visual inspection there are no distinct differences between the first 9 DMD modes computed by the deterministic and randomized algorithm. However, the randomized algorithm shows difficulties to recover the following modes as good as the deterministic algorithm does.

Figure 3.12 shows the average reconstruction error of the different algorithms for varying signal-to-noise ratios. The target rank is set to

$k = 15$, again. The error of the randomized DMD algorithm computed with additional power iterations is converging towards the results of the deterministic algorithm. In contrast, the compressed DMD algorithm and the randomized DMD algorithm without power iterations exhibit a considerable bias. Note that the randomized algorithm with oversampling parameter $p = 10$ is performing nearly as good as the compressed algorithm with oversampling parameter $c = 500$.

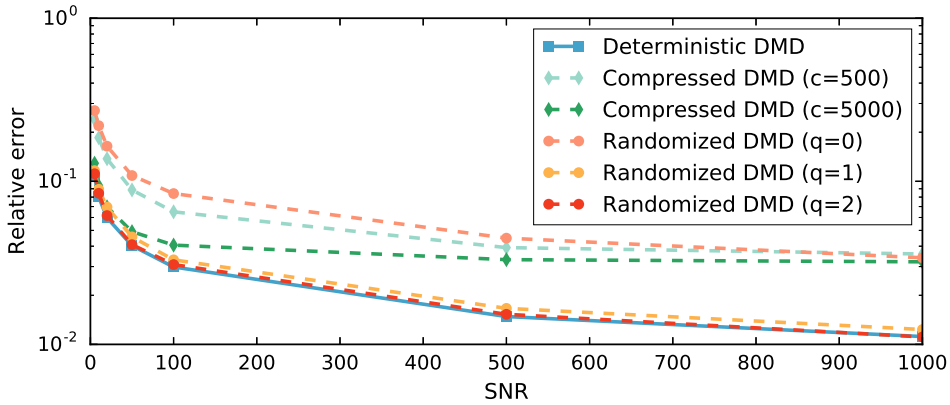
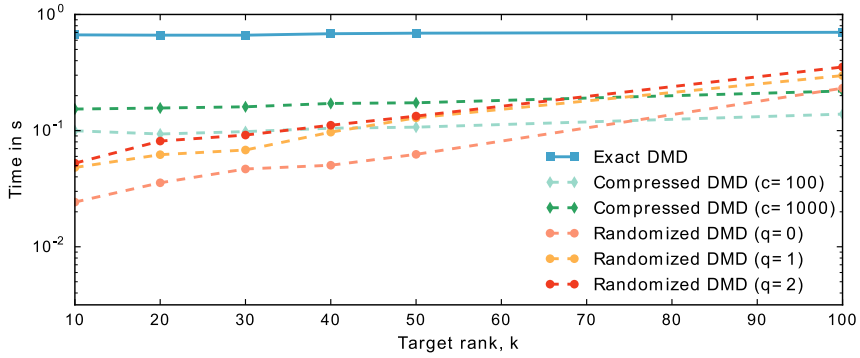


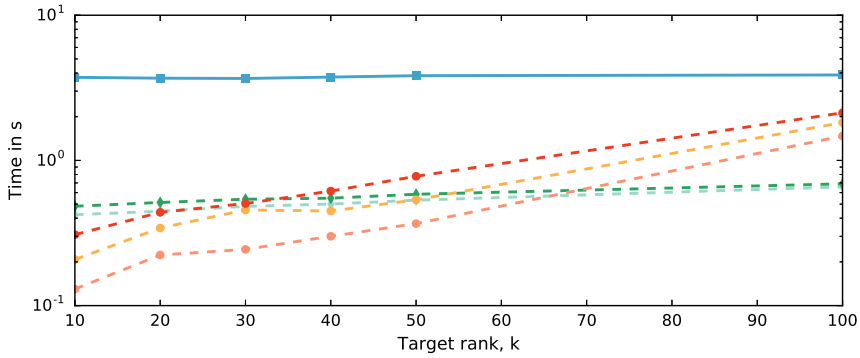
Fig. 3.12 Error of the different DMD algorithms averaged over 100 runs for varying signal-to-noise ratios. The randomized DMD algorithm with $q = \{1, 2\}$ power iterations achieves near-optimal results.

3.6.2 Computational Performance

The computational savings of the compressed and randomized DMD algorithms are mainly achieved by avoiding the expensive computation of the singular value decomposition. Figure 3.13 shows the computational performance of the algorithms for two matrices of different dimensions. Both the randomized and the compressed DMD algorithm can gain considerable savings for computing the low-rank approximation. Despite the additional computational costs of subspace iterations, the savings remain substantial.



(a) Matrix of dimension 20000×500.



(b) Matrix of dimension 100000×500.

Fig. 3.13 Computational performance of DMD algorithms. The probabilistic algorithms outperform the deterministic algorithms. For target ranks $k < 30$ the randomized DMD algorithm is even faster than the compressed DMD algorithm.

3.7 Conclusion

The dynamic mode decomposition is an emerging dimensionality reduction technique with a variety of applications across disciplines. However, the deterministic algorithm to compute the DMD modes and eigenvalues is computationally demanding and not suitable for real-time applications. In particular, in the area of fluid dynamics there is a need for highly efficient algorithms.

The proposed randomized algorithm can substantially decrease the computational load. In particular, the numerical results show that our algorithms has computational advantages over previously suggested probabilistic algorithms. In particular, the presented algorithms is computa-

tionally more efficient than the DMD algorithms using the randomized singular value decomposition (Bistrian and Navon, 2016; Erichson and Donovan, 2016). While, the compressed DMD algorithm (Brunton et al., 2015) is highly competitive in terms of the computational speed, the randomized algorithm is more robust and reliable. This is mainly, because the approximation quality of the randomized algorithms can be controlled using the concept of oversampling and the power scheme. The time complexity of the algorithm can be reduced even further from $O(mnk)$ to $O(mn \cdot \log(k))$ by using structured random test matrices to sample the range of the high-dimensional snapshot sequence. Moreover, the presented algorithm is embarrassingly parallel. Thus, the computational performance can benefit from a GPU accelerated implementation as demonstrated in Chapter 5. If we deal with massive data which are to big to fit into the fast memory, the algorithm can be reformulated using a blocked scheme. This allows to take advantage of distributed computing architectures.

Future research will investigate how other innovations around the dynamic mode decomposition can be embedded into the probabilistic framework as well.

Chapter 4

Dynamic Mode Decomposition for Background Modeling

"Prediction is very difficult, especially about the future."

— Niels Bohr

Note: The work described in this chapter was carried out in collaboration with Professors J. Nathan Kutz and Steven L. Brunton of University of Washington and Dr. Carl Donovan of the University of St Andrews. Parts of the work appeared in the Journal of Computer Vision and Image Understanding under the title: 'Randomized low-rank dynamic mode decomposition for motion detection' and in the Journal of Real-Time Image Processing under the title 'Compressed dynamic mode decomposition for background modeling'. My contributions involve conceptualizing the project idea, implementing the routines, running the simulations as well as writing the original draft.

4.1 Introduction

The demand for video processing is rapidly increasing, driven by greater numbers of sensors with greater resolution, new types of sensors, new collection methods and an ever wider range of applications. For example, video surveillance, vehicle automation or wild-life monitoring, with data gathered in visual/infra-red spectra or SONAR, from multiple sensors

being fixed or vehicle/drone-mounted etc. The overall result is an explosion in the quantity of high dimensional sensor data. At the most basic level, moving objects can be found in a video by subtracting the background and applying some thresholding, as illustrated in Figure 4.1. Background modeling is often the fundamental building block for more

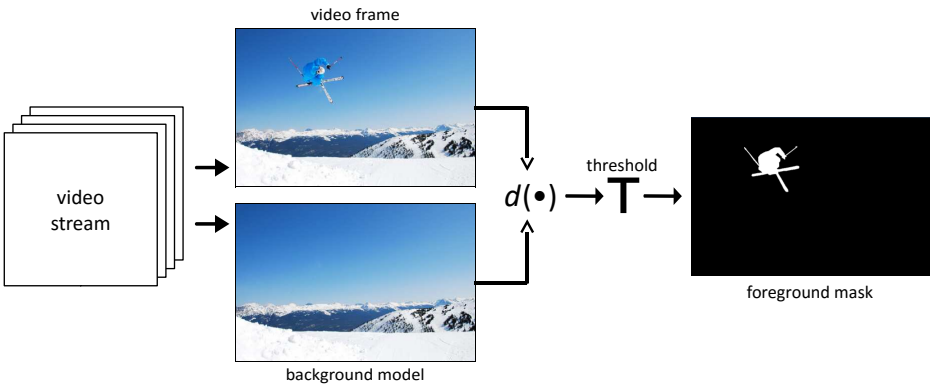


Fig. 4.1 Illustration of background subtraction

complex video processing and computer vision applications, e.g., object tracking or human behavior analysis. More generally, the task can be embedded into a framework of computational stages as illustrated in Figure 4.2.

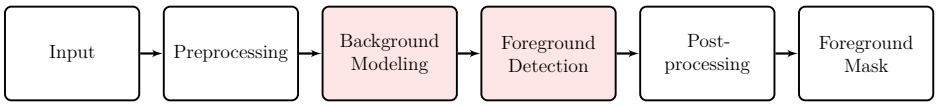


Fig. 4.2 Algorithmic flow of the computational stages involved in obtaining a foreground mask.

While there are many different types of sensors giving input data suitable for object extraction, we focus here on video data provided by static optical cameras. The first computational stage aims to preprocess the raw input video frames so that the data are in a suitable format for the subsequent stages. This involves simple tasks like reshaping the data, or converting a color into a grayscale image. However, the preprocessing stage can also involve more subtle tasks such as denoising or image registration between successive frames.

The next computational stage aims to find a suitable model for the background, or to update the previous estimated model. This is a challenging task in practice. Indeed, the list of challenges is significant and includes camera jitter, illumination changes, shadows and dynamic backgrounds. Hence, algorithms for background modeling are required to be both robust and adaptive. There is no single method currently available that is capable of handling all the challenges in real-time without suffering performance failures. Moreover, one of the great challenges in this field is to efficiently process high-resolution video streams, a task that is at the edge of performance limits for state-of-the-art algorithms.

Once the background model is obtained, foreground objects can be detected by background subtraction. Specifically, the distance of pixels between the video frame and the estimated background is computed. Then, outlying pixels are considered as foreground objects, i.e., pixels who exceed a pre-defined threshold. However, the challenge hereby is to distinguish between intensity changes related to moving foreground objects and intensity changes due to background noise, i.e., dynamic and complex backgrounds. Indeed, different threshold functions can yield distinctly different results as demonstrated by Benezeth et al. (2010).

Finally, post-processing techniques can be applied in order to improve the accuracy of the foreground mask. Standard post-processing techniques, for instance, are median or morphological filters. We refer to Sen-Ching and Kamath (2004) for a more detailed discussion of the computational stages involved in obtaining a suitable foreground mask for a video sequence.

Challenges and State-of-the-Art Methods

The task of finding a good estimate for the background model is difficult. This is, because surveillance videos face a large number of challenges:

- **Dynamic backgrounds:** The background in real scenarios if often highly dynamic. Objects like moving clouds, waving trees, water fountains or waves on a river behave like foreground objects, yet a good background model must be able to distinguish between

these and actual foreground objects in order to achieve a sufficient accuracy.

- **Illumination changes:** In long-time surveillance it is likely that illumination changes occur. This is, in particular, a problem in outdoor scenes were the light gradual changes over time. In indoor scenes illumination changes can be caused by a light switch, for instance. Thus, a 'good' background model is required to be flexible enough to adapt to these challenges.
- **Bad weather:** Detecting foreground objects in outdoor surveillance streams can be challenging, if the scene is disturbed by weather effects like snow or rain. It is even more critical, if the visibility is affected by fog or smog.
- **Shadows:** To distinguish between foreground objects and shadows is subtle. Hence, shadows are often misclassified as foreground objects, even by state-of-the-art machine vision algorithm.
- **Camouflage:** Foreground objects which share color features with the background are difficult to detect. This is, in particular, a problem in grayscale videos. Multi-feature algorithms are able to ease this challenge.
- **Night scenes:** An extreme form of illumination change is posed by night scenes. Specifically, the signal-to-noise ratio is very low in these video feeds. Thus, it is difficult to discriminate between foreground and background objects.
- **Camera jitter:** In theory, background models often assume a fixed (static) camera. In practice, however, the background model needs to deal with a certain amount of camera jitter, which can be caused by external physical effects.ns.

Given the importance of background modeling, a variety of mathematical methods and algorithms have been developed over the past decade to face these issues. Comprehensive overviews of traditional and state-of-the art methods are provided by Bouwmans (2011), Sobral and Vacavant (2014) and Xu et al. (2016).

Traditional methods aim to exploit the spatio-temporal variations of each pixel or regions of pixels. These methods include probabilistic models like Gaussian mixture-models and its variants (Pham et al., 2010; Shimada et al., 2006; Stauffer and Grimson, 1999; Zivkovic, 2004). Another interesting approach are fuzzy background models and nonparametric algorithm (Bouwmans, 2014). Among the best performing universal background subtraction algorithms are the ViBe algorithm (Barnich and Van Droogenbroeck, 2011), the IUTIS-5 algorithm (Bianco et al., 2015), and the PAWCS algorithm (St-Charles et al., 2015). These letter methods are highly specialized for the task of background subtraction and integrate a variety of innovations in order to estimate, maintain and update the background model.

Another (more general) recent direction is the research on decomposition into low-rank plus sparse matrices. The use of robust low-rank and sparse matrix decompositions for background modeling is discussed in detail by Bouwmans and Zahzah (2014) and Bouwmans et al. (2016a,b).

The most impressive background modeling performance is currently achieved by a supervised approach, which is based on a convolutional neural network (CNN) (Wang et al., 2016b). While this algorithm requires some human interaction (i.e. manually outlining a small number of moving objects), the algorithm achieves near human precision in detecting foreground objects.

Motivation and Overview

In the following, we advocate the method of randomized dynamic mode decomposition (rDMD) for background modeling. While the principal application of the DMD is in the area of fluid dynamics, the method has been successfully used for background modeling of surveillance videos previously (Grosek and Kutz, 2014; Kutz et al., 2015). In particular Erichson and Donovan (2016) and later Erichson et al. (2016a) have presented fast probabilistic DMD algorithms to reduce the computational costs. The former paper uses the randomized SVD to compute the DMD, while the latter paper advocates the compressed DMD algorithm, which shows to be extremely computationally efficient for background modeling. This chapter, evaluates the new rDMD algorithm, described in Section 3.4, for

background modeling. While, we do not expect a significant gain in terms of accuracy compared to the previous proposed DMD algorithms, it is still of interest to evaluate the performance of the rDMD algorithm for this task. We are confident that the randomized dynamic mode decomposition establishes the new standard among the probabilistic DMD algorithms. This is because the randomized algorithm is more robust, and reliable than the compressed algorithm, and computationally more efficient than using the randomized SVD for computing the DMD (Erichson et al., 2017).

First, Section 4.2 presents a video interpretation of the dynamic mode decomposition. Then, Section 4.3 outlines how the DMD can be used for real-time background modeling. In particular it is shown that the optimal mode selection for background modeling can be formulated as a sparsity-constrained sparse coding problem. Section 4.4 presents the measures which are used to evaluate the detection performance of the DMD algorithm. Finally, Section 4.5 shows some numerical results using standard background modeling benchmark datasets, which include real and synthetic surveillance videos. Concluding remarks and further research directions are outlined in Section 4.6.

4.2 Video Interpretation of the DMD

Surveillance videos are an appropriate application for the DMD, because a sequence of video frames forms a spatio-temporal grid. Further, video frames are equally spaced in time. For computational convenience the flattened grayscale video frames (snapshots) of a given video stream are stored, ordered in time, as column vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ of a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$. Here m denotes the number of pixels per frame, and n is the number of video frames taken. The matrix element x_{it} corresponds to a pixel intensity in space and time. The video frames can be thought of as snapshots of some underlying dynamics. Each video frame $\mathbf{x}_{t+1}$ at time $t+1$ is assumed to be connected to the previous frame $\mathbf{x}_t$ by the linear map $\mathbf{M} : \mathbb{R}^m \to \mathbb{R}^m$. Mathematically, the linear map $\mathbf{M}$ is a time-independent operator which constructs the approximate linear

evolution

$$\mathbf{x}_{t+1} \approx \mathbf{M}\mathbf{x}_t. \tag{4.1}$$

The objective of the dynamic mode decomposition is to find an estimate for the matrix $\mathbf{M}$ and its eigenvalue decomposition that characterizes the system dynamics. For more details we refer to Chapter 3.

Once the dynamic modes and eigenvalues for a given snapshot sequence are obtained, the method can attempt to reconstruct any given frame, or even possibly future frames. Specifically, a video frame $\mathbf{x}_t$ at time points $t \in 1, ..., m$ is approximately reconstructed as follows

$$\tilde{\mathbf{x}}_t = \sum_{j=1}^{k} b_j \phi_j \lambda_j^{t-1}. \tag{4.2}$$

Notice that the DMD mode ϕ_j is a $m \times 1$ vector containing the spatial structure of the decomposition, while the eigenvalue λ_j^{t-1} describes the temporal evolution. The scalar b_j is the amplitude of the corresponding DMD mode. Further, we make use of the intrinsic low-rank structure of videos and compute only the first k dominant dynamic modes and eigenvalues.

At time $t = 1$, Equation (4.2) reduces to $\tilde{\mathbf{x}}_1 = \sum_{j=1}^{k} b_j \phi_j$, because $\lambda_j^0 = 1$. Since the amplitude is time-independent, b_j can be obtained by solving the following least-square problem using the first video frame $\mathbf{x}_1$ as initial condition

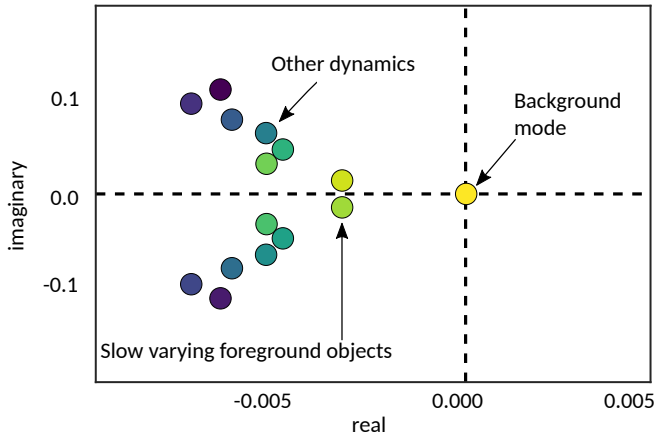
$$\hat{\mathbf{b}} = \underset{\mathbf{b}}{\operatorname{argmin}} \|\mathbf{x}_1 - \mathbf{\Phi}\mathbf{b}\|_F^2, \tag{4.3}$$

where the columns of $\mathbf{\Phi}$ contains the dominant DMD modes, and $\hat{\mathbf{b}}$ is the estimator for the amplitudes. It becomes apparent that any portion of the first video frame that does not change in time, or changes very slowly in time, must have an associated continuous-time eigenvalue

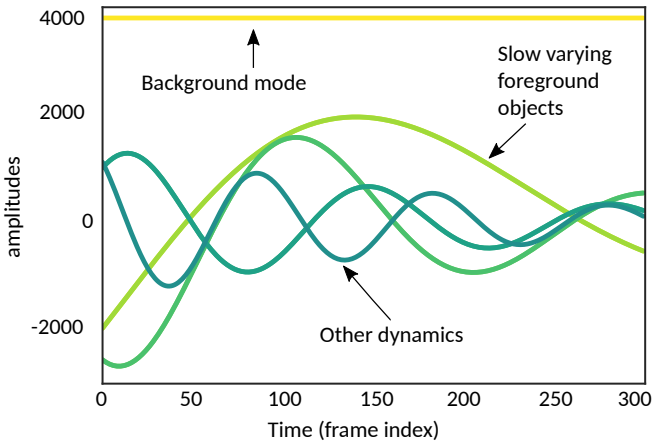
$$\omega_j = \frac{\log(\lambda_j)}{\Delta t} \tag{4.4}$$

that is located near the origin in complex space: $|\omega_j| \approx 0$ or equivalent $|\lambda_j| \approx 1$. This fact becomes the key principle to separate foreground elements from background information. Figure 4.3 shows the dominant continuous-time eigenvalues for a video sequence. Subplot (a) shows three

(a) Example frames ($t = 0, 150, 300$) of video sequence.



(b) Dominant continuous-time eigenvalues ω_j.



(c) Amplitudes over time.

Fig. 4.3 Subplot (a) shows three frames of the video sequence 'canoe', which is part of the change detection benchmark dataset. Subplot (b) and (c) show the continuous-time eigenvalues and the temporal evolution of the amplitudes. The modes corresponding to the amplitudes with the highest variance are capturing the dominant foreground object (canoe), while the zero mode captures the dominant structure of the background. Modes corresponding to high frequency amplitudes capture other dynamics in the video sequence like waves.

sample frames from this video sequence that includes a canoe. Here the foreground object (canoe) is not present at the beginning and the end of the video sequence. The dynamic mode decomposition factorizes this sequence into modes describing the different dynamics present. The analysis of the continuous-time eigenvalue ω_j and the amplitudes over time $\mathbf{B}\mathcal{V}$ (i.e. the amplitudes multiplied by the Vandermonde matrix) can provide interesting insights, shown in subplot (b) and (c). First, the amplitude for the prominent zero mode is constant over time, indicating that this mode is capturing the dominant (static) content of the video sequence, i.e., the background. The next pair of modes correspond to the canoe, a foreground object slowly moving over time. The amplitude reveals the presence of this object. Specifically, the amplitude reaches its maximum at about the frame index 150, when the canoe is in the center of the video frame. At the beginning and end of the video the canoe is not present, indicated by the negative values of the amplitude. The subsequent modes describe other dynamics in the video sequence, e.g., the movements of the canoeist and the waves. For instance, the modes describing the waves have high frequency and small amplitudes (not shown here). Hence, a theoretical viewpoint we will build upon with the DMD methodology centers around the recent idea of low-rank and sparse matrix decompositions. Following this approach, background modeling can be formulated as a matrix separation problem into low-rank (background) and sparse (foreground) components. This viewpoint has been advocated, for instance, by Candès et al. (2011) in the framework of robust principal component analysis (RPCA). For a thorough discussion of robust low-rank approximation techniques for background modeling, we refer to Bouwmans and Zahzah (2014), Bouwmans et al. (2016a) and Bouwmans et al. (2016b). The connection between DMD and RPCA was first established by Grosek and Kutz (2014). Specifically, the DMD provides a set of background modes corresponding to continuous-time eigenvalues $\{\omega_p : |\omega_p| \approx 0\}$ and a set of modes that corresponds to eigenvalues bounded away from 0 such as $\{\omega_{j\neq p} : |\omega_{j\neq p}| \gg 0\}$. Then,

Equation (4.2) can be re-expressed in the following form

$$\begin{aligned} \hat{\mathbf{X}} &= \mathbf{L} + \mathbf{S} \\ &= \underbrace{\sum_p b_p \phi_p \lambda_p^{\mathbf{t}-1}}_{\text{Background Video}} + \underbrace{\sum_{j \neq p} b_j \phi_j \lambda_j^{\mathbf{t}-1}}_{\text{Foreground Video}}, \end{aligned} \tag{4.5}$$

where $\mathbf{t} = [1, ..., m]$ is a $1 \times m$ time vector and $\hat{\mathbf{X}} \in \mathbb{C}^{m \times n}$.[1] Hence, the DMD provides an approximate matrix decomposition of the form $\hat{\mathbf{X}} = \mathbf{L} + \mathbf{S}$, where the low-rank matrix $\mathbf{L}$ will render the video of just the background, and the sparse matrix $\mathbf{S}$ will render the complementary video of the moving foreground objects. We can interpret these DMD results as follows: stationary background objects translate into highly correlated pixel regions from one frame to the next, which suggests a low-rank structure within the video data. Thus the DMD algorithm can be thought of as an RPCA method. The advantage of the DMD method and its sparse/low-rank separation is the computationally efficiency of achieving Equation (4.5), especially when compared to the optimization methods of RPCA. The analysis of the time evolving amplitudes provide interesting opportunities. Specifically, learning the amplitudes' profiles for different foreground objects allows automatic separation of video feeds into different components. For instance, it could be of interest to discriminate between cars and pedestrians in a given video sequence.

4.3 Real-Time Background Modeling

When dealing with high-resolution videos, the standard DMD approach is expensive in terms of computational time and memory, because the whole video sequence is reconstructed. Instead a 'good' static background model is often sufficient for background subtraction. This is because background dynamics can be filtered out or thresholded. The challenge remains to automatically select the modes best describing the background. This is essentially a bias-variance trade-off. Using just the zero mode (background) leads to an under-fitted background model, while a large set

[1] Note that by construction $\hat{\mathbf{X}}$ is complex, while pixel intensities of the original video stream are real-valued. Hence, only the the real part is considered in the following.

of modes tends to overfit. Motivated by the sparsity-promoting variant of the DMD algorithm introduced by Jovanović et al. (2014), we formulate a sparsity-constrained sparse coding problem for mode selection. The idea is to augment Equation (4.3) by an additional term that penalizes the number of non-zero elements in the vector $\mathbf{b}$

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \|\mathbf{x}_1 - \boldsymbol{\Phi}\boldsymbol{\beta}\|_F^2 \text{ such that } \|\boldsymbol{\beta}\|_0 < K, \tag{4.6}$$

where $\boldsymbol{\beta}$ is the sparse representation of $\mathbf{b}$, and $\|\cdot\|_0$ is the ℓ_0 pseudo norm which counts the non-zero elements in $\boldsymbol{\beta}$. Solving this sparsity problem exactly is NP-hard. However, the problem in Equation (4.6) can be efficiently solved using greedy approximation methods. Specifically, we utilize orthogonal matching pursuit (OMP) as formulated by Mallat and Zhang (1993) and Tropp and Gilbert (2007). A highly computationally efficient algorithm was later proposed by Rubinstein et al. (2008) and is implemented in the scikit-learn software package (Pedregosa et al., 2011). The greedy OMP algorithm works iteratively, selecting at each step the mode with the highest correlation to the current residual. Once a mode is selected the initial condition $\mathbf{x}_1$ is orthogonally projected on the span of the previously selected set of modes. Then the residual is recomputed and the process is repeated until K non-zero entries are obtained. If no priors are available, the optimal number of modes K can be determined using cross-validation. Finally, the background model is computed as

$$\hat{\mathbf{x}}_{BG} = \boldsymbol{\Phi}\hat{\boldsymbol{\beta}}. \tag{4.7}$$

The disadvantage of this approach is, however, that the quality depends on the frame which is used as initial condition. If $\mathbf{x}_1$ is heavily crowded, then the method tends to overfit. Thus, to avoid degenerated results, the previous modeled background $\hat{\mathbf{x}}_{BG}$ frame can be used as initial condition instead. Alternatively, the median frame can be used as well.

Finally, a binary foreground mask $\mathcal{X}$ can be obtained by thresholding the distance between the pixels of a given video frame and the corresponding background model. Thus, we yield the following binary classification

problem

$$\mathcal{X}_t(j) = \begin{cases} 1 & \text{if } \mathbf{d}(x_{jt}, \hat{x}_j) > \tau, \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

where x_{jt} denotes the j-th pixel of the t-th video frame and $\hat{x}_j$ denotes the corresponding pixel of the modeled background. Pixels which exceed the threshold parameter τ are classified as foreground objects, i.e., they are set to 1 and 0 otherwise. Due to its simplicity, the Euclidean metric is often used to compute the distance $\mathbf{d}()$ between pixels in practice (Benezeth et al., 2010).

4.4 Evaluation Measures

Assuming that the true foreground mask is known, a confusion matrix can be formed to assess the performance of the background model as follows

		Truth		
		0	1	
Prediction	0	TN	FN	#pred neg
	1	FP	TP	#pred pos
		#true neg	#true pos	

(4.9)

TP denotes the (number of) true positive predictions, i.e. pixels which are correctly classified as belonging to a moving foreground object. Similarly TN denotes the (number of) true negative predictions, i.e., pixels which are correctly classified as background. False Positive (FP) and false negative (FN) are the respective misclassifications for foreground and background elements. Based on the confusion matrix we can compute the following common evaluation measures: recall, precision and the F-measure. Among others, these three are the most common measures to evaluate the background performance in the literature (Bouwmans et al., 2016b; Sobral and Vacavant, 2014; Xu et al., 2016).

Recall (also called sensitivity, true positive rate or hit rate) measures the algorithm's ability to correctly detect pixels belonging to foreground objects. It is computed as the ratio of predicted true positives to the

total number of true positive foreground pixels

$$\textbf{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{4.10}$$

Precision (also called false alarm rate or true positive accuracy) measures how confident we can be that a positive classified pixel actually belongs to a foreground object. It is computed as the ratio of predicted true positives to the total number of pixels predicted as foreground objects

$$\textbf{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}. \tag{4.11}$$

The F-measure combines recall and precision as their harmonic mean, weighting both measures evenly, defined as

$$\mathbf{F} = 2 \times \frac{\mathrm{Recall} \times \mathrm{Precision}}{\mathrm{Recall} + \mathrm{Precision}}. \tag{4.12}$$

4.5 Numerical Results

In this section we evaluate the computational performance and the suitability of the randomized dynamic mode decomposition for background modeling, using both syntactic and real benchmark videos. Specifically, the performance is evaluated using the change detection (CD) and the background models challenge (BMC) benchmark datasets (Vacavant et al., 2013; Wang et al., 2014b). In particular, the following complex situations are encountered:

- **Illumination**: Gradual change of pixel intensities caused by illumination conditions, e.g., fog and sun, as well as bad light conditions, e.g., night videos.
- **Bad weather**: Small moving objects related to weather conditions, e.g., snow and rain.
- **Dynamic backgrounds**: Moving objects belonging to the background, e.g., waving trees, fountains, water surfaces and clouds.
- **Sleeping foreground objects**: Objects that are becoming motionless and moving again at a later point in time.

4.5.1 Evaluation Settings

The following settings have been used in order to obtain reproducible results. For a given video sequence, the randomized dynamic mode decomposition is computed using a fixed target rank $k = 25$, an oversampling parameter $p = 10$, and $q = 1$ power iteration. Compressed DMD is computed using $c = 1000$ (random rows) single pixel measurements. Once the DMD is obtained, the optimal set of modes is selected using the orthogonal matching pursuit method. In general the use of $K = 10$ non-zero entries achieves good results. Notice that the DMD is formulated here as a batch algorithm. Thus, a given long video sequence is split into batches of 200 consecutive frames. The decomposition is then computed for each batch independently. For every video an individual threshold value has been selected by hand in order to compute the foreground mask.

4.5.2 Evaluation Using the CD Dataset

First, six CD video sequences (Wang et al., 2014b) are used to contextualize the background model quality using the sparse-coding approach (denoted as 'opt'). This is compared to using the dominant (static) background mode only (denoted as '0'). Figure 4.4 shows the results by plotting the F-measure versus varying thresholds. Note, that here only one batch (i.e., 200 frames) is used for the evaluation. In four out of six examples the sparse-coding approach dominates. In particular, significant improvements are achieved for the dynamic background video sequences 'Canoe' and 'Fountain02'. Only in case of the 'Park' video sequence the method tends to over-fit. This is, because here the initial frame contains several moving objects. Interestingly, the performance of the randomized algorithm is slightly better than the deterministic DMD algorithm for some of the video sequences, overall. This is due to the implicit regularization of randomized algorithms (Mahoney, 2011). However, it is important to note that the performance can also slightly vary over several runs due to random fluctuations.

Next, we show in Table 4.1 the evaluation results of 8 real videos from the CD dataset. The videos are from three different categories: 'Baseline', 'Dynamic Background' and 'Thermal'. For comparison, the results of

two leading algorithms in the CD ranking are shown as well. Firstly, the flux tensor with split Gaussian models (FTSG) algorithm (Wang et al., 2014a), and secondly the pixel-based adaptive word consensus segmenter (PAWCS) algorithm (St-Charles et al., 2015). The former algorithm is based on a mixture of Gaussians and won the 2014 CD challenge. The latter algorithm is based on a word-based approach for background modeling.

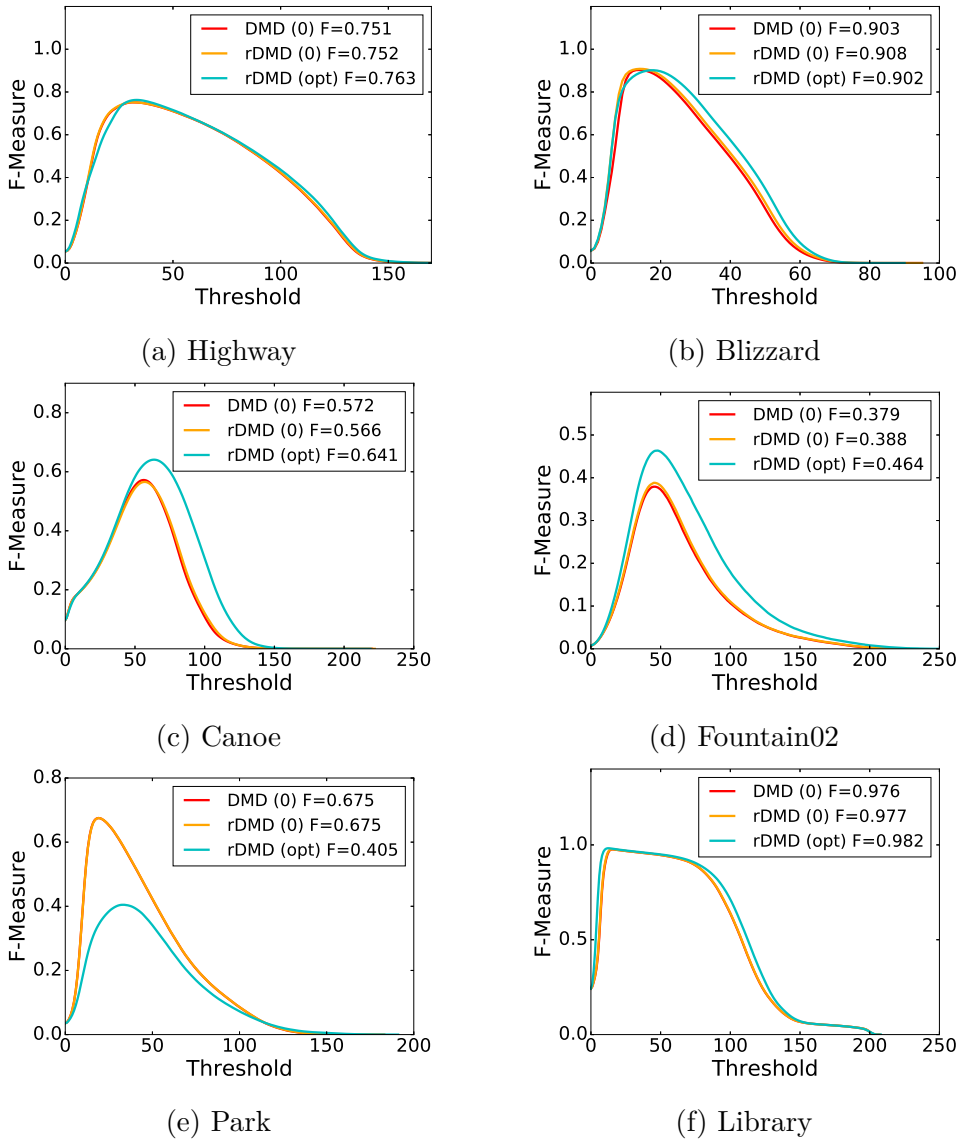


Fig. 4.4 The F-measure shows the improved performance of the sparsity-promoting approach over using only the zero mode.

As expected, the deterministic and probabilistic DMD algorithms show very similar results. Interestingly, the randomized DMD algorithm performs slightly better than the compressed and the deterministic algorithm on most of the videos. However, the overall detection accuracy of the DMD algorithms looks poor compared to the FTSG and PAWCS algorithm. This can be related to several challenges faced here. While the performance on the two baseline videos 'Highway' and 'Pedestrians' are good, issues arise for the other two baseline videos. The video 'PETS2006' is challenging for the DMD algorithm, due to camouflage effects as well as because some of the objects are sleeping foreground objects. The video 'Office' shows even more drastically that DMD cannot cope with sleeping foreground objects.

Further, if videos feature dynamic backgrounds, it can be seen that the DMD achieves good recall rates, while the precision is poor. The performance for the thermal videos is relatively good.

Overall, the raw results of the DMD cannot compete with the FTSG and PAWCS methods. These, two methods are highly optimized for the task of background modeling. Moreover, they consider the information from all three RGB channels, while DMD is using only the information of the grayscale video. But, it can be seen that the performance can be substantially improved by simple post-processing techniques like the median filter. In addition to the quantitative results, Figure 4.5 shows some visual results for three selected videos.

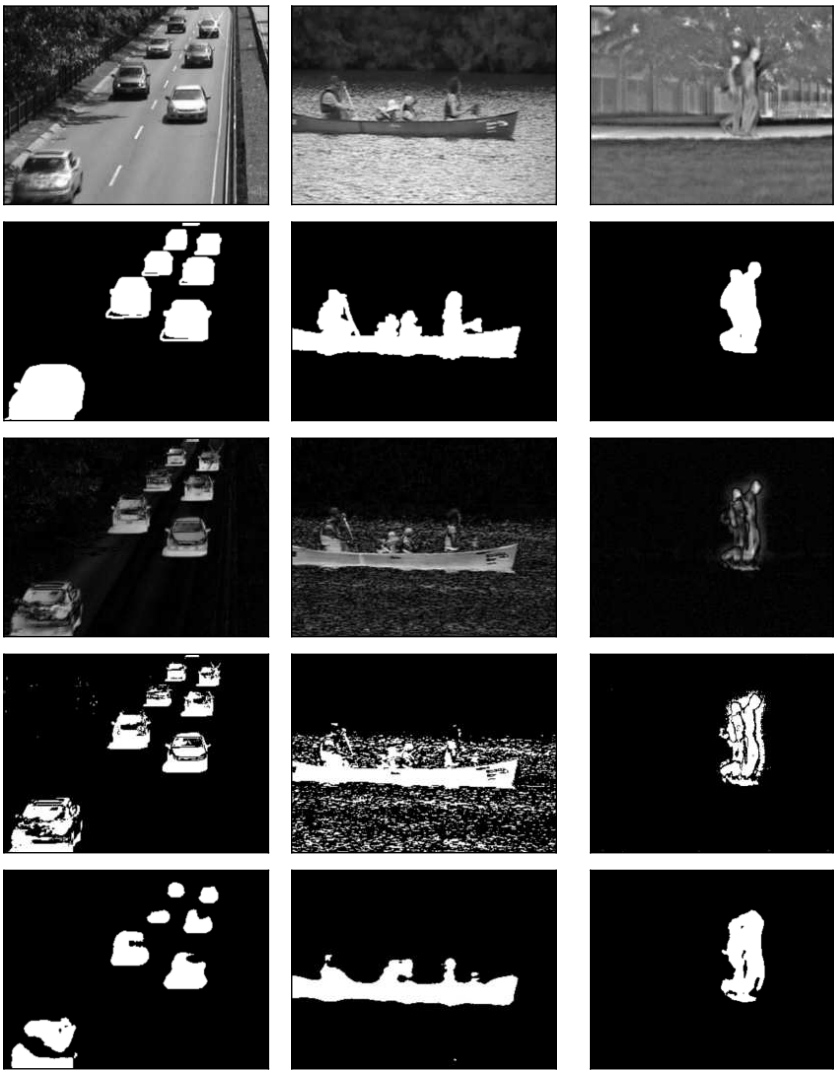


Fig. 4.5 Visual results showing frames of the 'Highway', 'Canoe' and 'Park' video. The top row shows the grayscale frames, and the second row the corresponding true foreground masks. The third row shows the differencing between the frames and the background model. The fourth and fives row show the thresholded, and median filtered foreground masks.

Table 4.1 Evaluation results of eight real videos from the CD dataset. For comparison, the results of two algorithms from the CD ranking are presented.

	Measure	Baseline				Dynamic Background		Thermal		Average
		Highway	Pedestrians	PETS2006	Office	Overpass	Canoe	Park	Lakeside	
PAWCS St-Charles et al. (2015)	Recall	0.952	0.961	0.945	0.905	0.961	0.947	0.899	0.520	-
	Precision	0.935	0.931	0.919	0.972	0.957	0.929	0.768	0.752	-
	F-Measure	0.944	0.946	0.932	0.937	0.959	0.938	0.829	0.615	0.89
FTSG Wang et al. (2014a)	Recall	0.956	0.979	0.963	0.908	0.944	0.913	0.666	0.228	-
	Precision	0.934	0.890	0.883	0.961	0.941	0.985	0.724	0.960	-
	F-Measure	0.945	0.932	0.921	0.934	0.943	0.948	0.694	0.369	0.84
DMD	Recall	0.803	0.943	0.692	0.347	0.540	0.621	0.736	0.447	-
	Precision	0.759	0.781	0.736	0.558	0.448	0.663	0.614	0.402	-
	F-Measure	0.780	0.854	0.713	0.428	0.490	0.641	0.669	0.423	0.62
cDMD	Recall	0.803	0.942	0.692	0.387	0.552	0.629	0.736	0.435	-
	Precision	0.753	0.743	0.716	0.246	0.467	0.664	0.613	0.567	-
	F-Measure	0.777	0.831	0.704	0.301	0.506	0.646	0.669	0.492	0.61
rDMD	Recall	0.803	0.943	0.693	0.349	0.552	0.618	0.736	0.439	-
	Precision	0.760	0.783	0.735	0.567	0.471	0.660	0.614	0.531	-
	F-Measure	0.781	0.855	0.713	0.432	0.508	0.638	0.669	0.480	0.63
rDMD (with median filter)	Recall	0.899	0.975	0.696	0.346	0.764	0.615	0.798	0.655	-
	Precision	0.845	0.948	0.741	0.578	0.813	0.985	0.764	0.571	-
	F-Measure	0.871	0.961	0.718	0.433	0.788	0.758	0.781	0.610	0.74

4.5.3 Evaluation Using the BMC Dataset

Next, we use the BMC dataset which comprises nine real and ten synthetic video sequences. Figure 4.6 shows sample frames of the nine real videos. These video sequences reproduce challenges which are commonly encountered in outdoor surveillance videos. Further, the BMC dataset allows us to compare the randomized DMD algorithm to leading RPCA algorithms by using the results of Bouwmans et al. (2016b). Here, we compare the performance of the DMD to the online robust principal component analysis (OR-PCA) algorithm (Javed et al., 2014) as well as to the GoDec and the IALM algorithms. This OR-PCA algorithm has been designed, in particular, as a robust foreground detection algorithm, and the continuous constraint approach using the markov random field (MRF) helps to improve the accuracy considerably. Currently, this algorithm is the best performing algorithm according to the ranking provided by Bouwmans et al. (2016b). Javed et al. (2015b) have improved the algorithm even further by introducing the concept of dynamic feature selection. The GoDec algorithm is an approximated RPCA algorithm, which is based on bilateral random projections (Zhou and Tao, 2011). Compared to most of the other RPCA algorithms, this algorithm shows a relatively good computational performance. The IALM algorithm is one of the standard RPCA algorithms for obtaining the low-rank and sparse components of a data matrix (Lin et al., 2011). In particular, the IALM algorithm can benefit from using the randomized SVD as demonstrated in Chapter 2. In addition we show the results of other prominent background models like the adaptive mixture-of-Gaussians background model (Shimada et al., 2006), the robust orthonormal subspace learning background model (Shu et al., 2014) as well as the DECOLOR (DEtecting Contiguous Outliers in the LOw-rank Representation) model (Zhou et al., 2013).

Table 4.3 shows the evaluation results for the real video sequences.[2] Overall rDMD achieves an average F-measure of about 0.65. This is slightly better than the performance of GoDec and IALM, but substantially poorer than the F-measure of the OR-PCA (with MRF) algorithm. In particular, the videos '001', '005' , '008' , and '009' are challenging

[2]The evaluation results were obtained by using the BMC wizard: http://bmc.iut-auvergne.com.

for the DMD algorithm. These videos contain all a lot of background noise such as moving trees, clouds, rain and snow. On the other hand the results show that DMD can deal with large moving objects and low illumination conditions.

Figure 4.7 presents some visual results for fife videos. Here, the last row shows the smoothed (median filtered) foreground mask.

(001) Boring parking

(002) Big trucks

(003) Wandering students

(004) Rabbit in the night

(005) Snowy Christmas

(006) Beware of the trains

(007) Train in the tunnel

(008) Traffic during windy day'

(009) One rainy hour

Fig. 4.6 BMC dataset: Example frames of the 9 real videos.

Table 4.4 shows the results of randomized DMD for the ten synthetic videos of the BMC dataset and the picture is very similar to the previous evaluation results. The synthetic videos show that DMD is flexible enough to deal with some illumination changes, clouds, fog and sun. However, in particular, the videos 'Street 412' and 'Rotary 422' are challenging, due to the emulated windy scenes, and the additional noise. Overall, the OR-PCA algorithm outperforms, while the dynamic mode decomposition performs slightly better than the GoDec and IALM algorithm here.

4.5.4 Computational Performance

Table 4.2 shows the average computational time required to obtain the background model using different DMD algorithms. In addition the number of processed frames per second (FPS) is shown. Specifically, a

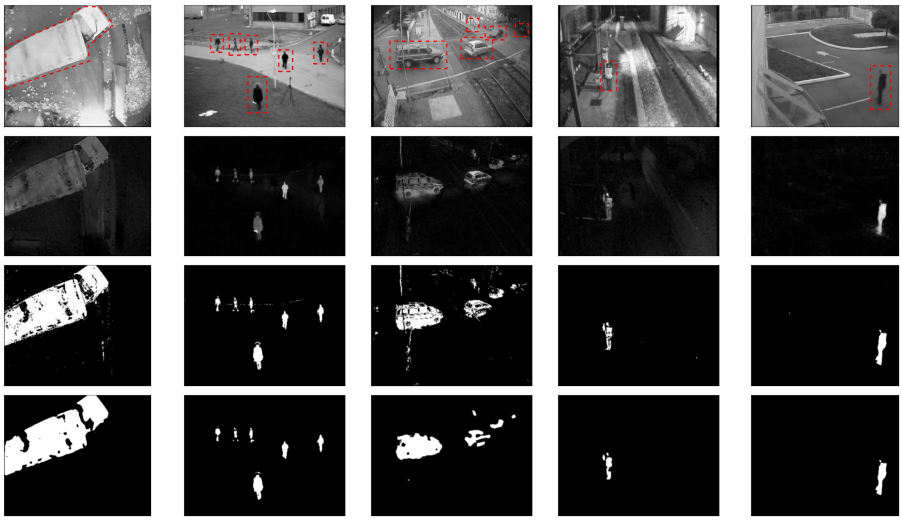


Fig. 4.7 Visual results for fife frames corresponding to the BMC Videos: '002', '003', '006', '007' and '009'. The top row shows the original grayscale images. The second row shows the differencing between the background model and the video frame. The third and fourth row show the thresholded and the median filtered foreground mask.

sequence of 200 frames is decomposed using the target rank $k = 25$. The randomized DMD algorithm as well as the DMD algorithm using the randomized SVD is computed with the parameter $p = 10$ and $q = 1$. The compressed DMD algorithm is computed using $c = 1000$ random rows.

The deterministic algorithm shows a good performance for low resolution videos, but due to some memory error the algorithm fails to find a decomposition for the high resolution videos. Similar does the compressed DMD algorithm fail to decompose the HD (720) video sequence.

Overall, the probabilistic algorithms show substantial computational savings compared to the deterministic algorithm. Erichson and Donovan (2016) proposed to use the randomized SVD for computing the DMD, and indeed this approach shows a considerable speedup. However, the compressed and the randomized DMD algorithm show an even better performance. The compressed DMD shows a slightly better performance than the randomized algorithms, but the latter algorithm is more memory efficient. Hence, overall the randomized algorithms is favorable. The advantage of the rDMD algorithm becomes more pronounced for data of increasing dimensions. In addition, Figure 5.8 in Chapter 5 shows the

Table 4.2 Algorithms runtime for obtaining the background model for varying video resolutions. Here a sequence of 200 frames is decomposed using target rank $k = 25$, and the parameters $p = 10$, $q = 1$ and $c = 1000$ for the probabilistic algorithms.

		Resolution					
		QVGA 320×240	HVGA 480×320	VGA 640×480	SVGA 800×600	XGA 1024×768	HD 720 1280×720
Deterministic DMD	Time (s)	2.27	4.41	9.49	16.18	-	-
	FPS	88	45	21	12	-	-
DMD using rSVD	Time (s)	1.30	2.63	5.65	9.00	14.98	19.66
	FPS	153	75	35	22	13	10
Compressed DMD	Time (s)	0.65	1.31	2.67	4.11	7.07	-
	FPS	307	153	75	48	28	-
Randomized DMD	Time (s)	0.77	1.38	2.80	4.41	7.75	8.55
	FPS	261	144	71	45	25	23

computational performance of the GPU accelerated randomized DMD algorithm.

4.6 Conclusion

Background modeling is a complex and challenging task. In particular, real-time HD video analysis remains one of the grand challenges of the field. This is, because most algorithms in this area are very computationally demanding.

The dynamic mode decomposition has shown to be a viable and computationally efficient candidate for background modeling. Despite the significant computational savings, the randomized DMD algorithm remains competitive with other leading methods in the quality of the decomposition itself. In particular, the trade-off between speed and accuracy of the randomized DMD is compelling. Specifically, our results show, that for both standard and challenging environments, the rDMD's background subtraction accuracy in terms of the F-measure is competitive to some robust principal component analysis algorithms. However, the DMD algorithm cannot compete, in terms of the F-measure, with highly specialized algorithms for background modeling like FTSG, PAWC or OR-PCA with MRF. Thus, while the DMD provides an interesting theoretical framework for background modeling, it requires the integration into a more general vision system in order to be useful in practice.

As a final remark in this Chapter, we need to note that the emergence of deep learning greatly challenges the future of low-rank matrix approximations for the task of background modeling. The work by Wang et al. (2016a), Braham and Van Droogenbroeck (2016), Babaee et al. (2017) and Kang et al. (2016) show compelling results.

Table 4.3 Evaluation results of nine real videos from the BMC dataset. For comparison, the results of three leading robust PCA algorithms are presented, adapted from Bouwmans et al. (2016b).

	Measure	BMC real videos									Average
		001	002	003	004	005	006	007	008	009	
OR-PCA with MRF Javed et al. (2014)	Recall	0.776	0.845	0.905	0.799	0.779	0.800	0.806	0.566	0.95	-
	Precision	0.936	0.781	0.738	0.870	0.860	0.891	0.768	0.558	0.74	-
	F-Measure	0.848	0.812	0.813	0.834	0.826	0.843	0.786	0.562	0.854	0.8
IALM Lin et al. (2011)	Recall	0.697	0.515	0.759	0.691	0.635	0.642	0.433	0.617	0.70	-
	Precision	0.585	0.723	0.798	0.678	0.483	0.643	0.683	0.632	0.80	-
	F-Measure	0.637	0.605	0.778	0.684	0.551	0.643	0.536	0.624	0.754	0.64
SemiSoft GoDec Zhou and Tao (2011)	Recall	0.666	0.491	0.769	0.681	0.636	0.644	0.438	0.594	0.68	-
	Precision	0.548	0.706	0.809	0.694	0.489	0.632	0.642	0.629	0.81	-
	F-Measure	0.602	0.583	0.789	0.687	0.555	0.638	0.525	0.611	0.744	0.64
Adaptive MOG Shimada et al. (2006)	Recall	0.849	0.580	0.859	0.829	0.754	0.780	0.691	0.723	0.828	-
	Precision	0.682	0.546	0.780	0.580	0.435	0.636	0.603	0.495	0.790	-
	F-Measure	0.757	0.562	0.818	0.785	0.558	0.702	0.644	0.591	0.809	0.68
ROSL Shu et al. (2014)	Recall	0.743	0.837	0.912	0.851	0.823	0.843	0.778	0.562	0.768	-
	Precision	0.865	0.731	0.779	0.531	0.512	0.680	0.684	0.508	0.852	-
	F-Measure	.799	0.781	0.840	0.654	0.631	0.753	0.728	0.534	0.808	0.725
DMD	Recall	0.552	0.779	0.772	0.694	0.615	0.701	0.723	0.512	0.566	-
	Precision	0.579	0.643	0.756	0.771	0.542	0.595	0.823	0.509	0.575	-
	F-Measure	0.565	0.705	0.764	0.731	0.576	0.644	0.769	0.510	0.570	0.65
cDMD	Recall	0.552	0.746	0.770	0.693	0.611	0.698	0.720	0.515	0.566	-
	Precision	0.581	0.605	0.760	0.770	0.541	0.594	0.823	0.510	0.574	-
	F-Measure	0.566	0.668	0.765	0.730	0.574	0.642	0.768	0.512	0.570	0.64
rDMD	Recall	0.552	0.777	0.772	0.694	0.615	0.699	0.723	0.516	0.566	-
	Precision	0.581	0.639	0.754	0.771	0.543	0.594	0.823	0.511	0.575	-
	F-Measure	0.566	0.701	0.763	0.731	0.577	0.642	0.769	0.513	0.570	0.65

Table 4.4 Evaluation results of ten synthetic videos from the BMC dataset. For comparison, the results of three leading RPCA algorithms are presented, adapted from Bouwmans et al. (2016b).

	Measure	Street					Rotary					Average
		112	212	312	412	512	122	222	322	422	522	
OR-PCA with MRF Javed et al. (2014)	Recall	0.871	0.870	0.894	0.850	0.860	0.937	0.940	0.923	0.917	0.841	-
	Precision	0.956	0.952	0.882	0.873	0.894	0.924	0.924	0.901	0.846	0.92	-
	F-Measure	0.911	0.909	0.888	0.861	0.876	0.931	0.932	0.912	0.880	0.879	0.897
IALM Lin et al. (2011)	Recall	0.774	0.689	0.741	0.738	0.677	0.743	0.750	0.741	0.705	0.70	-
	Precision	0.662	0.811	0.719	0.743	0.664	0.779	0.769	0.740	0.773	0.74	-
	F-Measure	0.715	0.746	0.730	0.741	0.670	0.761	0.759	0.740	0.737	0.725	0.732
SemiSoft GoDec Zhou and Tao (2011)	Recall	0.692	0.700	0.717	0.730	0.664	0.726	0.718	0.673	0.642	0.71	-
	Precision	0.816	0.818	0.752	0.772	0.601	0.792	0.799	0.750	0.804	0.68	-
	F-Measure	0.750	0.755	0.734	0.750	0.631	0.758	0.757	0.710	0.715	0.702	0.726
Adaptive MOG Shimada et al. (2006)	Recall	0.827	0.827	0.797	0.761	0.821	0.823	0.831	0.797	0.743	-	
	Precision	0.766	0.768	0.480	0.426	0.519	0.786	0.790	0.526	0.435	0.740	-
	F-Measure	0.796	0.796	0.605	0.553	0.640	0.804	0.810	0.638	0.555	0.784	0.698
DECOLOR Zhou et al. (2013)	Recall	0.982	0.985	0.983	0.980	0.978	0.983	0.983	0.981	0.967	0.980	-
	Precision	0.778	0.748	0.747	0.729	0.599	0.764	0.759	0.760	0.762	0.694	-
	F-Measure	0.868	0.851	0.849	0.836	0.743	0.860	0.857	0.857	0.852	0.813	0.838
DMD	Recall	0.857	0.856	0.789	0.710	0.739	0.861	0.836	0.708	0.682	0.748	-
	Precision	0.910	0.879	0.558	0.633	0.676	0.905	0.921	0.896	0.576	0.829	-
	F-Measure	0.883	0.867	0.653	0.670	0.706	0.882	0.876	0.791	0.624	0.787	0.77
cDMD	Recall	0.858	0.856	0.790	0.710	0.740	0.861	0.835	0.707	0.681	0.748	-
	Precision	0.908	0.898	0.559	0.632	0.707	0.905	0.924	0.895	0.575	0.836	-
	F-Measure	0.882	0.877	0.655	0.669	0.723	0.882	0.877	0.790	0.623	0.789	0.77
rDMD	Recall	0.857	0.856	0.788	0.710	0.739	0.861	0.836	0.710	0.682	0.748	-
	Precision	0.910	0.878	0.558	0.633	0.695	0.905	0.922	0.868	0.575	0.827	-
	F-Measure	0.883	0.867	0.653	0.670	0.717	0.882	0.877	0.781	0.624	0.786	0.77

Chapter 5

GPU Accelerated Randomized Algorithms

"Machines take me by surprise with great frequency."

— Alan Turing

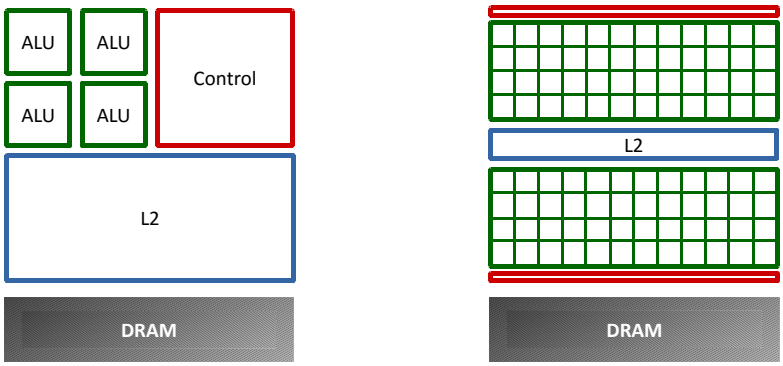
Note: The work described in this chapter was carried out in collaboration with Professors J. Nathan Kutz and Steven L. Brunton of University of Washington. Parts of the work appeared in the Journal of Real-Time Image Processing under the title 'Compressed dynamic mode decomposition for background modeling'. My contributions involve conceptualizing the project idea, implementing the routines, running the simulations as well as writing the original draft.

5.1 Introduction

Graphics processing units (GPUs) becoming increasingly popular for general-purpose high-performance computing. Compared to central processing units (CPUs), the architecture of GPUs enable massive parallel processing. This paradigm of parallel computing has been proven valuable for a variety of computational tasks, e.g., linear algebra or Monte-Carlo simulations. Specifically CUDA, NVIDIA's programming model for parallel computing opens up GPUs as a general parallel computing device (Nickolls et al., 2008).

Originally GPUs were designed for the real-time creation of high-definition 2D/3D graphics. Thus, the computational architecture was

optimized for data-parallel, throughput computations of array like data structures. This is achieved by a large number of small arithmetic logic units (ALUs). In contrast, CPUs consist of a few ALUs, which are highly optimized for low-latency access to cached data sets.[1] Figure 5.1 illustrates the architecture of both modern CPUs and GPUs.



(a) CPU (b) GPU

Fig. 5.1 Illustration of the CPU and GPU architecture. Compared to the CPU, the GPU consist of many arithmetic logic units (green) which enable massive parallel processing.

Motivation and Overview

The probabilistic framework for computing low-rank matrix decompositions as presented in Chapter 1 is embarrassingly parallel. Take, for instance, the matrix multiplication of two $n \times n$ square matrices, illustrated in Figure 5.2. The computation involves the evaluation of n^2 dot products.[2] The data parallelism therein is that each dot-product can be computed independently. With enough ALUs the computational time can be substantially accelerated. This parallelism applies readily to the generation of random numbers and many other operations in linear algebra.

[1]Intel has recently introduced the new Xeon Phi coprocessor, which provides many ALUs. This allows highly parallel computing using a CPU architecture, and benchmarks show that the performance is highly competitive with GPUs.

[2]Modern efficient matrix-matrix multiplications are based on block matrix decomposition or other computational tricks, and do not actually compute n^2 dot products. However the concept of parallelism remains the same.

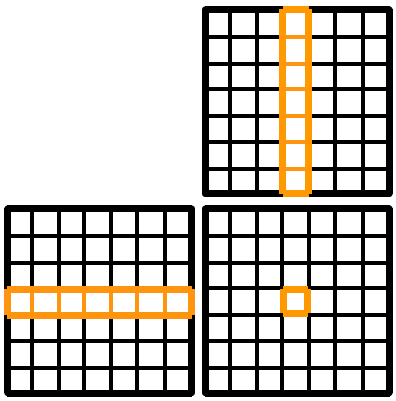


Fig. 5.2 Illustration of the data parallelism in matrix-matrix multiplications. The entries of the resulting matrix can be computed as independent dot products in parallel.

In Section 5.2 we give a brief background about GPU computing. Then, in Section 5.3 we outline the **Scikit-CUDA** package which provides GPU-based libraries for linear algebra routines in Python. Section 5.4 shows the performance of the GPU accelerated implements for both the randomized singular value decomposition and the dynamic mode decomposition. Section 5.5 discusses the results.

5.2 Background: GPU Computing

Linear algebra is at the core of most scientific computing applications, and the mathematics behind machine learning and vision. In particular, solving large linear systems of equations is a standard problem in these areas. In the area of image processing it is often required to perform the same computations on a large number of pixels. The GPU is a powerful engine to tackle these computationally demanding tasks. This is, because often the required computations can be parallelized effectively on GPU. Thus, the paradigm of parallel computing is the future of computing in the era of 'big data' (Owens et al., 2008). We refer the reader for a comprehensive introduction into the field of parallel and GPU computing to Wen-Mei (2011), Cai and See (2015) and Sanders and Kandrot (2010).

In particular, fundamental matrix decompositions like the Cholesky factorization, the LU, and the QR decomposition benefit substantially

from GPU accelerated implementations. Tomov et al. (2010) show that the performance of the GPU accelerated Cholesky factorization is up to 100 times higher than the performance of the CPU routine. Similar, Volkov and Demmel (2008) show impressive performance results of GPU acclerated implementations of these dense matrix factorizations. Substantial performance gains are also achieved for randomized matrix algorithms, for instance, see Ji and Li (2014), Voronin and Martinsson (2015) and Martinsson et al. (2017).

Analogous, performance gains are demonstrated in the area of image processing (Wang et al., 2013). For instance, Park et al. (2008) propose a GPU accelerated Canny Edge detection algorithm. Further, Zhang et al. (2014) demonstrate the computational advantage of a highly parallelized mixture-of-Gaussian algorithm for background subtraction.

Finally, it is to mention that GPU computing plays a fundamental role in deep learning to make training faster (Chetlur et al., 2014; Krizhevsky et al., 2012).

5.3 The scikit-CUDA Package

The multi-author **scikit-CUDA** package (initiated by Lev Givon) provides **NumPy** like interfaces to several low-level CUDA routines, e.g., **CUBLAS**, **CUFFT**, and **CUSOLVER** (Givon et al., 2015). In addition, several wrapper functions for the high-performance linear algebra library **CULA** (Humphrey et al., 2010) are provided. These low-level wrapper functions allow to implement GPU accelerated matrix decompositions in **Python**. We have implemented routines to compute the randomized singular value decomposition, and the randomized dynamic mode decomposition. The interfaces are similar to the CPU implementations of Algorithm 1 and 3. The basic GPU **rsvd()** interface is

```
U, S, V = rsvd(a_gpu, k=None, p=10, q=1, ...)
```

where the first argument **a_gpu** passes a $m \times n$ **GPUArray**. Given a **NumPy** like 2-dimensional array **A**, the GPU array is simply created as follows

```
a_gpu = gpuarray.to_gpu(A)
```

Further, the basic GPU **rdmd()** interface is

```
F, b, V = rdmd(a_gpu, k=None, p=10, q=1,
   return_amplitudes=False, return_vandermonde=False, ...)
```

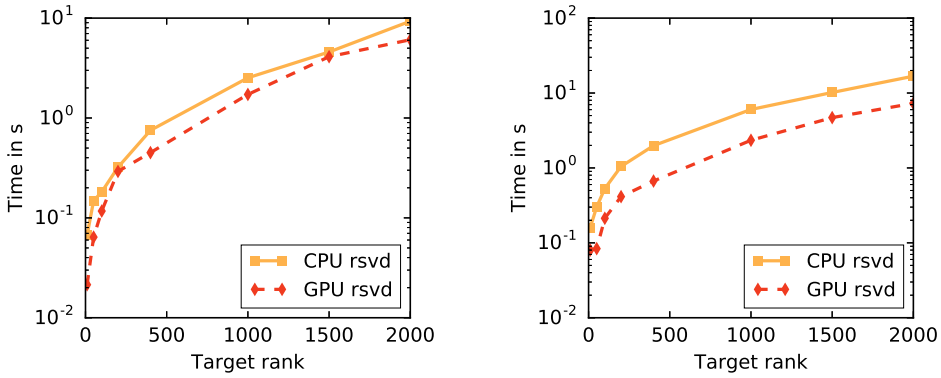
It is important to note that the **GPUArray** assumes Fortran memory ordering. For installation instructions visit the GIT repository https://github.com/lebedov/scikit-cuda. See also the package manual, as all package functions are fully documented.

5.4 Numerical Results

In the following, we are using random matrices to compare the computational performance of the CPU and GPU accelerated implementations in **Python**. All computations are performed on a standard notebook with Intel Core i7-5500U 2.4GHz, 8GB DDR3 memory and a NVIDIA GeForece GTX 980M with 4GB VRAM.

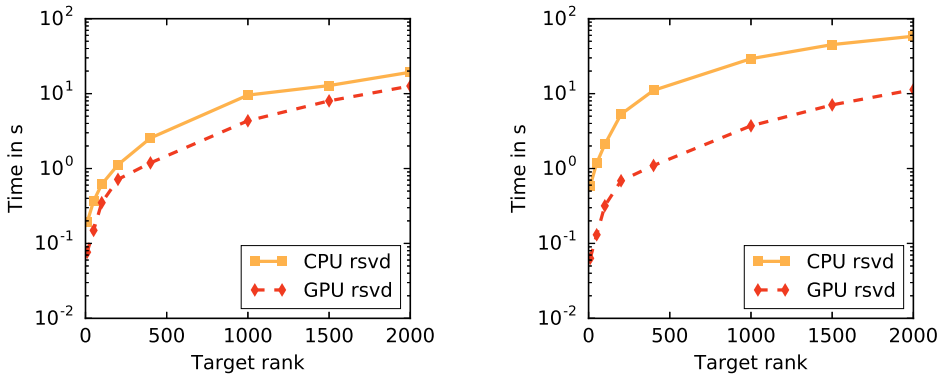
5.4.1 Randomized Singular Value Decomposition

First we compare the performance for a square random matrix of dimension 5000×5000 with and without power iterations. Figure 5.3 shows the results. Independent of the target rank, the performance gain is only minor when the computation of power iterations are omitted. However, some computational savings are achieved in the latter case, where $q = 2$ power iterations are computed. The computational advantage becomes more distinct with increasing dimensions of the input matrix. Figure 5.4 shows the performance for a square random matrix of dimension 10000×10000 with and without power iterations. Here speedups of about $2 - 5$ are achieved.



(a) $p = 10, q = 0$. (b) $p = 10, q = 2$.

Fig. 5.3 Average runtime of the CPU and GPU accelerated randomized singular value decomposition for varying target ranks. Here, the low-rank matrix decomposition is performed on a 5000×5000 square matrix.



(a) $p = 10, q = 0$. (b) $p = 10, q = 2$.

Fig. 5.4 Average runtime of the CPU and GPU accelerated randomized singular value decomposition for varying target ranks. Here, the low-rank matrix decomposition is performed on a 10000×10000 square matrix.

Figure 5.5 shows the performance for square random matrices of varying dimensions without power iterations. For small matrices the GPU accelerated implementations shows no computational advantage. The break-even point is given by matrices of dimension about 3000×3000. The weak performance for small matrices is mainly due-to some overhead of the **scikit-CUDA** implementation as well as due to the costs of data transfer between the GPU and the fast memory.

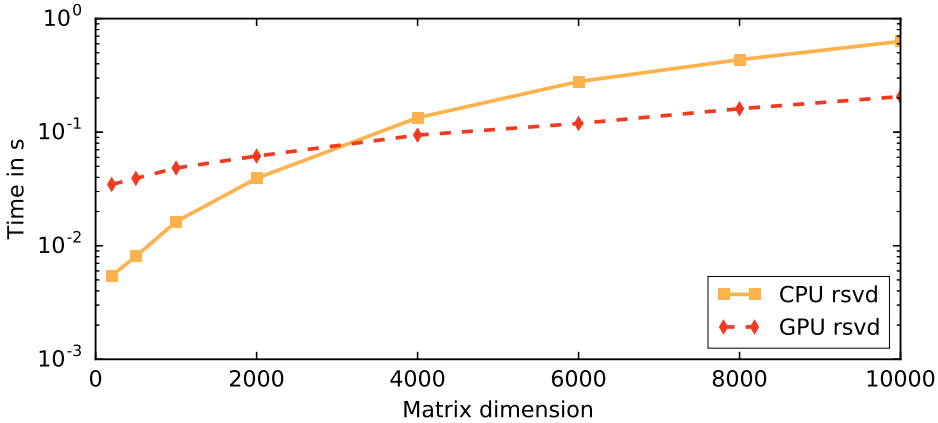
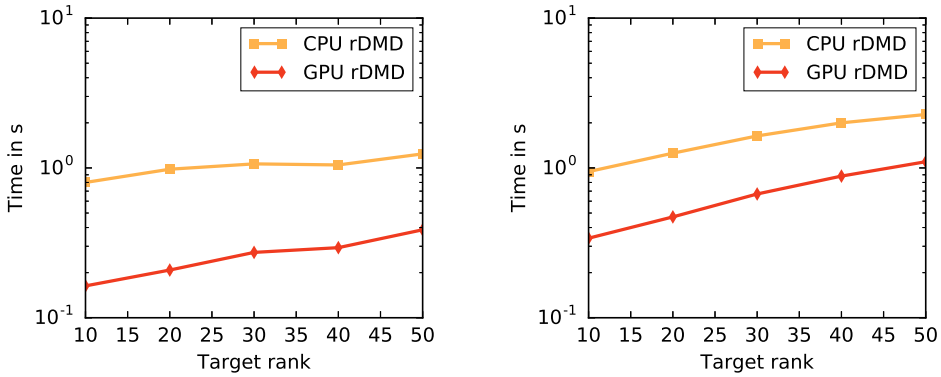


Fig. 5.5 Average runtime of CPU and GPU accelerated randomized singular value decomposition for varying matrix dimensions. Break-even point is given by matrices of dimension about 3000×3000.

5.4.2 Randomized Dynamic Mode Decomposition

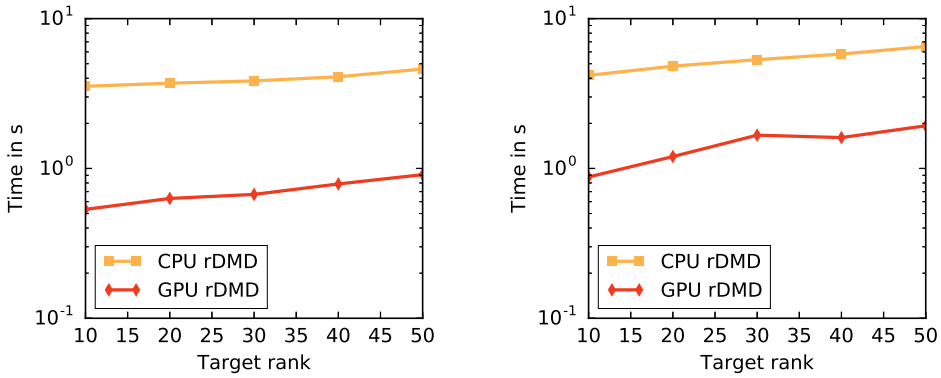
Next, we compare the computational performance of the CPU and GPU accelerated randomized dynamic mode decomposition. Here, we use tall random matrices, instead of square matrices. This is, because we normally face a (relatively) small sequence of high-dimensional snapshots in the area of fluid dynamics or video processing. Figure 5.5 shows the performance for a 100000×200 random matrix with and without power iterations. The GPU accelerated implementation achieves computational savings in both cases. As before, the computational savings become more distinct with increasing dimensions. Figure 5.5 shows the performance for a 200000×500 random matrix. Here speedups of about $2 - 3.5$ are achieved.

Figure 5.8 shows the computational advantage of the GPU accelerated implementation for varying video resolutions. Here, the performance is measured as the average processed frames per second (fps). The target rank is set to $k = 10$, and no additional power iterations are computed. For instance, the decomposition of a high definition (HD) 1280×720 video feed using the GPU accelerated implementation achieves a speedup of about 5 over the CPU implementation. More substantial, a speedup of about 24 is achieved over the deterministic DMD implementation.



(a) $p = 10, q = 0$. (b) $p = 10, q = 2$.

Fig. 5.6 Average runtime of CPU and GPU accelerated randomized dynamic mode decomposition for varying target ranks. Here, the low-rank matrix decomposition is performed on a 100000×200 square matrix.

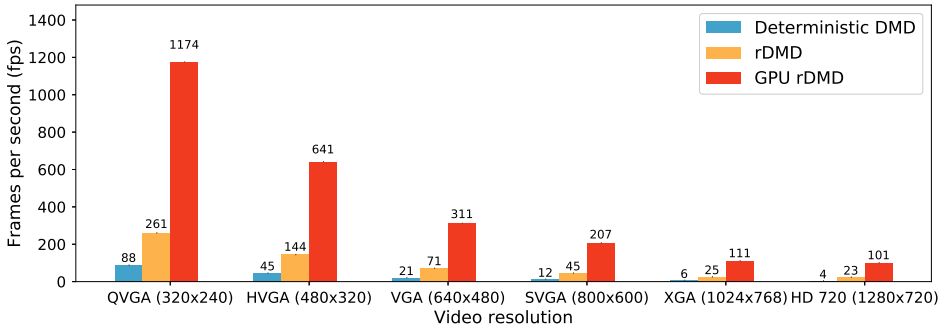


(a) $p = 10, q = 0$. (b) $p = 10, q = 2$.

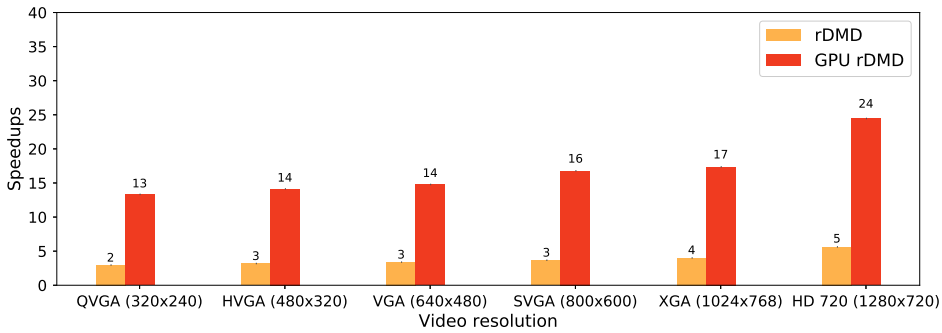
Fig. 5.7 Average runtime of CPU and GPU accelerated randomized dynamic mode decomposition for varying target ranks. Here, the low-rank matrix decomposition is performed on a 200000×500 square matrix.

5.5 Conclusion

The paradigm of parallel computing becomes increasingly important in the area of 'big data'. In particular, first the computational power of modern GPU architectures made the training of deep neural networks possible in practice. The computation, for instance, of the deterministic SVD, the



(a) Average runtime.



(b) Speedup.

Fig. 5.8 Average runtime of the CPU and GPU DMD algorithms for varying video resolutions. Here, 200 frames are used and the low-rank approximation is computed with target-rank $k = 25$.

eigenvalue decomposition and the least-square solver can benefit from the GPU architecture as well. Randomized matrix decompositions can benefit by design even more from a GPU accelerated implementation. This is, because the expensive computational steps (generating the test matrix $\mathbf{\Omega}$ as well as computing the sample matrix $\mathbf{Y} = \mathbf{A\Omega}$) are embarrassingly parallel. However, the disadvantage of current GPUs is the rather limited bandwidth, i.e., the amount of data which can be exchanged per unit of time, between CPU and GPU memory.

Here, just a GeForce GTX 950M was used for the computations. This, is a low-performance GPU, and thus the computational performance compared to the *MKL* (Intel Math Kernel Library) accelerated routines is not that significant. However, specialized GPUs like the Tesla K40 or GTX Titan are substantially more powerful and show a much better performance gain.

Chapter 6

Randomized CP Decomposition

"In mathematics you don't understand things.
You just get used to them."
— John von Neumann

Note: The work described in this chapter was carried out in collaboration with Professors J. Nathan Kutz and Steven L. Brunton of University of Washington and Krithika Manohar. It is submitted to the Journal of Computational Statistics and Data Analysis under the title: 'Randomized CP Tensor Decomposition'. My contributions involve conceptualizing the project idea, implementing the routines, running the simulations as well as writing the original draft.

6.1 Introduction

Numerous applications across the physical, biological, social and engineering sciences generate multi-modal data. In previous chapters, we have encountered already some prominent examples like fluid flows and video footage. These type of data arise naturally as 3 or higher-modal data structures. In order to be able to employ traditionally matrix decompositions techniques such as the singular value decomposition or the dynamic mode decomposition we were required to flatten the data first. However, this process of reshaping the data into a 2-dimensional array can lead to an information loss, because some of the relative locational information of the entries to one another get lost. Thus, subsequently matrix

decomposition can fail to reveal important structures in the data. Tensor decompositions overcome this issue of information loss. The canonical CP (*CANDECOMP/PARAFAC*) decomposition is particularly suitable for data-driven discovery since it expresses a tensor as a sum of rank-one tensors. Specifically, this simpler representation helps to reduce the dimensionality as well as to better understand the underlying properties of the data.

However, tensor decompositions of massive multidimensional data pose a tremendous computational challenge. Hence, innovations that reduce the computational demands have become increasingly relevant in this field. A key concept to ease the computational challenges is the idea of tensor compression. This involves the computation of a smaller (compressed) tensor, which is then used as a proxy to efficiently approximate the CP decomposition. Such a compressed tensor can be obtained, for instance, using the Tucker decomposition (Bro and Andersson, 1998; De Lathauwer et al., 2000). However, this approach requires the expensive computation of the left singular vectors for each mode. This computational challenge can be eased using modern randomized techniques developed to compute the singular value decomposition. Tsourakakis (2010) presents a randomized Tucker decomposition algorithm based on the work of Achlioptas and McSherry (2007). Later, Zhou et al. (2014) proposed a more refined randomized CP algorithm using the ideas of Halko et al. (2011b), however, they omit the important concept of power iterations in their work. As an alternative to random projections, Drineas and Mahoney (2007) proposed a randomized tensor algorithm based on the idea of random column selection. Related work by Vervliet et al. (2014) proposed a sparsity-promoting algorithm for incomplete tensors using concepts of compressed sensing. Alternatively, a different approach to efficiently compute large-scale tensor decompositions is based on the idea of subdividing a tensor into a set of blocks. These smaller blocks can then be used to approximate the CP decomposition of the full tensor in a parallelized or distributed fashion (Phan and Cichocki, 2011). Sidiropoulos et al. (2014) fused the idea of randomization (random projections) and blocking into a highly computationally efficient algorithm. More recently, Vervliet and Lathauwer (2016) also proposed a block sampling CP decomposi-

tion method for the analysis of large-scale tensors using randomization as computational strategy. The presented numerical evaluations show significant computational savings, while attaining near optimal accuracy. These block based algorithms are particularly relevant if the tensor is too large to fit into fast memory.

Motivation and Overview

Previously we have seen the computational benefits of embedding low-rank matrix approximations into the powerful probabilistic framework outlined in Chapter 1. Here, our aim is to generalize this framework to multi-modal data. Again, the idea is to build a suitable basis using random projections. Once a good basis is obtained, the high-dimensional tensor is projected onto this low-dimensional space. The resulting smaller tensor can then be used to obtain the CP decomposition. Once the approximate factor matrices are computed, the full factor matrices can be efficiently recovered using the orthonormal basis matrices. A crucial aspect to build a sufficient basis are power iterations, however, as far as we are aware, it has not been applied in the context of tensors before. In particular, in the presence of white noise the performance of randomized algorithms based on random projections can suffer considerably without additional power iterations. Thus, both the concept of oversampling and power iterations allow one to control the error of the decomposition. Our numerical results show outstanding computational savings, while achieving a near-optimal approximation quality. In addition, we provide an open-software package written in Python, which allows the reproduction of all results. The **cTensor** package can be obtained from the GIT repository: https://github.com/Benli11/rtensor.

The remainder of this manuscript is organized as follows. Section 6.2 briefly introduces some additional tensor notation. Then, Section 6.3 reviews the CP decomposition. Section 6.4 presents an efficient randomized tensor algorithm, which is used to compute the CP tensor decomposition as discussed in Section 6.5. The algorithm is motivated using both alternating least squares and a block coordinate descent approach. Section 6.7 shows the computational performance using both synthetic and real world

examples. Finally, Section 6.8 summarizes the research findings and outlines further directions.

6.2 Some Tensor Notation

A tensor can be seen as a multi-index numerical array. The order of a tensor is the number of its modes or dimensions. A real-valued tensor of order 3 is denoted by $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ in the standard notation. A tensor element x_{ijk} is indexed by the three integers i, j and k. This concept can be generalized to N-way tensors denoted by $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$.

Tensor decompositions are computed on tensor modes, which are flattened or *unfolded*, *matricized* versions of the N-way array. An N-way tensor has N different modal flattenings or unfoldings along each of its N different dimensions. For example, the mode-3 unfolding of $\boldsymbol{\mathcal{X}}$ is formed by holding the third index k fixed, creating K number of $I \times J$ matrices denoted $\mathbf{X}_{::k}$, then flattening these matrices into vectors $\mathbf{x}_{::k}$. The mode-3 unfolding $\boldsymbol{\mathcal{X}}_{(3)} \in \mathbb{R}^{K \times (I \cdot J)}$ results then from stacking these as row vectors within

$$\boldsymbol{\mathcal{X}}_{(3)} = \begin{bmatrix} \mathbf{x}_{::1}^T \\ \mathbf{x}_{::2}^T \\ \vdots \\ \mathbf{x}_{::K}^T \end{bmatrix}.$$

Several matrix product operators are used in computation on the flattened modes, which we briefly reproduce here. The matrix *Kronecker product* of two matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$ is

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \dots & a_{1J}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \dots & a_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \dots & a_{IJ}\mathbf{B} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_1 \otimes \mathbf{b}_2 & \dots & \mathbf{a}_J \otimes \mathbf{b}_{L-1} & \mathbf{a}_J \otimes \mathbf{b}_L \end{bmatrix}.$$

The *Khatri-Rao product* of $\mathbf{A} \in \mathbb{R}^{I \times K}$ and $\mathbf{B} \in \mathbb{R}^{J \times K}$ is given by the columnwise Kronecker product

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \dots & \mathbf{a}_K \otimes \mathbf{b}_K \end{bmatrix}.$$

Finally, the *Hadamard product* of two equal size matrices is simply the elementwise multiplication

$$(\mathbf{A} * \mathbf{B})_{ij} = a_{ij} b_{ij}.$$

These operators are discussed in more detail by Kolda and Bader (2009). Further, the inner product of two tensors is expressed as $\langle \cdot, \cdot \rangle$.

6.3 Deterministic CP Decomposition

In the past, computation was severely inhibited by available computational power. Today, tensor decompositions enjoy more popularity, yet runtime bottlenecks still persist. Ideas for multi-way factor analysis emerged in the 1920s with the formulation of the polyadic decomposition by Hitchcock (1927). However, the polyadic tensor decomposition only achieved popularity much later in the 1970s with the canonical decomposition (*CANDECOMP*) introduced by Carroll and Chang (1970) in psychometrics. At the same time, Harshman (1970) introduced the method of parallel factors (*PARAFAC*) in chemometrics. Hence, this method became known as the CP (*CANDECOMP/PARAFAC*) decomposition. The CP decomposition is the natural tensor equivalent of the singular value decomposition, since it approximates a tensor by a sum of rank-one tensors. Specifically, *tensor rank* is defined as the smallest sum of rank-one tensors required to generate the tensor (Hitchcock, 1927). The CP decomposition can be used to approximate these rank-one tensors. Given a third order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$, the rank-$R$ CP decomposition is expressed as

$$\boldsymbol{\mathcal{X}} \approx \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r, \tag{6.1}$$

where $\circ$ denotes the outer product. Specifically, each rank-one tensor is formulated as the outer product of the rank-one components $\mathbf{a}_r \in \mathbb{R}^I$, $\mathbf{b}_r \in \mathbb{R}^J$, and $\mathbf{c}_r \in \mathbb{R}^K$. Components are often constrained to unit length with the weights absorbed into the vector $\boldsymbol{\lambda} = [\lambda_1, ..., \lambda_R] \in \mathbb{R}^R$. Then,

Equation (6.1) can be re-expressed as

$$\boldsymbol{\mathcal{X}} \approx \sum_{r=1}^{R} \lambda_r \cdot \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r. \tag{6.2}$$

More compactly the components can be expressed as factor matrices, i.e., $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_R]$, $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_R]$, and $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_R]$. Using the Kruskal operator as defined by Kolda and Bader (2009), Equation (6.2) can be more compactly expressed as

$$\boldsymbol{\mathcal{X}} \approx [\![\boldsymbol{\lambda}; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!].$$

The CP decomposition is illustrated in Figure 6.1. For more technical details, we refer to Kolda and Bader (2009) and Smilde et al. (2005).

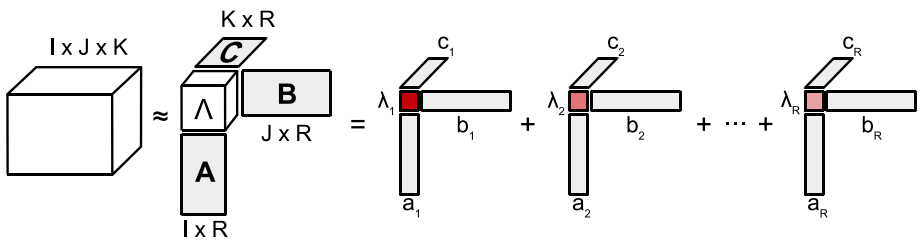


Fig. 6.1 Schematic of the CP decomposition.

An astonishing property of the CP decomposition is the parsimonious data representation. This becomes evident comparing the compression ratio of the CP to the SVD. For a third order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ of rank R, the CP decomposition achieves the following compression ratio

$$c_{CP} = \frac{I \cdot J \cdot K}{R \cdot (I + J + K + 1)}$$

In contrast, the singular value decomposition requires to reshape the tensor into a 2-dimensional array first. Thus the compression ratio is as follows

$$c_{SVD} = \frac{I \cdot J \cdot K}{R \cdot (I \cdot J + K + 1)}$$

Figure 6.2 illustrates the compression performance of both methods for a tensor of dimension $100 \times 100 \times 100$. The compression performance is

striking, however, the downside is that the CP decomposition does not reconstruct the data as accurate as the SVD does.

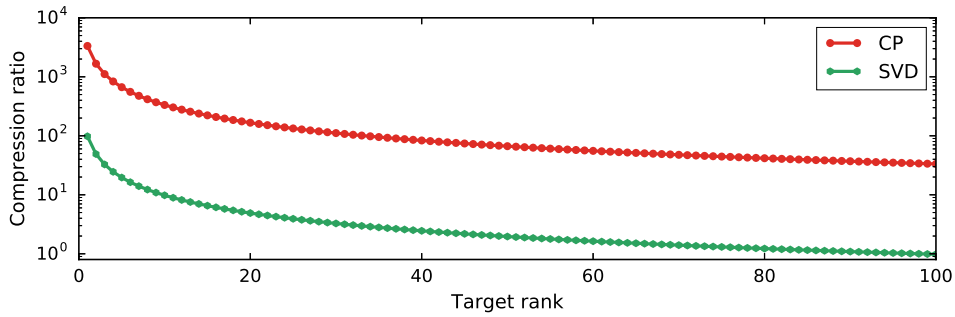


Fig. 6.2 Compression ratio of the CP singular value decomposition for varying rank-R approximations. The achieved compression rate of the CP decomposition is substantial.

6.4 Randomized Tensor Algorithm

The aim is to use randomization as a computational strategy to efficiently build a suitable basis that captures the action of the tensor $\boldsymbol{\mathcal{X}}$. Therefor, we generalize the concepts of randomized matrix algorithms to tensors. In particular, we build upon the methods introduced by Martinsson et al. (2011) and Halko et al. (2011b), as well as related work on randomized tensors by Drineas and Mahoney (2007), who proposed a randomized algorithm based on random column selection.

Assuming a N-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, the aim is to obtain a smaller tensor $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{k \times \cdots \times k}$, so that its N tensor modes capture the action of the input tensor modes. Hence, we seek a natural basis in the form of a set of orthonormal matrices $\{\mathbf{Q}_n \in \mathbb{R}^{I_n \times R_n}\}_{n=1}^N$, so that

$$\boldsymbol{\mathcal{X}} \approx \boldsymbol{\mathcal{X}} \times_1 \mathbf{Q}_1\mathbf{Q}_1^\top \times_2 \cdots \times_N \mathbf{Q}_N\mathbf{Q}_N^\top \tag{6.3}$$

is satisfied. Here the operator $\times_n$ denotes tensor-matrix multiplication defined as follows.

Definition 1. The n-mode matrix product $\boldsymbol{\mathcal{X}} \times_n \mathbf{Q}_n\mathbf{Q}_n^\top$ multiplies a tensor by the matrices $\mathbf{Q}_n\mathbf{Q}_n^\top$ in mode n, i.e., each mode-n fiber is multiplied by

$\mathbf{Q}_n\mathbf{Q}_n^\top$

$$\boldsymbol{\mathcal{M}} = \boldsymbol{\mathcal{X}} \times_n \mathbf{Q}_n\mathbf{Q}_n^\top \quad \Leftrightarrow \quad \mathbf{M}_{(n)} = \mathbf{Q}_n\mathbf{Q}_n^\top \boldsymbol{\mathcal{X}}_{(n)}.$$

Given a fixed target rank k, these basis matrices can be efficiently obtained using a randomized algorithm. Specifically, the approximate basis for the n-th tensor mode is obtained by first drawing k random vectors $\omega_1, \ldots, \omega_k$ from a Gaussian distribution. These random vectors form the measurement matrix $\boldsymbol{\Omega}_n \in \mathbb{R}^{\sum_{i\neq n} I_i \times k}$, which is used to sample the column space of $\boldsymbol{\mathcal{X}}_{(n)} \in \mathbb{R}^{I_n \times \sum_{i\neq n} I_i}$ as follows

$$\mathbf{Y}_n = \boldsymbol{\mathcal{X}}_{(n)}\boldsymbol{\Omega}_n, \tag{6.4}$$

where $\mathbf{Y}_n \in \mathbb{R}^{I_n \times k}$ is the sample matrix. The sample matrix serves as an approximate basis for the range of the n-th tensor mode. Probability theory guarantees that the set of random vectors $\{\omega_i\}_{i=1}^k$ are linearly independent with high probability. Hence, the corresponding random projections $\mathbf{y}_1, \ldots, \mathbf{y}_k$ efficiently sample the range of a rank deficient tensor mode $\boldsymbol{\mathcal{X}}_{(n)}$. The economic QR decomposition of the sample matrix $\mathbf{Y}_n = \mathbf{Q}_n\mathbf{R}_n$ is then used to obtain a natural basis, so that $\mathbf{Q}_n \in \mathbb{R}^{I_n \times k}$ is orthonormal and has the same column space as $\mathbf{Y}_n$. The final step restricts the tensor mode to this low-dimensional subspace

$$\boldsymbol{\mathcal{B}}_n = \boldsymbol{\mathcal{X}} \times_n \mathbf{Q}_n^\top \quad \Leftrightarrow \quad \mathbf{B}_n = \mathbf{Q}_n^\top \boldsymbol{\mathcal{X}}_{(n)}. \tag{6.5}$$

Thus, after N iterations a compressed tensor $\boldsymbol{\mathcal{B}}$ and a set of orthonormal matrices is obtained. Since this is an iterative algorithm, we set $\boldsymbol{\mathcal{X}} \leftarrow \boldsymbol{\mathcal{B}}_n$ after each iteration. The number of columns of the basis matrices form a trade-off between accuracy and computational performance. Thus the aim is to use as few columns as possible, yet allow an accurate approximation of the input tensor. Assuming that the tensor $\boldsymbol{\mathcal{X}}$ exhibits low-rank structure, or equivalently, the rank R is much smaller than the ambient dimensions of the tensor, the basis matrices will be a efficient representation. However, to improve the performance of the basis, we allow for additional oversampling in practice. This means, instead of drawing exactly k random vectors, it is preferred to draw $l = k+p$ random vectors, where p denotes the oversampling parameter. In practice, a small

oversampling parameter is often sufficient, i.e., $p = \{5, 10\}$. Moreover, it is important to note that it may be beneficial to assign different oversampling parameters to different modes. Indeed, most real world tensors exhibit distinct tensor modes, and some of them may not be low-rank.

The randomized algorithm as presented requires that the mode-n unfolding of the tensor has a rapidly decaying spectrum in order to achieve good performance. However, this assumption is often not suitable. In particular, the spectrum starts to decay slowly if the tensor is compressed several times. To overcome this issue, the algorithm's performance can be substantially improved using power iterations (Gu, 2015; Halko et al., 2011b; Rokhlin et al., 2009). Specifically, power iterations turn a slowly decaying spectrum into a rapidly decaying one by taking powers of the tensor modes. Thus, instead of sampling $\boldsymbol{\mathcal{X}}_{(n)}$ the idea is to sample from the following tensor mode

$$\boldsymbol{\mathcal{X}}^q_{(n)} := (\boldsymbol{\mathcal{X}}_{(n)}\boldsymbol{\mathcal{X}}^\top_{(n)})^q\boldsymbol{\mathcal{X}}_{(n)},$$

where q denotes a small integer. This power operation ensures that the singular values of $\boldsymbol{\mathcal{X}}^q_{(n)}$ are $\{\sigma_j^{2q+1}\}_j$. Now, instead of using Equation (6.4), the improved sample matrix can be computed from

$$\mathbf{Y}_n = (\boldsymbol{\mathcal{X}}_{(n)}\boldsymbol{\mathcal{X}}^\top_{(n)})^q\boldsymbol{\mathcal{X}}_{(n)}\boldsymbol{\Omega}_n. \tag{6.6}$$

However, if Equation (6.6) is implemented in this form it tends to distort the basis due to round-off errors. Therefore, in practice (normalized) subspace iterations are used to form the sample matrix. This means that the sample matrix is orthornormalized between each power iteration in order to stabilize the algorithm. For implementation details see Voronin and Martinsson (2015) or Szlam et al. (2014).

The combination of oversampling and additional power iterations can be used to control the trade-off between approximation quality and computational efficiency of the randomized tensor algorithm. Our results, for example, show that just $q = 2$ subspace iterations and an oversampling parameter of about $p = 10$ achieves near-optimal results. Algorithm 6 summarizes the computational steps.

Algorithm 6 A prototype randomized tensor compression algorithm.

Require: An N-way tensor $\boldsymbol{\mathcal{X}}$, and a desired target rank k.

Optional: Parameters p and q to control oversampling, and the number of power iterations.

(1)	$\boldsymbol{\mathcal{B}} = \boldsymbol{\mathcal{X}}$	Initialize compressed tensor.
(2)	**for** $n = 1, \ldots, N$	Iterate over all tensor modes.
(3)	$\quad l = k + p$	Slight oversampling.
(4)	$\quad I, J = \texttt{dim}(\boldsymbol{\mathcal{B}}_{(n)})$	Dimension of the n-th tensor mode.
(5)	$\quad \boldsymbol{\Omega} = \texttt{rand}(J, l)$	Generate random matrix.
(6)	$\quad \mathbf{Y} = \boldsymbol{\mathcal{B}}_{(n)}\boldsymbol{\Omega}$	Compute sampling matrix.
(7)	$\quad$ **for** $j = 1, \ldots, q$	Power iterations (optional).
(8)	$\quad\quad [\mathbf{Q}, \sim] = \texttt{qr}(\mathbf{Y})$	
(9)	$\quad\quad [\mathbf{Z}, \sim] = \texttt{qr}(\boldsymbol{\mathcal{B}}_{(n)}^\top \mathbf{Q})$	
(10)	$\quad\quad \mathbf{Y} = \boldsymbol{\mathcal{B}}_{(n)}\mathbf{Z}$	
(11)	$\quad$ **end for**	
(12)	$\quad [\mathbf{Q}_n, \sim] = \texttt{qr}(\mathbf{Y})$	Orthonormalize sampling matrix.
(13)	$\quad \boldsymbol{\mathcal{B}} = \boldsymbol{\mathcal{B}} \times_n \mathbf{Q}_n^\top$	Project tensor to smaller space.
(14)	**end for**	

Return: Compressed tensor $\boldsymbol{\mathcal{B}}$ of dimension $l \times \cdots \times l$, and a set of orthonormal basis matrices $\{\mathbf{Q}_n \in \mathbb{R}^{I_n \times l}\}_{n=1}^N$.

Remark 3. For better numerical stability, subspace iterations using the QR decomposition are computed in step 7-11. We recommend a default value of $q = 2$.

Remark 4. In practice, the user can decide which modes to compress and specify different oversampling parameters for these modes.

Performance analysis

In the following the average-case behavior of the randomized tensor algorithm is characterized. In particular, we are interested in the expected residual error

$$\mathsf{E}\|\boldsymbol{\mathcal{E}}\|_F = \|\boldsymbol{\mathcal{X}} - \hat{\boldsymbol{\mathcal{X}}}\|_F, \tag{6.7}$$

where $\hat{\boldsymbol{\mathcal{X}}} = \boldsymbol{\mathcal{X}} \times_1 \mathbf{Q}_1\mathbf{Q}_1^\top \times_2 \cdots \times_N \mathbf{Q}_N\mathbf{Q}_N^\top$. Theorem 1 follows as a generalization from Theorem 10.5 formulated by Halko et al. (2011b).

Theorem 1 (Expected Frobenius error). Consider a low-rank real N-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$. Then the expected approximation error, given a target rank $k \geq 2$ and an oversampling parameter $p \geq 2$ for each mode, is

$$\mathsf{E}\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_1 \mathbf{Q}_1\mathbf{Q}_1^\top \times_2 \cdots \times_N \mathbf{Q}_N\mathbf{Q}_N^\top\|_F \leq \sqrt{1 + \tfrac{k}{p-1}} \cdot \sqrt{\textstyle\sum_{n=1}^N \sum_{j>k} \sigma_{nj}^2}.$$

The proof is shown in Appendix A. Intuitively, the projection of each tensor mode onto a low-dimensional space introduces an additional

residual. This is expressed by the double sum on the right hand side. If the low-rank approximation captures the column space of each mode accurately, then the singular values $j > k$ for each mode n are small. Further, it can be seen that the error can be improved by the oversampling parameter.

The computation of additional power (subspace) iterations can improve the error further. This result again follows by generalizing the results of Halko et al. (2011b) to tensors. Sharper performance bounds for both oversampling and additional power iterations can be derived following, for instance, the results by Witten and Candes (2015).

6.5 Randomized CP Decomposition

6.5.1 Conceptual Overview

Given a third order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$, the objective of the CP decomposition is to find a set of R normalized rank-one tensors $\{\mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r\}_{r=1}^{R}$ which best approximates $\boldsymbol{\mathcal{X}}$, i.e., minimizes the Frobenius norm

$$\underset{\hat{\boldsymbol{\mathcal{X}}}}{\text{minimize}} \quad \|\boldsymbol{\mathcal{X}} - \hat{\boldsymbol{\mathcal{X}}}\|_F^2 \quad \text{subject to} \quad \hat{\boldsymbol{\mathcal{X}}} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r. \tag{6.8}$$

If the dimensions of $\boldsymbol{\mathcal{X}}$ are large, the computational costs of solving this optimization problem can be enormous. However, for obtaining the factor matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ only the column spaces are of importance, rather then the individual columns of the mode $\boldsymbol{\mathcal{X}}_{(1)}$, $\boldsymbol{\mathcal{X}}_{(2)}$, $\boldsymbol{\mathcal{X}}_{(3)}$ matricizations of the tensor $\boldsymbol{\mathcal{X}}$. This is because the CP decomposition learns the components based on proportional variations in inter-point distances between the components. Therefore, the small tensor $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{k \times k \times k}$ must preserve pairwise Euclidean distances, where $k \geq R$. This in turn requires that column spaces, and thus pairwise distances, are approximately preserved which is achieved by design in the randomized tensor algorithm presented in the previous section. Figure 6.3 shows the schematic of the randomized CP decomposition architecture.

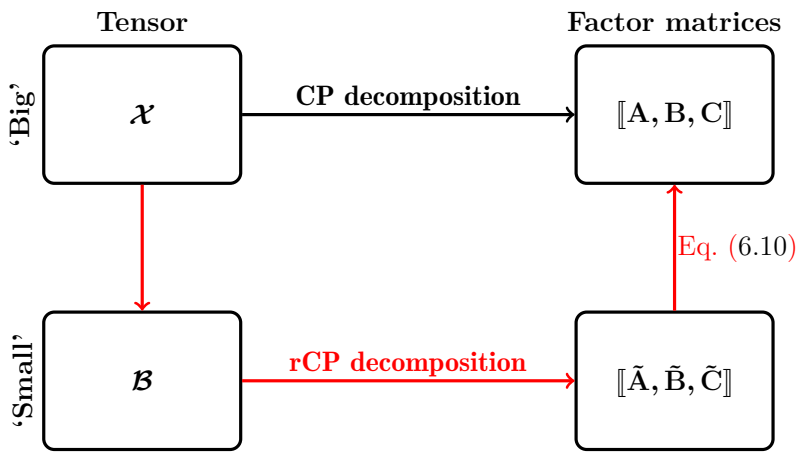


Fig. 6.3 Schematic of the randomized CP decomposition architecture. The tensor $\mathcal{X}$ is first compressed using random projections. Then the CP decomposition is performed on the small tensor $\mathcal{B}$. Finally, the factor matrices $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ are recovered from the compressed factor matrices $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}$ and $\tilde{\mathbf{C}}$ using Eq. (6.10).

6.5.2 Randomized Algorithm

While there exist several optimization strategies for minimizing the objective function defined in Equation (6.9), we consider alternating least squares (ALS) and block coordinate descent (BCD). Both methods are suitable to deal with a compressed tensor $\mathcal{B} \in \mathbb{R}^{k \times \cdots \times k}$, where $k \geq R$. Specifically, the optimization problem in Equation (6.8) is reformulated

$$\underset{\hat{\mathcal{B}}}{\text{minimize}} \quad \|\mathcal{B} - \hat{\mathcal{B}}\|_F^2 \quad \text{subject to} \quad \hat{\mathcal{B}} = \sum_{r=1}^{R} \tilde{\mathbf{a}}_r \circ \tilde{\mathbf{b}}_r \circ \tilde{\mathbf{c}}_r, \tag{6.9}$$

where $\tilde{\mathbf{a}}_r, \tilde{\mathbf{b}}_r$ and $\tilde{\mathbf{c}}_r$ denote the compressed rank-one components. Once the compressed factor matrices $\tilde{\mathbf{A}} \in \mathbb{R}^{k \times R}$, $\tilde{\mathbf{B}} \in \mathbb{R}^{k \times R}$, $\tilde{\mathbf{C}} \in \mathbb{R}^{k \times R}$ are estimated, the full factor matrices can be recovered from

$$\begin{aligned} \mathbf{A} &\approx \mathbf{Q}_1 \tilde{\mathbf{A}}, \\ \mathbf{B} &\approx \mathbf{Q}_2 \tilde{\mathbf{B}}, \\ \mathbf{C} &\approx \mathbf{Q}_3 \tilde{\mathbf{C}}, \end{aligned} \tag{6.10}$$

where $\mathbf{Q}_1 \in \mathbb{R}^{I\times k}$, $\mathbf{Q}_2 \in \mathbb{R}^{J\times k}$, $\mathbf{Q}_3 \in \mathbb{R}^{K\times k}$ denote the orthonormal basis matrices. For simplicity we focus on third order tensors, but the concept generalizes to N-way tensors.

Alternating Least Squares Algorithm

Alternating least squares is the most popular method for computing the CP decomposition (Comon et al., 2009; Kolda and Bader, 2009). This is because the algorithm is simple and efficient. First we note that the optimization problem in Equation (6.9) is equivalent to

$$\underset{\tilde{\mathbf{A}},\tilde{\mathbf{B}},\tilde{\mathbf{C}}}{\text{minimize}} \quad \|\boldsymbol{\mathcal{B}} - \sum_{r=1}^{R} \tilde{\mathbf{a}}_r \circ \tilde{\mathbf{b}}_r \circ \tilde{\mathbf{c}}_r\|_F^2$$

with respect to the factor matrices $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}$ and $\tilde{\mathbf{C}}$. Further, the tensor $\boldsymbol{\mathcal{B}}$ can be expressed in matricized form

$$\begin{aligned}
\boldsymbol{\mathcal{B}}_{(1)} &\approx \tilde{\mathbf{A}}(\tilde{\mathbf{C}} \odot \tilde{\mathbf{B}})^\top, \\
\boldsymbol{\mathcal{B}}_{(2)} &\approx \tilde{\mathbf{B}}(\tilde{\mathbf{C}} \odot \tilde{\mathbf{A}})^\top, \\
\boldsymbol{\mathcal{B}}_{(3)} &\approx \tilde{\mathbf{C}}(\tilde{\mathbf{B}} \odot \mathbf{A})^\top,
\end{aligned}$$

where $\odot$ denotes the Khatri-Rao product. The optimization problem in this form is non-convex, however, an estimate for the factor matrices can be obtained using the least-squares method. Therefore, the ALS algorithm updates one component, holding the other two components fixed, in an alternating fashion until convergence. Specifically, the algorithm iterates over the following subproblems

$$\tilde{\mathbf{A}}^{j+1} = \underset{\tilde{\mathbf{A}}}{\operatorname{argmin}} \|\boldsymbol{\mathcal{B}}_{(1)} - \tilde{\mathbf{A}}(\tilde{\mathbf{C}}^j \odot \tilde{\mathbf{B}}^j)^\top\|, \tag{6.11}$$

$$\tilde{\mathbf{B}}^{j+1} = \underset{\tilde{\mathbf{B}}}{\operatorname{argmin}} \|\boldsymbol{\mathcal{B}}_{(2)} - \tilde{\mathbf{B}}(\tilde{\mathbf{C}}^j \odot \tilde{\mathbf{A}}^{j+1})^\top\|, \tag{6.12}$$

$$\tilde{\mathbf{C}}^{j+1} = \underset{\tilde{\mathbf{C}}}{\operatorname{argmin}} \|\boldsymbol{\mathcal{B}}_{(3)} - \tilde{\mathbf{C}}(\tilde{\mathbf{B}}^{j+1} \odot \tilde{\mathbf{A}}^{j+1})^\top\|. \tag{6.13}$$

Thus, each step involves a least-squares problem which can be solved using the Khatri-Rao product pseudo-inverse. Algorithm 7 summarizes the computational steps.

Definition 2. The Khatri-Rao product pseudo-inverse is defined as

$$(\mathbf{A} \odot \mathbf{B})^{\dagger} = (\mathbf{A}^{\top}\mathbf{A} * \mathbf{B}^{\top}\mathbf{B})^{\dagger}(\mathbf{A} \odot \mathbf{B})^{\top},$$

where the operator $*$ denotes the Hadamard product, i.e., the elementwise multiplication of two equal sized matrices.

There exist few general convergence guarantees for the alternating least squares algorithm, see for example the work by Uschmajew (2012) and Wang and Chu (2014). Moreover, the final solution tends to depend on the initial guess $\tilde{\mathbf{A}}^{\mathbf{0}}, \tilde{\mathbf{B}}^{\mathbf{0}}$ and $\tilde{\mathbf{C}}^{\mathbf{0}}$. A standard initial guess uses the eigenvectors of $\boldsymbol{\mathcal{B}}_{(1)}\boldsymbol{\mathcal{B}}_{(1)}^{\top}$, $\boldsymbol{\mathcal{B}}_{(2)}\boldsymbol{\mathcal{B}}_{(2)}^{\top}$, $\boldsymbol{\mathcal{B}}_{(3)}\boldsymbol{\mathcal{B}}_{(3)}^{\top}$ (Bader and Kolda, 2015). Further, it is important to note that normalization of the factor matrices is necessary after each iteration to achieve good convergence. Specifically, this prevents singularities of the Khatri-Rao product pseudo-inverse (Kolda and Bader, 2009). The algorithm can be further improved by reformulating the above subproblems as regularized least-squares problems, for instance, see Li et al. (2013) for technical details and convergence results. The ability of the alternating least squares algorithm to impose structure on the factor matrices permits the formulation of non-negative, or sparsity-constrained tensor decompositions (Cichocki et al., 2009).

Algorithm 7 A prototype randomized CP algorithm using ALS.

Require: An $I \times J \times K$ tensor $\boldsymbol{\mathcal{X}}$, and a desired target rank R.
Optional: Parameters p and q to control oversampling, and the number of power iterations.

(1) $\boldsymbol{\mathcal{B}}, \mathbf{Q}_A, \mathbf{Q}_B, \mathbf{Q}_C = \texttt{compress}(\boldsymbol{\mathcal{X}}, R, p, q)$ — compress tensor using Algorithm 6
(2) $\mathbf{B}, \mathbf{C} = [\texttt{eig}(\boldsymbol{\mathcal{B}}_{(2)}, \boldsymbol{\mathcal{B}}_{(2)}^{\top}), \texttt{eig}(\boldsymbol{\mathcal{B}}_{(3)}, \boldsymbol{\mathcal{B}}_{(3)}^{\top})]$ — use first R eigenvectors for initialization
(3) **repeat**
(4) $\quad \mathbf{A} = \boldsymbol{\mathcal{B}}_{(1)}(\mathbf{C} \odot \mathbf{B})(\mathbf{C}^{\top}\mathbf{C} * \mathbf{B}^{\top}\mathbf{B})^{\dagger}$
(5) $\quad \mathbf{A} = \mathbf{A}/\texttt{norm}(\mathbf{A})$
(6) $\quad \mathbf{B} = \boldsymbol{\mathcal{B}}_{(2)}(\mathbf{C} \odot \mathbf{A})(\mathbf{C}^{\top}\mathbf{C} * \mathbf{A}^{\top}\mathbf{A})^{\dagger}$
(7) $\quad \mathbf{B} = \mathbf{B}/\texttt{norm}(\mathbf{B})$
(8) $\quad \mathbf{C} = \boldsymbol{\mathcal{B}}_{(3)}(\mathbf{B} \odot \mathbf{A})(\mathbf{B}^{\top}\mathbf{B} * \mathbf{A}^{\top}\mathbf{A})^{\dagger}$
(9) $\quad \boldsymbol{\lambda} = \texttt{norm}(\mathbf{C})$
(10) $\quad \mathbf{C} = \mathbf{C}/\boldsymbol{\lambda}$
(11) **until** convergence criterion is reached
(12) $\mathbf{A}, \mathbf{B}, \mathbf{C} = [\mathbf{Q}_A\mathbf{A}, \mathbf{Q}_B\mathbf{B}, \mathbf{Q}_C\mathbf{C}]$ — recover factor matrices
(13) re-normalize the factor matrices and update the scaling vector $\boldsymbol{\lambda}$

Return: Normalized factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and the scaling vector $\boldsymbol{\lambda}$.

Block Coordinate Descent Algorithm

While ALS is the most popular algorithm for computing the CP decomposition, many alternative algorithms have been developed. One particularly intriguing algorithm is based on the method of block coordinate descent (BCD) (Xu and Yin, 2013). First, Cichocki and Phan (2009) proposed this approach for computing nonnegative tensor factorizations. The BCD algorithm is based on the idea of successive rank-one deflation. Unlike ALS, which updates the entire factor matrix at each step, BCD computes the rank-1 tensors in a hierarchical fashion. Therefore, the algorithm treats each component $\mathbf{a}_r, \mathbf{b}_r, \mathbf{c}_r$ as a block. First, the most correlated rank-1 tensor is computed; then the second most correlated rank-1 tensor is learned on the residual tensor, and so on. Assuming that $\tilde{R} = r - 1$ components have been computed, then the r-th compressed residual tensor $\boldsymbol{\mathcal{Y}}_r$ is defined

$$\boldsymbol{\mathcal{Y}}_r = \boldsymbol{\mathcal{B}} - \sum_{r=1}^{\tilde{R}} \tilde{\mathbf{a}}_r \circ \tilde{\mathbf{b}}_r \circ \tilde{\mathbf{c}}_r. \tag{6.14}$$

Then, the algorithm iterates over the following subproblems

$$\tilde{\mathbf{a}}_r^{j+1} = \underset{\tilde{\mathbf{a}}_r}{\operatorname{argmin}} \|\boldsymbol{\mathcal{Y}}_{r(1)} - \tilde{\mathbf{a}}_r(\tilde{\mathbf{c}}_r^j \odot \tilde{\mathbf{b}}_r^j)^\top\|, \tag{6.15}$$

$$\tilde{\mathbf{b}}_r^{j+1} = \underset{\tilde{\mathbf{b}}_\mathbf{r}}{\operatorname{argmin}} \|\boldsymbol{\mathcal{Y}}_{r(2)} - \tilde{\mathbf{b}}_r(\tilde{\mathbf{c}}_r^j \odot \tilde{\mathbf{a}}_r^{j+1})^\top\|, \tag{6.16}$$

$$\tilde{\mathbf{c}}_r^{j+1} = \underset{\tilde{\mathbf{c}}_r}{\operatorname{argmin}} \|\boldsymbol{\mathcal{Y}}_{r(3)} - \tilde{\mathbf{c}}_r(\tilde{\mathbf{b}}_r^{j+1} \odot \tilde{\mathbf{a}}_r^{j+1})^\top\|. \tag{6.17}$$

Note that the computation can be more efficiently evaluated without explicitly constructing the residual tensor $\boldsymbol{\mathcal{Y}}_r$; see Kim et al. (2014) for further details. Algorithm 8 summarizes the computational steps of the BCD algorithm.

6.6 The rTensor Package

We have developed the package **rTensor**, which implements the previous presented algorithms in Python. The routines are based on the numerical linear algebra tools provided by the **SciPy** (Open Source Library of Scientific Tools) package (Jones et al., 2001). Specifically, **SciPy** provides MKL (Math Kernel Library) accelerated high performance implementations of

Algorithm 8 A prototype randomized CP algorithm using BCD.

Require: An $I \times J \times K$ tensor $\boldsymbol{\mathcal{X}}$, and a desired target rank R.
Optional: Parameters p and q to control oversampling, and the number of power iterations.

(1) $\boldsymbol{\mathcal{B}}, \mathbf{Q}_A, \mathbf{Q}_B, \mathbf{Q}_C = \texttt{compress}(\boldsymbol{\mathcal{X}}, R, p, q)$ — compress tensor using Algorithm 6
(2) $\mathbf{B}, \mathbf{C} = [\texttt{eig}(\boldsymbol{\mathcal{B}}_{(2)}, \boldsymbol{\mathcal{B}}_{(2)}^\top), \texttt{eig}(\boldsymbol{\mathcal{B}}_{(3)}, \boldsymbol{\mathcal{B}}_{(3)}^\top)]$ — use first R eigenvectors for initialization
(3) $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{B}}$ — initialize residual tensor
(4) **for** $r = 1, \ldots, R$ — compute rank-r approximation
(5) **repeat**
(6) $\mathbf{a}_r = \boldsymbol{\mathcal{Y}}_{(1)}(\mathbf{c}_r \odot \mathbf{b}_r)(\mathbf{c}_r^\top \mathbf{c}_r * \mathbf{b}_r^\top \mathbf{b}_r)^\dagger$
(7) $\mathbf{a}_r = \mathbf{a}_r / \texttt{norm}(\mathbf{a}_r)$
(8) $\mathbf{b}_r = \boldsymbol{\mathcal{Y}}_{(2)}(\mathbf{c}_r \odot \mathbf{a}_r)(\mathbf{c}_r^\top \mathbf{c}_r * \mathbf{a}_r^\top \mathbf{a}_r)^\dagger$
(9) $\mathbf{b}_r = \mathbf{b}_r / \texttt{norm}(\mathbf{b}_r)$
(10) $\mathbf{c}_r = \boldsymbol{\mathcal{Y}}_{(3)}(\mathbf{b}_r \odot \mathbf{a}_r)(\mathbf{b}_r^\top \mathbf{b}_r * \mathbf{a}_r^\top \mathbf{a}_r)^\dagger$
(11) $\boldsymbol{\lambda}_r = \texttt{norm}(\mathbf{c}_r)$
(12) $\mathbf{c}_r = \mathbf{c}_r / \boldsymbol{\lambda}_r$
(13) **until** convergence criterion is reached
(14) $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{B}} - [\![\boldsymbol{\lambda}_{[1:r]}; \mathbf{A}_{[:,1:r]}, \mathbf{B}_{[:,1:r]}, \mathbf{C}_{[:,1:r]}]\!]$ — update residual tensor
(15) **end for**
(16) $\mathbf{A}, \mathbf{B}, \mathbf{C} = [\mathbf{Q}_A\mathbf{A}, \mathbf{Q}_B\mathbf{B}, \mathbf{Q}_C\mathbf{C}]$ — recover factor matrices
(17) re-normalize the factor matrices and update the scaling vector $\boldsymbol{\lambda}$

Return: Normalized factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and the scaling vector $\boldsymbol{\lambda}$.

BLAS and **LAPACK** routines. Thus, all linear algebra operations are threaded and highly optimized on Intel processors. This allows us to optimally exploit the computational benefits of the randomized algorithm. The **rTensor** package provides the following core functions:

- Randomized CP using the alternating least squares method (ALS): **ccp_als()**.
- Randomized CP using the block coordinate descent method (BCD): **ccp_bcd()**.

Our packages builds on the **scikit-tensor** package, which provides routines for basic tensor operations such as folding/unfolding, tensor-matrix and tensor-vector products. The implementation of the CP decomposition follows the **MATLAB Tensor Toolbox** implementation (Bader and Kolda, 2015). For installation instructions visit the GIT repository https://github.com/Benli11/rTensor.

In the following we only briefly describe the **ccp_bcd()** function, since the **ccp_als()** is similar in design. The interface is as follows

```
P = ccp_bcd(T , r, c=True, p=10, q=2, maxiter=500)
```

The first mandatory argument **T** passes the N-way input tensor. Given a **NumPy** array **A**, a dense tensor is created as follows

```
T = dtensor(A)
```

The second mandatory argument **r** sets the target rank, and it is assumed that **c** is smaller then the ambient dimensions of the input tensor. The third argument **c** can be used to chose between the randomized accelerated algorithm **c=True**, and the deterministic algorithm **c=False**. Next, we can set pass values for the two tuning parameters **p** and **q** in order to control the accuracy of the algorithm. The former parameter is used to oversample the basis, and is set by default to **p=10**. The parameter **q** can be used to compute additional power iterations (subspace iterations). By default this parameter is set to **q=2** which shows a good performance in our numerical experiments. The resulting model object, **P**, is itself a list and contains the factor matrices as well as a vector $\boldsymbol{\lambda}$ which contains the weights.

This implementation normalizes the components after each step to achieve better convergence. Further, we use eigenvectors (see above) to initialize the factor matrices. Interestingly, randomly initialized factor matrices have the ability to achieve slightly better approximation errors. However, re-running the algorithms several times with different random seeds can display significant variance in the results. Thus, only the former approach is used for initialization. We note that the randomized algorithm introduces some randomness and slight variations into the CP decompositions as well. However, randomization can also act as an implicit regularization on the CP decomposition (Mahoney, 2011), meaning that the results of the randomized algorithm can be in some cases even 'better' than the results of the corresponding deterministic implementation. Following Bader and Kolda (2015), the convergence criterion is defined as the change in fit. The algorithm therefore stops when the improvement of the fit ρ is less then a predefined threshold, where the fit is computed using

$$\rho = 1 - \frac{\|\boldsymbol{\mathcal{X}}\|_F^2 + \|\hat{\boldsymbol{\mathcal{X}}}\|_F^2 - 2 \cdot \langle \hat{\boldsymbol{\mathcal{X}}}, \boldsymbol{\mathcal{X}} \rangle}{\|\boldsymbol{\mathcal{X}}\|_F^2}.$$

6.7 Numerical Results

In the following the performance of the randomized CP algorithm is evaluated using both random tensors as well as several canonical data sets from fluids and climate. The fluid data consists of vorticity fields from a fluid simulation, and real-world measurements of ocean sea surface temperature spanning two decades. The results demonstrate that the near optimal approximation for massive tensors can be achieved in a fraction of the time using the randomized algorithm. In order to characterize the approximation accuracy the relative error is computed as $\|\boldsymbol{\mathcal{X}} - \hat{\boldsymbol{\mathcal{X}}}\|_F / \|\boldsymbol{\mathcal{X}}\|_F$, where $\hat{\boldsymbol{\mathcal{X}}}$ denotes the approximated tensor. All computations are performed on a workstation running Ubuntu 16 LTS, with the following hardware specifications: 12 Intel Xeon CPUs E5-2620 (2.4GHz), and 32GB DDR3 memory.

6.7.1 Computational Performance

First, the robustness of the randomized CP algorithm is assessed on random low-rank tensors. Specifically, it is of interest to examine the approximation quality in the presence of additive white noise. Figure 6.4 shows the averaged relative errors over 100 runs for varying signal-to-noise ratios (SNR). In the presence of little noise all algorithms converge towards the same relative error. However, at excessive levels of noise (i.e., SNR$<$ 4) the deterministic CP algorithms exhibit small gains in accuracy over the randomized algorithms using $q = 2$ power iterations. Here, both the ALS and BCD algorithm show the same performance. The performance of the randomized algorithm without power iterations ($q = 0$) is, however, poor. This highlights the importance of the power operation for real applications. The oversampling parameter for the randomized algorithms is set here and in the following to $p = 10$. Increasing p can slightly improve the accuracy, but setting $p = 10$ is generally sufficient.

Next, the reconstruction errors and runtimes for tensors of varying dimensions are compared. Figure 6.5 shows the average evaluation results over 100 runs for random low-rank tensors of different dimensions, and for varying target ranks k. The randomized algorithms achieve near optimal approximation accuracy while demonstrating substantial computational

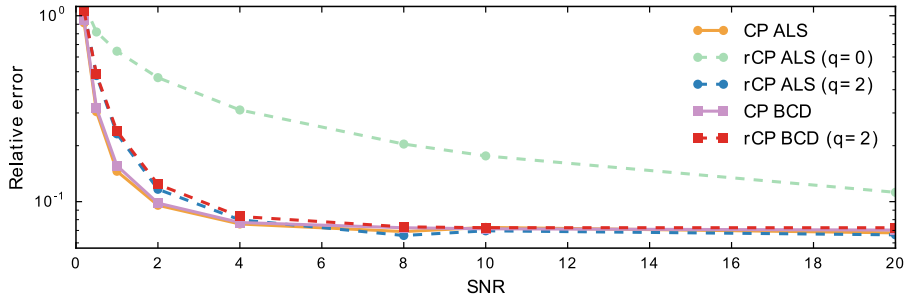
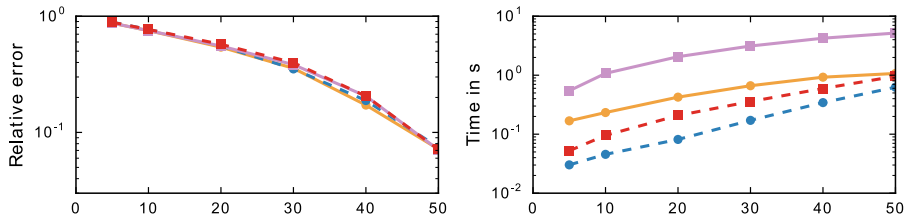


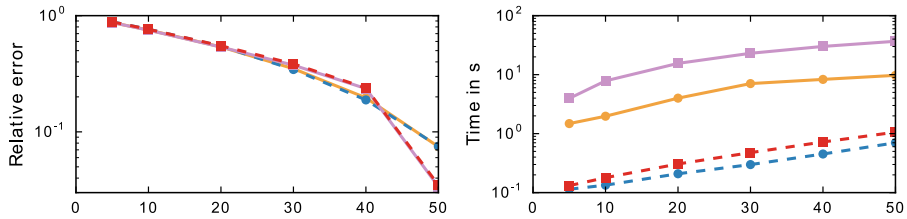
Fig. 6.4 Average relative error, plotted on a log scale, against increasing signal to noise ratio. The analysis is performed on a rank $R = 50$ tensor of dimension $100 \times 100 \times 100$. Power iterations improve the the approximation accuracy considerably.

savings. The computational advantage becomes pronounced with increasing tensor dimensions, as well as with an increasing number of iterations required for convergence. Using random tensors as presented here, all algorithms rapidly converge after about 4 to 6 iterations. However, it is evident that the computational cost per iteration of the randomized algorithm is substantially lower. Thus, the computational savings in real world applications, which require several hundred iterations to converge, can be even more substantial. Overall, the ALS algorithm is computationally more efficient than BCD. The deterministic ALS algorithm is faster than the BCD by nearly one order of magnitude. However, the randomized algorithms exhibit similar computational timings. Interestingly, the BCD relative error decreases sharply by about one order of magnitude when the target rank is achieved, and the tensor rank is much smaller then the ambient dimensions. Similar performance results are achieved for higher order tensors as well. Figure 6.6 shows the computational performance for a 4-way tensor of dimension $100 \times 100 \times 100 \times 100$. Again, the randomized algorithms achieve speedups of 1-2 orders of magnitude, while attaining good approximation errors. Once more, the BCD algorithm achieves better approximation at the true tensor rank.

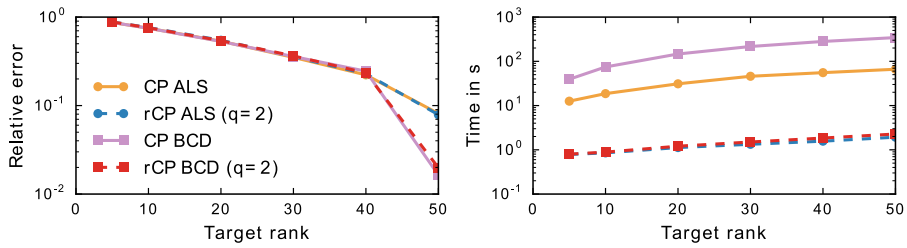
Figure 6.7 shows the computational savings and speedups of a rank $k = 20$ approximation for varying tensor dimensions. In particular the speedups achieved for the block coordinate descent method are substantial.



(a) Tensor of dimension $100 \times 100 \times 100$.



(b) Tensor of dimension $200 \times 200 \times 200$.



(c) Tensor of dimension $400 \times 400 \times 400$.

Fig. 6.5 Random tensor approximation and performance for rank $R = 50$ tensors: rCP methods achieve speedups by 1-2 orders of magnitude and the same accuracy as their deterministic counterpart. Speedups rise sharply with increasing dimensions.

6.7.2 Numerical Examples

In the following, several examples are presented demonstrating the performance of the randomized CP decomposition. The first is a multiscale toy video example, the second is the simulated flow field behind a stationary cylinder, and the third consists of real-world measurement data of global sea surface temperature. Due to the better and more natural interpretability of the block coordinate descent algorithm, only this algorithm is considered in subsequent sections. BCD is in particular advantageous

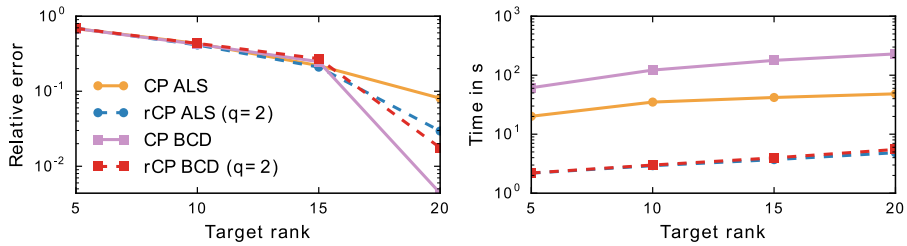


Fig. 6.6 Random tensor approximation and performance for a 4-way rank $R = 20$ tensor of dimension $100 \times 100 \times 100 \times 100$.

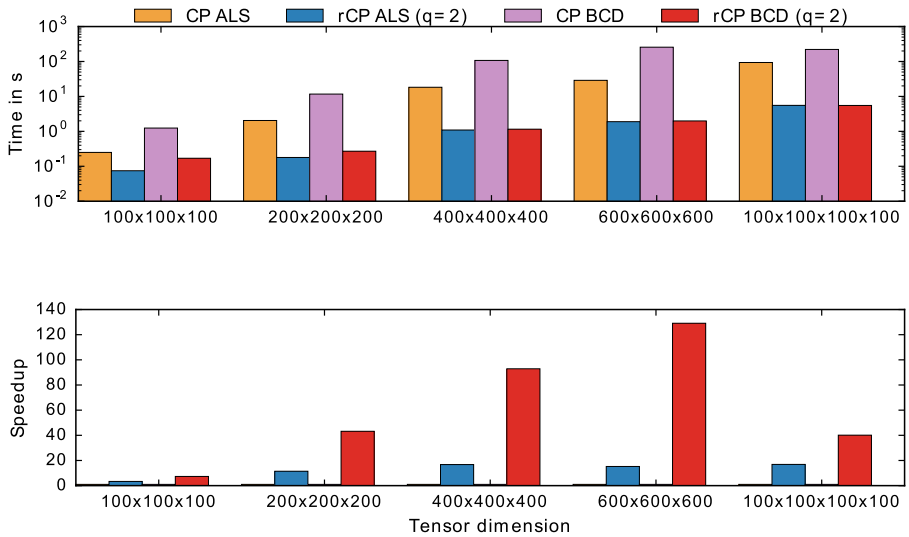


Fig. 6.7 Algorithm runtimes and speedups for target rank $k = 20$ approximation for varying tensor dimensions. The runtime savings increase with the tensor size.

when dealing with data consisting of both spatial modes and temporal dynamics, as presented in the following examples.

Multiscale Toy Video Example

The approximation of the underlying spatial modes and temporal dynamics of a system is a common problem in signal processing. In the following, we consider a toy example presenting multiscale intermittent dynamics in the time direction. Specifically, the data is generated by four Gaussian modes on a two-dimensional spatial grid (200×200), which are undergoing intermittent oscillations in the temporal direction resolved with 215

time steps. Thus, the resulting tensor is of dimension $200 \times 200 \times 215$. Figure 6.8 shows the corresponding modes and the time dynamics.

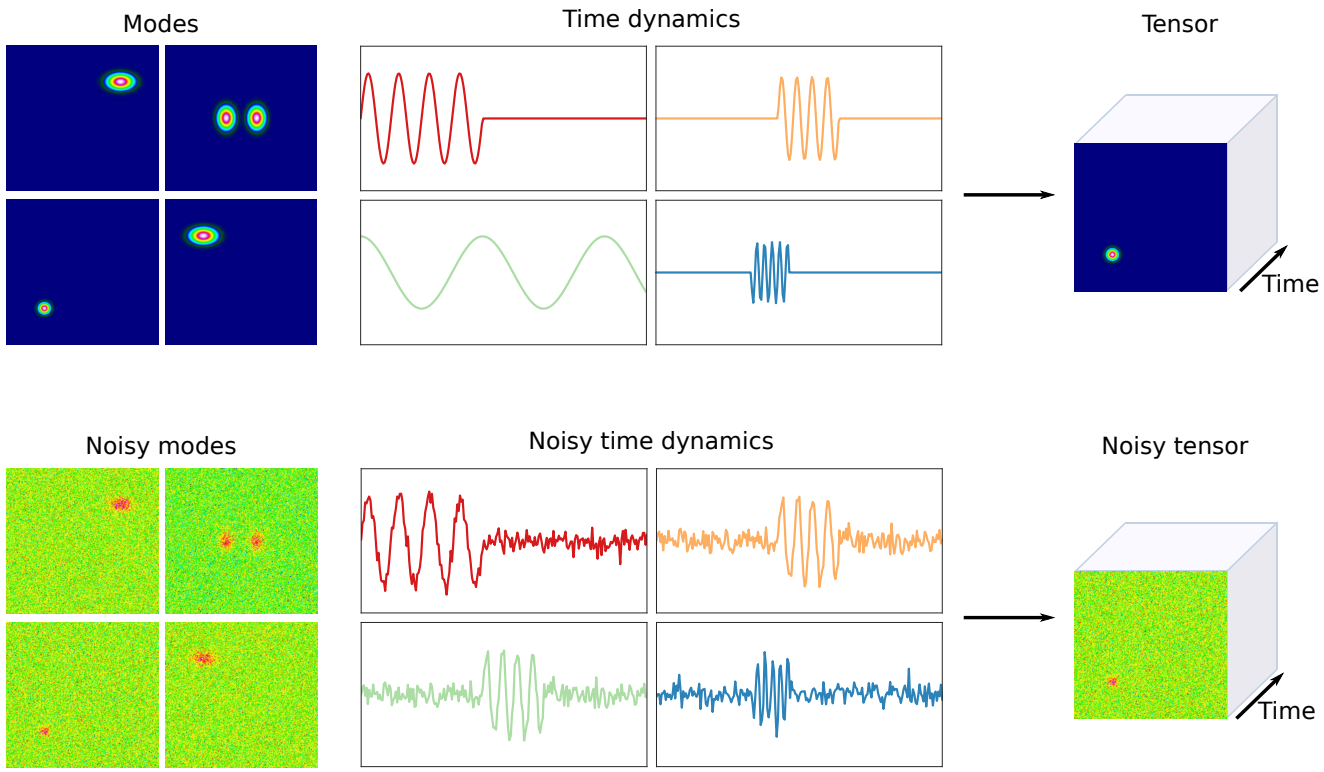


Fig. 6.8 Illustration of the multiscale toy video. The system is governed by four spatial modes experiencing intermittent oscillations in the temporal direction. The bottom subplot shows the noisy signal with a signal-to-noise ratio of 2.

This problem becomes even more challenging when the underlying structure needs to be reconstructed from noisy measurements. In particular, traditional matrix decomposition techniques such as the singular value decomposition (SVD) or standard principal component analysis (PCA) face difficulties approximating the underlying system of such intermittent multiscale dynamics. This is mainly due to the fact that the SVD estimates more coefficients than necessary. Thus, the approximation tends to overfit. The CP decomposition needs to estimate many fewer coefficients, in contrast. This is because the data do not need to be reshaped, which allows for a parsimonious approximation. Comparing the compression ratios between the two methods illustrates the difference. For a real rank $R = 4$ tensor of dimension $100 \times 100 \times 100$, the compression ratios are computed as follows

$$c_{SVD} = \frac{I \cdot J \cdot K}{R \cdot (I \cdot J + K + 1)} = \frac{100^3}{4 \cdot (100^2 + 100 + 1)} \approx 24.75,$$

and

$$c_{CP} = \frac{I \cdot J \cdot K}{R \cdot (I + J + K + 1)} = \frac{100^3}{4 \cdot (100 + 100 + 100 + 1)} \approx 830.56.$$

The SVD requires this data to be reshaped in some dimension. The comparison displays the striking difference between compression ratios. It is evident that the CP decomposition requires computing many fewer coefficients in order to approximate the tensor. This makes the method more robust in the presence of noise. A further advantage is that much less storage is required to approximate the data. This can be of importance if the bandwidth is constrained and only a limited amount of information can be transmitted, in which case the CP decomposition may be advantageous. However, the advantage of the SVD is that noise free data can be approximated with an accuracy as low as machine precision.

Figure 6.9, shows the decomposition results of the noisy (SNR=2) toy video for both the randomized CP decomposition and the SVD. The subplots (a) and (b) show the results of a rank $k = 4$ approximation computed using the rCP algorithm with $q = 2$ power iterations, and a small oversampling parameter $p = 10$. The method faithfully captures

the underlying spatial modes and the time dynamics. For illustration, the subplots (c) and (d) show the decomposition results without additional power iterations. It can clearly be seen that this approach introduces distinct artifacts, and the approximation quality is relatively poor overall. The subplots (e) and (f) correspond to the singular value decomposition. The results show poor performance at separating the different modes. In particular, the spatiotemporal dynamics of modes 2 & 3 are mixed, as well as modes 2 & 4 to a lesser extent. Table 6.1 further quantifies the observed results. The achieved speedup of rCP is substantial, with a speedup factor of about 15. Interestingly, the relative error using the randomized algorithm with $q = 2$ power iterations is slightly better than the deterministic algorithm. This is due to the beneficial properties of randomization, which can act to regularize. However, the reconstruction error without power iterations is large, as is the error resulting from the SVD.

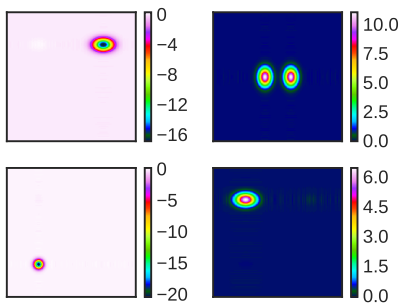
Table 6.1 Summary of the computational results for the noisy toy video.

	Parameters	**Time (s)**	**Speedup**	**Iterations**	**Error**
CP BCD	$k = 4$	2.31	-	9	0.0171
rCP BCD	$k = 4$, $p = 10$, $q = 0$	0.13	17	9	0.494
	$k = 4$, $p = 10$, $q = 1$	0.14	16	10	0.0191
	$k = 4$, $p = 10$, $q = 2$	0.15	15	10	0.0164
SVD	$k = 4$	0.52	-	-	0.137

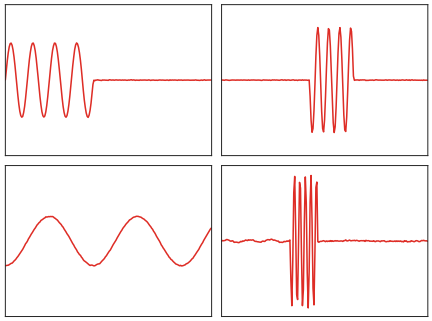
Flow behind a cylinder

Matrix decomposition techniques play an important role in the area of fluid dynamics (Holmes et al., 2012). Extracting the dominant coherent structures from fluid flows helps to better characterize them for modeling and control (Brunton and Noack, 2015). The workhorse algorithm in fluid dynamics and for model reduction is the singular value decomposition. However, fluid simulations generate high-resolution spatiotemporal grids of data which naturally manifest as tensors. In the following we examine the suitability of the CP decomposition for decomposing flow data, and compare the results to those of the SVD.[1]

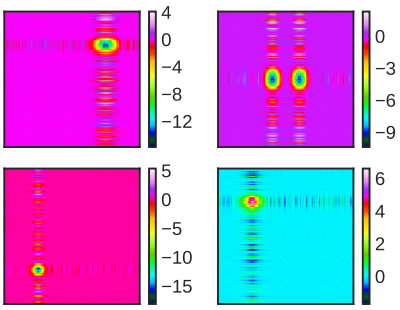
[1] We omit here the comparison with the dynamic mode decomposition, because the results of the SVD and the DMD are similar for this specific example in the following.



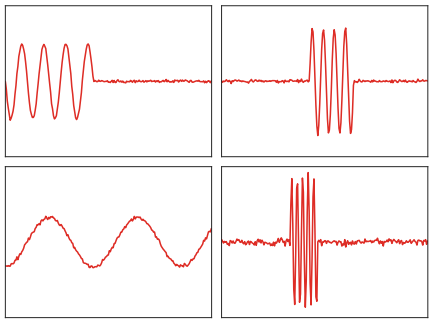
(a) rCP BCD modes, $q = 2$.



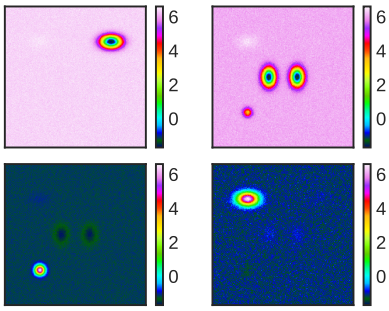
(b) rCP BCD time dynamics, $q = 2$.



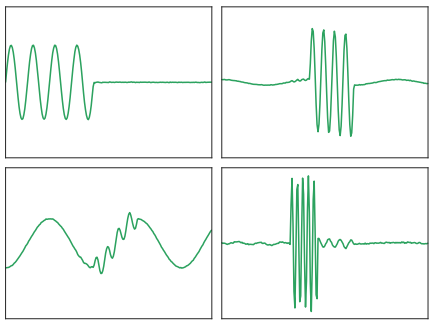
(c) rCP BCD modes, $q = 0$.



(d) rCP BCD time dynamics, $q = 0$.



(e) SVD modes.



(f) SVD time dynamics.

Fig. 6.9 Toy video decomposition results. Randomized CP with $q = 2$ successfully reconstructs the original spatiotemporal dynamics from noise-corrupted data, while SVD and rCP without subspace iterations yield poor reconstruction results.

Here, we revisit the example of a fluid flow behind a cylinder as introduced in Section 3.6, which is a canonical example in fluid dynamics (Noack et al., 2003). Recall, the data are constructed as a time-series of fluid vorticity fields behind a stationary cylinder on an equispaced grid. The data is obtained by solving the two-dimensional Navier-Stokes equations using the immersed boundary projection method in a fast multi-domain solver (Colonius and Taira, 2008; Taira and Colonius, 2007). The corresponding flow tensor is of dimension $199 \times 449 \times 151$, containing 151 snapshots of a 449×199 spatial grid. Further, the flow is characterized by a periodically shedding wake structure at Reynolds number $Re = 100$ and is inherently low-rank in the absence of noise. This can be seen in the normalized spectra for the singular values as shown in Figure 6.10. The characteristic frequencies of flow oscillations occur in pairs, reflecting

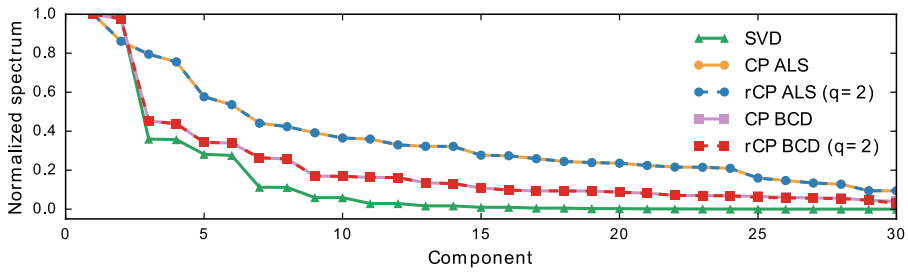


Fig. 6.10 Normalized spectrum. The SVD and (r)CP BCD decompositions successfully capture pairs of characteristic frequencies in the low-rank cylinder flow.

the complex-conjugate pairs of eigenvalues that define sine and cosine temporal dynamics. Accordingly, the normalized singular values and coefficients for both the SVD and the normalized lambda values of the CP decomposition using BCD accurately reflect the true physics of the cylinder wake. However, it can be seen that the singular values capture more energy in fewer modes than the CP decomposition, i.e., the SVD requires fewer singular vectors to approximate the flow. But, because the SVD is based on the flattened spatial dimensions of the flow, the decomposition ignores some of the spatial structure. In contrast, the CP decomposition is able to reveal the underlying structure more accurately.

Figure 6.11 shows both the approximated spatial modes and the temporal dynamics for the randomized CP decomposition and the SVD. The

temporal dynamics of both methods have similar patterns. However, the spatial modes exhibit a distinct structure. The SVD groups modes together according to their variance in the data, while ignoring coordinate structure. Thus, the first four spatial modes revealed by the CP decomposition are condensed into the first two SVD modes. CP spatial modes resemble spatial Fourier transforms of the SVD spatial modes. This is because CP modes are rank-one outer products of vectors in the x and y dimensions which by construction cannot capture the triangular wake interactions seen in the SVD modes.

The CP decomposition represents the spatial structure by single spatial frequencies per mode, which poses difficulties in obtaining a decomposition low in tensor rank. This explains why for a fixed target rank of 30 across all methods, the SVD achieves a substantially lower reconstruction error, seen in Table 6.2. However, the compression ratios for the CP and SVD methods are $c_{CP} \approx 562.17$ and $c_{SVD} \approx 5.02$ assuming a target rank $R = 30$. Thus the tensor representation compresses the data further by nearly two orders of magnitude.

Table 6.2 Summary of the computational results for the noise-free cylinder flow.

	Parameters	**Time (s)**	**Speedup**	**Iterations**	**Error**
CP BCD	$k = 30$	115.55	-	458	0.117
rCP BCD	$k = 30$, $p = 10$, $q = 0$	1.27	91	533	0.122
	$k = 30$, $p = 10$, $q = 1$	1.41	82	517	0.121
	$k = 30$, $p = 10$, $q = 2$	1.56	74	437	0.118
SVD	$k = 30$	0.57	-	-	4.25E-05

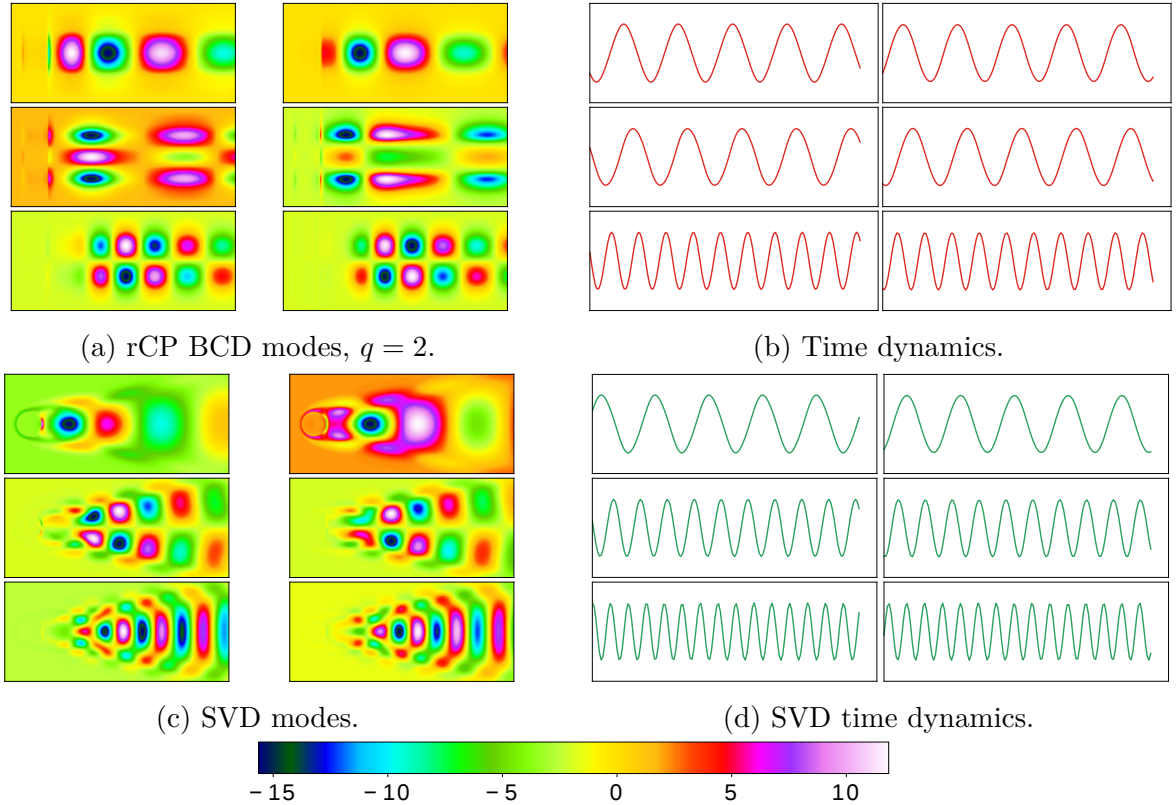
Next, the analysis of the same flow is repeated in the presence of additive white noise. While this is not of concern when dealing with flow simulations, it is realistic when dealing with flows obtained in empirical studies. We choose a signal-to-noise ratio of 2 to demonstrate the robustness of the CP decomposition to noise. Figure 6.12 shows again the corresponding dominant spatial modes and temporal dynamics. Both the SVD and the CP decomposition faithfully capture the temporal dynamics. However, it is apparent that the spatial modes are overfitted, i.e., they contain a large amount of noise. The spatial modes revealed by the CP decomposition show a significantly better approximation. Again, it is

crucial to use power iterations here to achieve a good approximation quality. Table 6.3 lists the results. By inspection, the relative reconstruction error using the SVD is poor compared to the error achieved using the CP decomposition. Here, we have shown the error for a rank $k = 30$ and $k = 6$ approximation. The latter target rank was determined using the optimal hard threshold for singular values (Gavish and Donoho, 2014). However, the suggested target rank is lower then the number of modes which are of interested as seen in 6.10. The CP decomposition overcomes this disadvantage, and is able to approximate the first $k = 30$ modes with only a slight loss of accuracy. Again, it is interesting to see, that the randomized CP decomposition is performing better then the deterministic algorithm in the presence of noise.

Table 6.3 Summary of the computational results for the noise-corrupted cylinder flow.

	Parameters	**Time (s)**	**Speedup**	**Iterations**	**Error**
CP BCD	$k = 30$	64.01	-	239	0.191
rCP BCD	$k = 30$, $p = 10$, $q = 0$	0.99	64	332	0.522
	$k = 30$, $p = 10$, $q = 1$	1.23	52	414	0.189
	$k = 30$, $p = 10$, $q = 2$	1.13	56	370	0.153
SVD	$k = 30$	0.58	-	-	0.655
	$k = 6$	0.58	-	-	0.311

Figure 6.13 further illustrates the approximation quality with an example snapshot from the reconstructed flow in time. The left column corresponds to the noise free scenario. Here the performance of the SVD is nearly perfect. However, the left column containing the noisy example shows the clear advantage of the CP decomposition. Despite the denoising effect of the SVD, the approximated low-rank subspace remains distorted by noise in the ambient space. While not perfect, the CP decomposition allows a more meaningful interpretation of the underlying structure, and thus can be seen as a valuable tool for the analysis of fluid flows in the presence of noise. In addition, the spatiotemporal, multiscale separation performed by the CP decomposition can be beneficial for flow interpretation in practice. From a purely data analytic perspective, the CP modes offer a new and intriguing interpretation of flow structure, hence the trade-off between lower-rank approximation with SVD and multiscale separation with tensors must be weighed carefully.



(a) rCP BCD modes, $q = 2$.

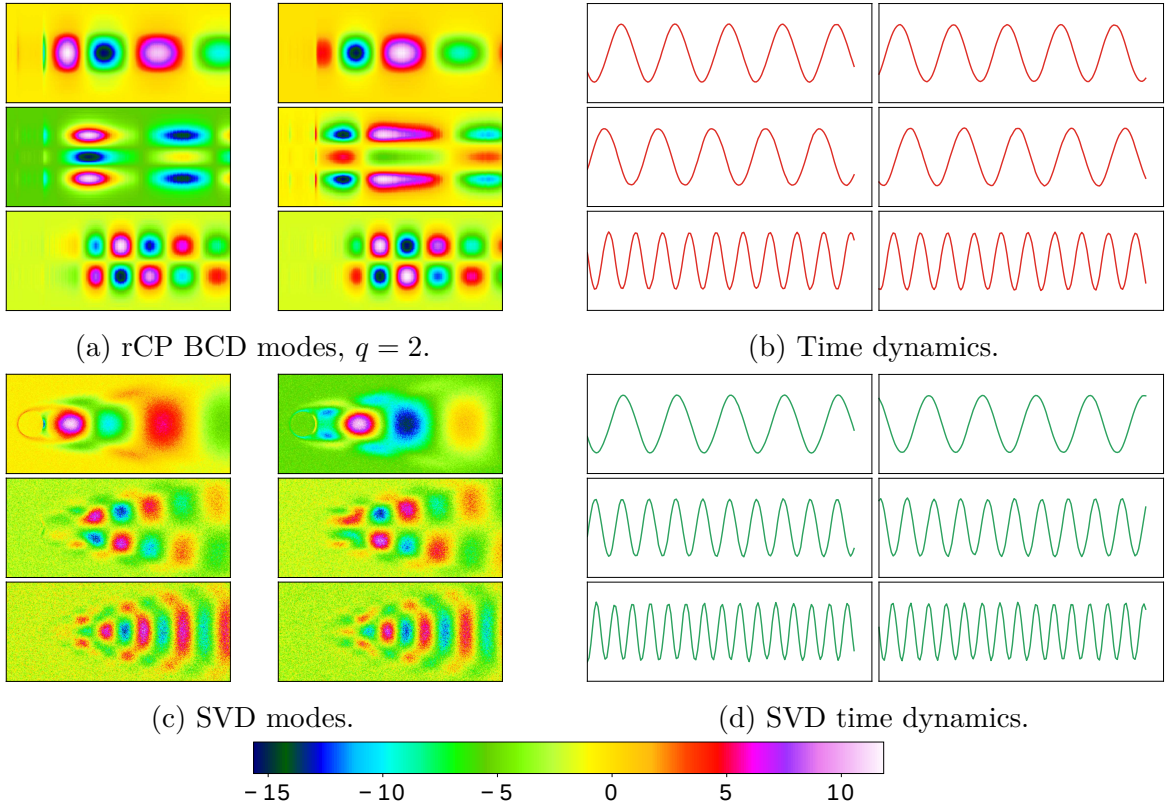
(b) Time dynamics.

(c) SVD modes.

(d) SVD time dynamics.

(e) Colorbar.

Fig. 6.11 Fluid flow decomposition, no noise. Both methods capture the same dominant frequencies from the time dynamics, while randomized CP requires more rank-1 outer products in x and y to represent single frequency spatial dynamics.



(a) rCP BCD modes, $q = 2$.

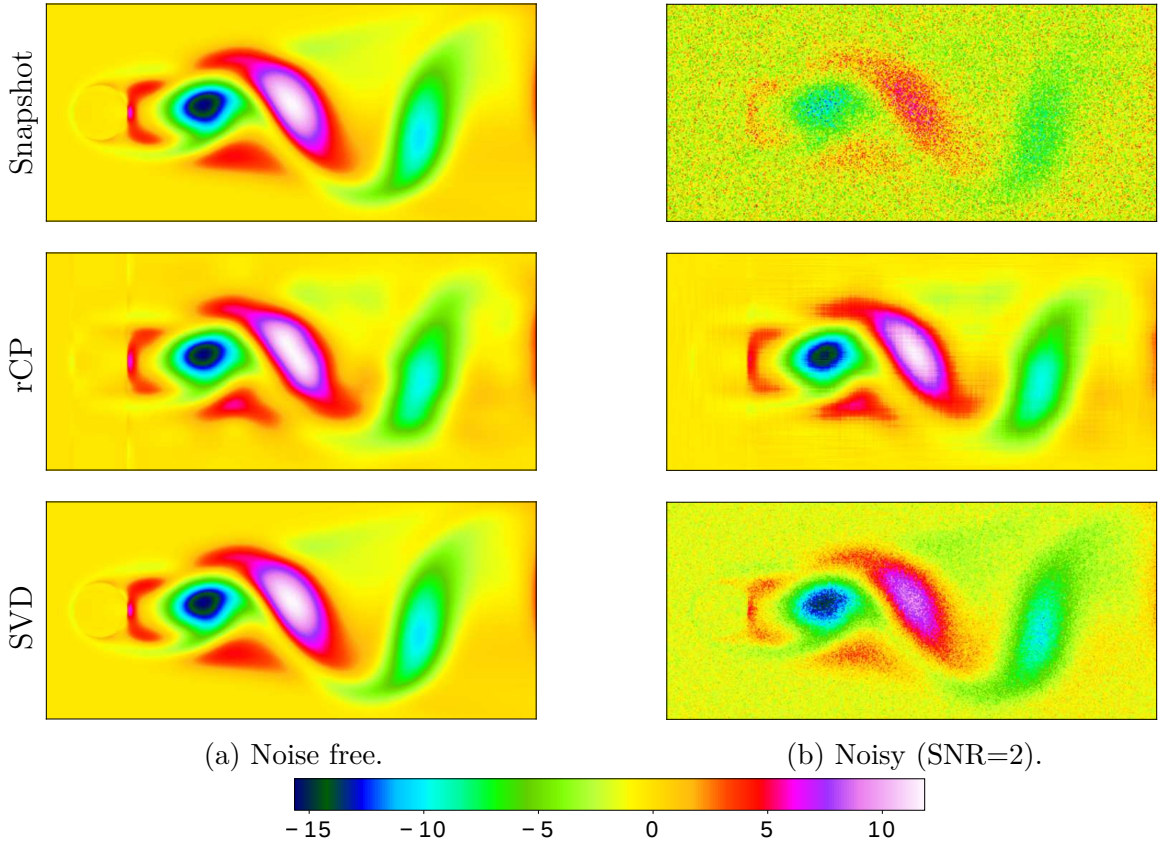
(b) Time dynamics.

(c) SVD modes.

(d) SVD time dynamics.

(e) Colorbar.

Fig. 6.12 Fluid flow decomposition noisy (SNR=2). Randomized CP modes are robust to additive noise and SVD spatial modes are corrupted by noise.



(a) Noise free.

(b) Noisy (SNR=2).

(c) Colorbar.

Fig. 6.13 Fluid snapshots, and randomized CP and SVD approximations of the snapshot are pictured in the first, second and third rows, respectively. The randomized CP better recovers the true signal (top left) from noise-corrupted flow (top right).

Sea Surface Temperature

The last example considers sea surface temperature (SST) data. In particular, we are interested in analyzing the deviations of sea surface temperatures in the equatorial Pacific Ocean, illustrated in Figure 6.14. Periodic departures from the expected sea surface temperatures in this

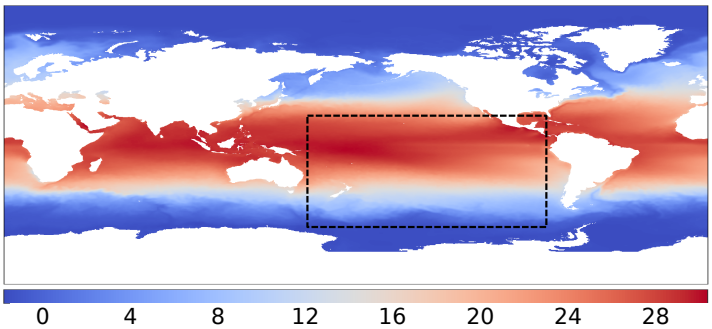


Fig. 6.14 Mean sea surface temperature field. The dashed rectangle indicates the area under consideration.

area are known as the El Niño and La Niña phenomena. Together, they are called the El Niño southern oscillation, describing a band of warmer or cooler ocean water temperatures than usual. These ocean currents are well known for their effects on the weather patterns around the world, and for their influence on winds and precipitation. Specifically, an El Niño event is defined as an anomaly, where the 5-month running means of SST exceed 0.4 °C for 6 months or more (Trenberth, 1997). Data of the weekly sea surface temperatures for the last 26 years are publicly available from the National Oceanic & Atmospheric Administration (NOAA). [2] The data can be expressed as a tensor of dimension $1390 \times 90 \times 125$, where the first index corresponds to time, and the other two index the spatial grid. Interpolated measurements are used for the land masses.

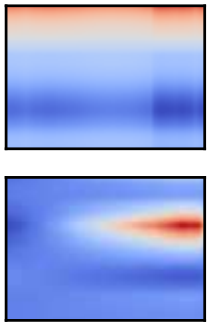
The aim of the following analysis is to isolate the temporal direction of the Sea surface temperature in order to identify El Niño and La Niña events. Therefore, the data are first mean centered in the temporal direction. The results are shown in Figure 6.15 for both the SVD and randomized CP decomposition using the BCD optimization algorithm.

[2]The data set can be obtained at http://www.esrl.noaa.gov/psd/ stored in the NetCDF data format.

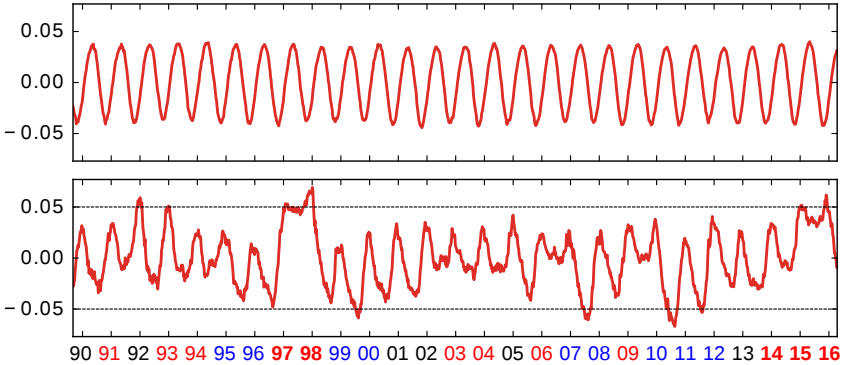
The time dynamics for both methods robustly identify the deviations from the mean ocean temperature over time. Both methods exhibit good agreement with the annual summer to winter mean field dynamics in the first temporal modes. More interestingly, the second temporal mode features the strong El Niño events occurring from 1997-1998 and 2015-2016 as well as the strong La Niña events beginning in 1999, 2007 and 2010. However, while the SVD mode exhibits the whole area of the warm and cold ocean water band, it seems that the second CP mode better characterizes the strongest sea surface temperature anomaly, which is located off the coast of South America. This result is relevant, because the literature distinguishes between several different types of El Niño events (Johnson, 2013). Thus, the CP decomposition is a promising method to reveal more refined insights about this highly complex process. Table 6.2 summarizes the computational results. The randomized CP algorithm achieves a substantial speedup of about 28 using two power iterations, while attaining the same reconstruction error as the deterministic algorithm. Overall, the reconstruction error can decreased by computing a higher rank approximation.

Table 6.4 Summary of the computational results for the sea surface temperature data.

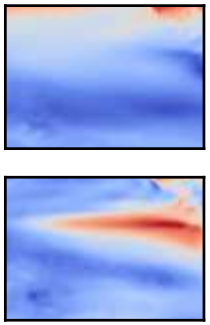
	Parameters	**Time (s)**	**Speedup**	**Iterations**	**Error**
CP BCD	$k = 2$	4.22	-	13	0.461
rCP BCD	$k = 2$, $p = 10$, $q = 0$	0.12	35	13	0.479
	$k = 2$, $p = 10$, $q = 1$	0.14	30	13	0.461
	$k = 2$, $p = 10$, $q = 2$	0.15	28	13	0.461
SVD	$k = 2$	1.93	-	-	0.39



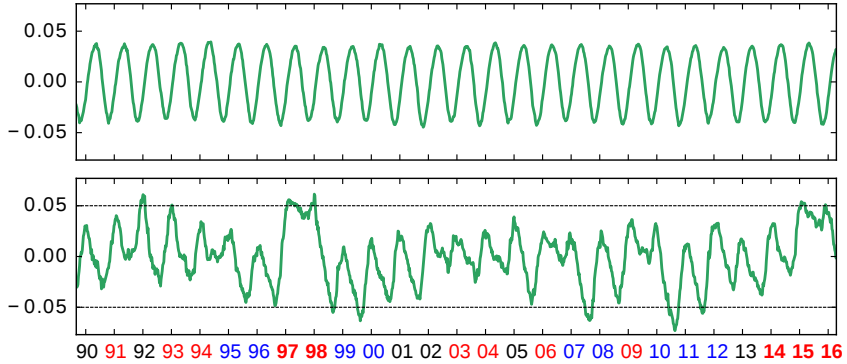
(a) rCP BCD modes



(b) Time dynamics.



(c) SVD modes



(d) Time dynamics.

Fig. 6.15 Sea Surface Temperature decomposition. El Niño (red) and La Niña (blue) events occur when the time dynamics fall above and below the dashed thresholds, respectively, with strong El Niño events indicated in bold red.

6.8 Conclusion

The emergence of massive tensors requires efficient algorithms for obtaining the CP decomposition. We have presented a randomized CP algorithm which substantially reduces the computational burdens involved in obtaining a tensor decomposition. Despite the computational savings, the approximation quality is near-optimal, and in the presence of noisy measurements even better than the deterministic CP algorithm. A key advantage of the randomized algorithm is that modern computational architectures are fully exploited. Thus, the algorithm benefits substantially from multithreading in a multi-core processor. In contrast to proposed algorithms which are based on computational concepts such as distributed computing, our proposed randomized algorithm provides substantial computational speedups even on standard desktop computers. This is achieved by substantially reducing the computational costs per iteration. In practice, real world examples require a larger number of iterations to converge, and thus traditional deterministic algorithms are often not feasible.

Randomized algorithms have established themselves as highly competitive methods for computing traditional matrix decompositions. Thus the generalization of these concepts to tensors are evident. In particular, the concept of oversampling as well as power iterations are crucial in order to achieve a robust tensor decomposition. Our experimental results show that in general a small oversampling parameter and about two power iterations are sufficient. Once a good basis is obtained, the CP decomposition can be obtained in a computationally efficient manner on the compressed tensors using well established optimization strategies. We have considered both the alternating least squares and the block coordinate descent methods. Due to its more natural relationship to the singular value decomposition, we have favored the BCD method throughout our numerical examples. Indeed, the randomized CP decomposition demonstrates outstanding performance on several examples using artificial and real-world data. While the singular value decomposition is able to achieve reconstruction errors as low as machine precision, the CP decomposition is the method of choice in the presence of noise. This is because the

parsimonious approximation is more robust to noise. Further, the CP decomposition reveals interesting structures not captured by the singular value decomposition.

The lean and powerful concepts of randomization make the algorithm useful as a common routine, which may simply replace the deterministic algorithm. The achieved approximation quality is sufficient for many approximations. However, if higher precision is required, the proposed algorithm can be used to efficiently determine the optimal tensor rank and provide good starting values to initialize the factor matrices. Further research will involve the development of randomized algorithms for sparse tensors, and tensors with missing values and non-negative tensor factorizations. A further interesting research direction is a randomized incremental (online) implementation of the CP tensor decomposition. Online tensor subspace learning plays an important role in many signal and video processing applications (Li et al., 2007). Extending the here presented ideas is straight forward, and will allow to substantially ease the computational demands of deterministic incremental tensor algorithms as well as enable real-time video processing.

Chapter 7

Conclusion

"Ce que nous connaissons est peu de chose,
ce que nous ignorons est immense."
— Pierre-Simon Laplace

7.1 Randomness as a Computational Strategy

Randomness is a powerful concept and an effective computational strategy for designing faster algorithms. In particular, employing a degree of randomness as part of the logic of low-rank matrix and tensor decomposition algorithms has been shown to be extremely successful. This new computational paradigm becomes increasingly important in the era of 'big data', where deterministic algorithms become computationally infeasible.

The basic idea of these randomized algorithms is to derive a smaller matrix from a high-dimensional data matrix, which can subsequently be used to efficiently compute a near-optimal low-rank approximation. Specifically, the concept of random projections is used to form a basis which best captures the action of the column space of the high-dimensional input matrix. Then, a smaller matrix is simply constructed by projecting the input data matrix on the low-dimensional subspace spanned by the natural basis. This framework ensures that the small matrix captures the essential spectral information with high probability, and in many applications the small matrix is sufficient to learn from the data.

The key advantage of using randomness as a computational strategy is that the resulting randomized algorithms are robust, and highly reliable, yet simple to implement. Furthermore, the approximation quality can be controlled using the concepts of oversampling and the power scheme.

7.2 Summary of the Contributions

This thesis essentially builds up on the probabilistic framework for low-rank matrix approximations, formulated by Halko et al. (2011b).

Despite the great success of randomized matrix algorithms over the last two decades, the techniques are still widely unfamiliar in areas outside of theoretical computer science, applied mathematics and statistics. In order to make the randomized singular value decomposition and randomized principal component analysis better accessible for researchers in other areas, e.g., social and biological science, we introduced the R software package **rsvd**. Chapter 2 as well as the accompanying paper (Erichson et al., 2016c) aim to give a user-friendly and hands-on introduction to randomized techniques for computing low-rank matrix approximations.

In addition, to broaden the audience, the second fundamental aim of this thesis is to extend the scope of the probabilistic framework to new applications. Chapter 3 demonstrates how the dynamic mode decomposition can benefit from randomness as a computational strategy by embedding the computational steps into the probabilistic framework. The DMD is an emerging dimensionality reduction technique which is extensively used in the area of fluid dynamics. The proposed randomized algorithm for computing the DMD enables to analyze fluid flows which are too big to decompose using the deterministic algorithm, yet the accuracy is better than that of previous proposed probabilistic strategies like the compressed DMD algorithm. Indeed, some of our ideas have started to attract the attention of researchers in the area of fluid dynamics (Bistrian and Navon, 2016; Dawson, 2017; Taira et al., 2017).

While not specifically designed for the task of background modeling, we have demonstrated in Chapter 4 that the DMD is an interesting candidate for background modeling as well. The randomized DMD algorithm allows to process videos at a much higher frame rate than several other techniques

designed for this task. While the achieved detection performance cannot compete with the leading techniques in this area, the randomized DMD might be still a viable building block for more complex real-time vision systems.

Further, in Chapter 5 we have demonstrated that GPU accelerated implementation of randomized routines can substantially increase the computational performance. High-performance computing using GPUs can be a crucial concept if real-time processing is required. The GPU accelerated routines are provided as part of the **scikit-cuda** package.

Finally, we have generalized the probabilistic framework for computing low-rank matrix decompositions to tensor decompositions. The randomized CP algorithm, presented in Chapter 6, substantially reduces the computational demands, while achieving a near-optimal decomposition. In addition to the presented numerical results, Theorem 1 characterizes the average-case behavior of the proposed randomized tensor algorithm. Moreover, our corresponding software package **rTensor** allows to use the randomized CP decomposition straight away in Python.

7.3 Perspectives

7.3.1 Short-Term Perspectives

The lean and powerful concept of randomization is of interest for a wide variety of applications. Future work has the potential to apply the ideas to several linear dimensionality reduction techniques like discriminant analysis, canonical correlation analysis and multi dimensional scaling. However, more generally, we have already shown that the small matrix can be used to learn from the data using techniques such as alternating least squares or block coordinate descent. Our current work shows that the probabilistic framework can be used in a similar manner for the expectation maximization algorithm. This allows us to impose structure like sparsity or non-negativity.

Moreover, the here presented algorithms for computing the DMD and the CP tensor decomposition can be implemented as incremental (online)

algorithms as well. This is, in particular, of interest in several signal and video processing applications, which often deal with streaming like data.

Javed et al. (2015a) show that the background model accuracy can be considerably improved by using multi-features. Thus, it is an interesting research direction to investigate how multi-features can be efficiently integrated into the DMD and CP framework.

7.3.2 Long-Term Perspectives

Randomized techniques will play a major role in future applications across science. This is because the size of the data problems is growing more rapidly than the available computing power is - meaning solutions will have to be found by algorithmic efficiencies, rather than brute force. Another important aspect is that randomized algorithms are energy efficient, i.e., the reduced run-time requires significant less computational resources than deterministic algorithms. This aspect becomes crucial if applications run on devices like robots, drones or autonomous underwater vehicles. Furthermore, randomness can not only be used to reduce the computational load, but also to reduce communications costs, i.e., instead of transmitting the full data it might be sufficient to just transmit the smaller (compressed) data matrix.

References

H. Abdi and L. J. Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2:433–459, 2010.

D. Achlioptas and F. McSherry. Fast computation of low-rank matrix approximations. *Journal of the ACM*, 54:9, 2007.

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. ***LAPACK*** *Users' Guide*. SIAM, 3rd edition, 1999.

M. Babaee, D. T. Dinh, and G. Rigoll. A deep convolutional neural network for background subtraction. *arXiv preprint arXiv:1702.01731*, 2017.

B. W. Bader and T. G. Kolda. MATLAB tensor toolbox version 2.6, 2015. URL http://www.sandia.gov/~tgkolda/TensorToolbox/.

R. G. Baraniuk. Compressive sensing. *IEEE Signal Processing Magazine*, 24(4):118–120, 2007.

O. Barnich and M. Van Droogenbroeck. Vibe: A universal background subtraction algorithm for video sequences. *IEEE Transactions on Image processing*, 20(6):1709–1724, 2011.

D. Bates and M. Maechler. ***Matrix****: Sparse and dense matrix classes and methods*, 2016. URL https://CRAN.R-project.org/package=Matrix. R package version 1.2-6.

Y. Benezeth, P.-M. Jodoin, B. Emile, H. Laurent, and C. Rosenberger. Comparative study of background subtraction algorithms. *Journal of Electronic Imaging*, 19(3):–, 2010. doi: 10.1117/1.3456695.

S. Bianco, G. Ciocca, and R. Schettini. How far can you get by combining change detection algorithms? *arXiv preprint arXiv:1505.02921*, 2015.

D. Bistrian and I. Navon. Randomized dynamic mode decomposition for non-intrusive reduced order modelling. *International Journal for Numerical Methods in Engineering*, pages 1–22, 2016. doi: 10.1002/nme.5499.

L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

T. Bouwmans. Recent advanced statistical background modeling for foreground detection: A systematic survey. *Recent Patents on Computer Science*, 4(3):147–176, 2011.

T. Bouwmans. Traditional and recent approaches in background modeling for foreground detection: An overview. *Computer Science Review*, 11-12: 31–66, 2014. doi: 10.1016/j.cosrev.2014.04.001.

T. Bouwmans and E.-H. Zahzah. Robust PCA via principal component pursuit: A review for a comparative evaluation in video surveillance. *Computer Vision and Image Understanding*, 122:22–34, 2014. doi: 10.1016/j.cviu.2013.11.009.

T. Bouwmans, N. S. Aybat, and E. Zahzah. *Handbook of Robust Low-Rank and Sparse Matrix Decomposition: Applications in Image and Video Processing*. CRC Press, 2016a. ISBN 9781498724623.

T. Bouwmans, A. Sobral, S. Javed, S. K. Jung, and E.-H. Zahzah. Decomposition into low-rank plus additive matrices for background/-foreground separation: A review for a comparative evaluation with a large-scale dataset. *Computer Science Review*, pages 1–71, 2016b. doi: 10.1016/j.cosrev.2016.11.001.

M. Braham and M. Van Droogenbroeck. Deep background subtraction with scene-specific convolutional neural networks. In *Systems, Signals and Image Processing (IWSSIP), 2016 International Conference on*, pages 1–4. IEEE, 2016.

R. Bro and C. A. Andersson. Improving the speed of multiway algorithms: Part II: Compression. *Chemometrics and Intelligent Laboratory Systems*, 42:105–113, 1998.

B. W. Brunton, L. A. Johnson, J. G. Ojemann, and J. N. Kutz. Extracting spatial–temporal coherent patterns in large-scale neural recordings using dynamic mode decomposition. *Journal of Neuroscience Methods*, 258: 1–15, 2016. doi: 10.1016/j.jneumeth.2015.10.010.

S. L. Brunton and B. R. Noack. Closed-loop turbulence control: Progress and challenges. *Applied Mechanics Reviews*, 67:1–60, 2015.

S. L. Brunton, J. L. Proctor, J. H. Tu, and J. N. Kutz. Compressed sensing and dynamic mode decomposition. *Journal of Computational Dynamics*, 2(2):165–191, 2015. ISSN 2158-2491. doi: 10.3934/jcd.2015002.

Y. Cai and S. See. *GPU Computing and Applications*. Springer, 2015.

E. J. Candès and M. B. Wakin. An introduction to compressive sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, 2008. doi: 10.1109/MSP.2007.914731.

E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Journal of the ACM*, 58(3):1–37, 2011. doi: 10.1145/1970392.1970395.

J. D. Carroll and J.-J. Chang. Analysis of individual differences in multidimensional scaling via an N-way generalization of 'Eckart-Young' decomposition. *Psychometrika*, 35:283–319, 1970.

S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

A. Cichocki and A. H. Phan. Fast local algorithms for large scale non-negative matrix and tensor factorizations. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 92:708–721, 2009.

A. Cichocki, R. Zdunek, A. Phan, and S. Amari. *Nonnegative matrix and tensor factorizations: Applications to exploratory multi-way data analysis and blind source separation.* John Wiley & Sons, 2009.

B. A. Cipra. The best of the 20th century: Editors name top 10 algorithms. *SIAM news*, 33(4):1–2, 2000.

T. Colonius and K. Taira. A fast immersed boundary method using a nullspace approach and multi-domain far-field boundary conditions. *Computer Methods in Applied Mechanics and Engineering*, 197:2131–2146, 2008.

P. Comon, X. Luciani, and A. De Almeida. Tensor decompositions, alternating least squares and other tales. *Journal of Chemometrics*, 23: 393–405, 2009.

J. P. Cunningham and Z. Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *Journal of Machine Learning Research*, 16:2859–2900, 2015.

L. Dalcin, R. Paz, and M. Storti. MPI for Python. *Journal of Parallel and Distributed Computing*, 65(9):1108–1115, 2005. doi: 10.1016/j.jpdc.2005.03.010.

M. A. Davenport and J. Romberg. An overview of low-rank matrix recovery from incomplete observations. *IEEE Journal of Selected Topics in Signal Processing*, 10(4):608–622, 2016.

S. T. Dawson. *Reduced-order modeling of fluids systems, with applications in unsteady aerodynamics.* PhD thesis, Princeton University, 2017.

L. De Lathauwer, B. De Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21:1253–1278, 2000.

J. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006. doi: 10.1109/TIT.2006.871582.

P. Drineas and M. W. Mahoney. A randomized algorithm for a tensor-based generalization of the singular value decomposition. *Linear Algebra and its Applications*, 420:553–571, 2007.

P. Drineas and M. W. Mahoney. RandNLA: Randomized numerical linear algebra. *Communications of the ACM*, 59:80–90, 2016.

C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218, 1936.

N. B. Erichson. ***rsvd**: Randomized singular value decomposition*, 2015. URL https://CRAN.R-project.org/package=rsvd. R package version 0.6.

N. B. Erichson and C. Donovan. Randomized low-rank dynamic mode decomposition for motion detection. *Computer Vision and Image Understanding*, 146:40–50, 2016. doi: 10.1016/j.cviu.2016.02.005.

N. B. Erichson, S. L. Brunton, and J. N. Kutz. Compressed dynamic mode decomposition for background modeling. *Journal of Real-Time Image Processing*, pages 1–14, 2016a. doi: 10.1007/s11554-016-0655-2.

N. B. Erichson, K. Manohar, S. L. Brunton, and J. N. Kutz. Randomized CP tensor decomposition. *Submitted*, pages 1–30, 2016b.

N. B. Erichson, S. Voronin, S. L. Brunton, and J. N. Kutz. Randomized matrix decompositions using R. *arXiv preprint arXiv:1608.02148*, 2016c.

N. B. Erichson, S. L. Brunton, and J. N. Kutz. Randomized dynamic mode decomposition. *arXiv preprint arXiv:1702.02912*, 2017.

J. R. Fienup. Invariant error metrics for image reconstruction. *Applied Optics*, 36:8352–8357, 1997.

A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. *J. ACM*, 51:1025–1041, 2004.

M. Gavish and D. L. Donoho. The optimal hard threshold for singular values is $4/\sqrt{3}$. *IEEE Transactions on Information Theory*, 60(8): 5040–5053, 2014.

A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:643–660, 2001.

L. E. Givon, T. Unterthiner, N. B. Erichson, D. W. Chiang, E. Larson, L. Pfister, S. Dieleman, G. R. Lee, S. van der Walt, T. M. Moldovan, F. Bastien, X. Shi, J. Schlüter, B. Thomas, C. Capdevila, A. Rubinsteyn, M. M. Forbes, J. Frelinger, T. Klein, B. Merry, L. Pastewka, S. Taylor, F. Wang, and Y. Zhou. scikit-cuda 0.5.1: a Python interface to GPU-powered libraries, 2015. URL http://dx.doi.org/10.5281/zenodo.40565. http://dx.doi.org/10.5281/zenodo.40565.

G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM, Series B: Numerical Analysis*, (2):205–224, 1965.

G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14:403–420, 1970.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3 edition, 1996.

N. Goyette, P.-M. Jodoin, F. Porikli, J. Konrad, and P. Ishwar. Changedetection.net: A new change detection benchmark dataset. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 1–8. IEEE, 2012.

J. Grosek and J. N. Kutz. Dynamic mode decomposition for real-time background/foreground separation in video. *arXiv preprint arXiv:1404.7592*, 2014.

M. Gu. Subspace iteration randomization and singular value problems. *SIAM Journal on Scientific Computing*, 37(3):1139–1173, 2015. doi: 10.1137/130938700.

N. Halko, P.-G. Martinsson, Y. Shkolnisky, and M. Tygert. An algorithm for the principal component analysis of large data sets. *SIAM Journal on Scientific Computing*, 33:2580–2594, 2011a.

N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011b. doi: 10.1137/090771806.

R. A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. Technical Report 16, Working Papers in Phonetics, UCLA, 1970.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer-Verlag, 2nd edition, 2009.

F. L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematical Physics*, 6:164–189, 1927.

P. J. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley. *Turbulence, coherent structures, dynamical systems and symmetry.* Monographs in Mechanics. Cambridge University Press, 2 edition, 2012.

J. R. Humphrey, D. K. Price, K. E. Spagnoli, A. L. Paolini, and E. J. Kelmelis. CULA: Hybrid GPU accelerated linear algebra routines, 2010.

S. T. IBM Reseach Division. ***libskylark**: Sketching-based distributed matrix computations for machine learning*, 2014. URL https://xdata-skylark.github.io/libskylark/. C++ package version 0.2.

A. J. Izenman. *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning.* Springer-Verlag, 2008.

S. Javed, S. H. Oh, A. Sobral, T. Bouwmans, and S. K. Jung. OR-PCA with MRF for robust foreground detection in highly dynamic backgrounds. In *Asian Conference on Computer Vision*, pages 284–299. Springer, 2014.

S. Javed, S. H. Oh, T. Bouwmans, and S. K. Jung. Robust background subtraction to global illumination changes via multiple features-based online robust principal components analysis with markov random field. *Journal of Electronic Imaging*, 24(4):1–16, 2015a. doi: 10.1117/1.JEI.24.4.043011.

S. Javed, A. Sobral, T. Bouwmans, and S. K. Jung. OR-PCA with dynamic feature selection for robust background subtraction. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 86–91. ACM, 2015b.

H. Ji and Y. Li. Gpu accelerated randomized singular value decomposition and its application in image compression. *Update*, 4:5, 2014.

N. C. Johnson. How many ENSO flavors can we distinguish? *Journal of Climate*, 26:4816–4827, 2013.

I. Jolliffe. *Principal Component Analysis.* Springer Series in Statistics. Springer-Verlag, 2nd edition, 2002.

E. Jones, T. Oliphant, and P. Peterson. SciPy: Open source scientific tools for Python, 2001. URL https://www.scipy.org/.

M. R. Jovanović, P. J. Schmid, and J. W. Nichols. Sparsity-promoting dynamic mode decomposition. *Physics of Fluids*, 26(2):1–22, 2014.

K. Kang, W. Ouyang, H. Li, and X. Wang. Object detection from video tubelets with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 817–825, 2016.

J. Kim, Y. He, and H. Park. Algorithms for nonnegative matrix and tensor factorizations: A unified view based on block coordinate descent framework. *Journal of Global Optimization*, 58:285–319, 2014.

M. Kirby and L. Sirovich. Application of the Karhunen-Loeve procedure for the characterization of human faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:103–108, 1990.

T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51:455–500, 2009.

B. Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17:315–318, 1931.

A. Korobeynikov and R. M. Larsen. ***svd**: Interfaces to various state-of-art SVD and eigensolvers*, 2016. URL https://CRAN.R-project.org/package=svd. R package version 0.4.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

J. N. Kutz, X. Fu, S. L. Brunton, and N. B. Erichson. Multi-resolution dynamic mode decomposition for foreground/background separation and object tracking. In *IEEE International Conference on Computer Vision Workshop (ICCVW)*, pages 921–929, 2015. doi: 10.1109/ICCVW.2015.122.

J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor. *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*. SIAM, 2016a.

J. N. Kutz, X. Fu, and S. L. Brunton. Multiresolution dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 15(2): 713–735, 2016b.

R. M. Larsen. Lanczos bidiagonalization with partial reorthogonalization. *DAIMI Report Series*, 27:1–101, 1998.

R. Lehoucq, D. Sorensen, and C. Yang. ***ARPACK** Users' Guide*. SIAM, 1998.

N. Li, S. Kindermann, and C. Navasca. Some convergence results on the regularized alternating least-squares method for tensor decomposition. *Linear Algebra and its Applications*, 438:796–812, 2013.

X. Li, W. Hu, Z. Zhang, X. Zhang, and G. Luo. Robust visual tracking based on incremental tensor subspace learning. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

E. Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 581–588. ACM, 2013.

Z. Lin, M. Chen, and Y. Ma. The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. *arXiv preprint arXiv:1009.5055*, 2011.

A. Liutkus. ***Randomized Singular Value Decomposition***, 2014. URL http://uk.mathworks.com/matlabcentral/fileexchange/47835-randomized-singular-value-decomposition. MATLAB package version 1.0.

M. W. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends in Machine Learning*, 3:123–224, 2011.

S. G. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

P.-G. Martinsson. Randomized methods for matrix computations and analysis of high dimensional data. *arXiv preprint arXiv:1607.01649*, 2016.

P.-G. Martinsson, V. Rokhlin, and M. Tygert. A randomized algorithm for the decomposition of matrices. *Applied and Computational Harmonic Analysis*, 30:47–68, 2011.

P.-G. Martinsson, G. Quintana-Orti, and N. Heavner. randutv: A blocked randomized algorithm for computing a rank-revealing utv factorization. *arXiv preprint arXiv:1703.00998*, 2017.

O. Mersmann, C. Beleites, R. Hurling, and A. Friedman. ***microbenchmark****: Accurate timing functions*, 2015. URL http://CRAN.R-project.org/package=microbenchmark. R package version 1.4-2.1.

I. Mezić and A. Banaszuk. Comparison of systems with complex behavior. *Physica D: Nonlinear Phenomena*, 197:101 – 133, 2004.

J. Monod. *Chance and Necessity: An Essay on the Natural Philosophy of Modern Biology.* Alfred A. Knopf, 1971.

R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective.* MIT Press, 2012.

J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA. *Queue*, 6(2):40–53, 2008. doi: 10.1145/1365490.1365500.

B. R. Noack, K. Afanasiev, M. Morzynski, G. Tadmor, and F. Thiele. A hierarchy of low-dimensional models for the transient and post-transient cylinder wake. *J. Fluid Mech.*, 497:335–363, 2003.

D. Okanohara. ***redsvd**: Randomized SVD*, 2011. URL https://code.google.com/archive/p/redsvd/. C package version 0.2.

J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008. ISSN 0018-9219. doi: 10.1109/JPROC.2008.917757.

S. I. Park, S. P. Ponce, J. Huang, Y. Cao, and F. Quek. Low-cost, high-speed computer vision using nvidia's cuda architecture. In *Applied Imagery Pattern Recognition Workshop, 2008. AIPR'08. 37th IEEE*, pages 1–7. IEEE, 2008.

K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, 2:559–572, 1901.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. **Scikit-learn**: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

V. Pham, P. Vo, V. T. Hung, et al. GPU implementation of extended gaussian mixture model for background subtraction. In *IEEE International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future*, pages 1–4, 2010.

A. Phan and A. Cichocki. PARAFAC algorithms for large-scale problems. *Neurocomputing*, 74:1970–1984, 2011.

J. Proctor and P. Echhoff. Discovering dynamic patterns from infectious disease data using dynamic mode decomposition. *International Health*, 7:139–145, 2015.

J. L. Proctor, S. L. Brunton, and J. N. Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1): 142–161, 2016.

Y. Qiu, J. Mei, G. Guennebaud, and J. Niesen. ***RSpectra**: Solvers for large scale eigenvalue and SVD problems*, 2016. URL https://CRAN.R-project.org/package=RSpectra. R package version 0.12-0.

V. Rokhlin, A. Szlam, and M. Tygert. A randomized algorithm for principal component analysis. *SIAM Journal on Matrix Analysis and Applications*, 31:1100–1124, 2009.

C. Rowley, I. Mezić, S. Bagheri, P. Schlatter, and D. Henningson. *Spectral analysis of nonlinear flows. Journal of Fluid Mechanics*, 641:115–127, 2009.

R. Rubinstein, M. Zibulevsky, and M. Elad. Efficient implementation of the K-SVD algorithm using batch orthogonal matching pursuit. *CS Technion*, 40(8):1–15, 2008.

J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming, Portable Documents*. Addison-Wesley, 2010.

T. Sarlos. Improved approximation algorithms for large matrices via random projections. In *Foundations of Computer Science. 47th Annual IEEE Symposium on*, pages 143–152, 2006.

S. Scardapane and D. Wang. Randomness in neural networks: An overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(2):1–41, 2017. doi: 10.1002/widm.1200.

P. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010. doi: 10.1017/S0022112010001217.

S. C. Sen-Ching and C. Kamath. Robust techniques for background subtraction in urban traffic video. In *Electronic Imaging 2004*, pages 881–892. International Society for Optics and Photonics, 2004.

A. Shimada, D. Arita, and R.-i. Taniguchi. Dynamic control of adaptive mixture-of-gaussians background model. In *Video and Signal Based Surveillance, 2006. AVSS'06. IEEE International Conference on*, pages 5–5. IEEE, 2006.

X. Shu, F. Porikli, and N. Ahuja. Robust orthonormal subspace learning: Efficient recovery of corrupted low-rank matrices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3874–3881, 2014.

N. Sidiropoulos, E.Papalexakis, and C. Faloutsos. Parallel randomly compressed cubes: A scalable distributed architecture for big tensor decomposition. *IEEE Signal Processing Magazine*, 31:57, 2014.

A. Smilde, R. Bro, and P. Geladi. *Multi-Way Analysis with Applications in the Chemical Sciences*. John Wiley & Sons, 2005.

A. Sobral and A. Vacavant. A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos. *Computer Vision and Image Understanding*, 122:4–21, 2014. doi: 10.1016/j.cviu.2013.12.005.

P.-L. St-Charles, G.-A. Bilodeau, and R. Bergevin. A self-adjusting approach to change detection based on background word consensus. In *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*, pages 990–997. IEEE, 2015.

C. Stauffer and W. Grimson. *Adaptive background mixture models for real-time tracking*. Proceedings IEEE Conf. on Computer Vision and Pattern Recognition, 1999.

G. W. Stewart. On the early history of the singular value decomposition. *SIAM Review*, 35:551–566, 1993.

A. Szlam, Y. Kluger, and M. Tygert. An implementation of a randomized algorithm for principal component analysis. *arXiv preprint arXiv:1412.3510*, 2014.

K. Taira and T. Colonius. The immersed boundary method: A projection approach. *Journal of Computational Physics*, 225:2118–2137, 2007.

K. Taira, S. L. Brunton, S. Dawson, C. W. Rowley, T. Colonius, B. J. McKeon, O. T. Schmidt, S. Gordeyev, V. Theofilis, and L. S. Ukeiley. Modal analysis of fluid flows: An overview. *arXiv preprint arXiv:1702.01453*, 2017.

S. Tomov, R. Nath, H. Ltaief, and J. Dongarra. Dense linear algebra solvers for multicore with gpu accelerators. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE, 2010.

L. N. Trefethen and D. Bau III. *Numerical Linear Algebra*. SIAM, 1997.

K. E. Trenberth. The definition of El niño. *Bulletin of the American Meteorological Society*, 78:2771, 1997.

J. A. Tropp and A. C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666, 2007.

J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher. Randomized single-view algorithms for low-rank matrix approximation. *arXiv preprint arXiv:1609.00048*, 2016.

C. E. Tsourakakis. MACH: Fast randomized tensor decompositions. In *Proc. 2010 SIAM Int. Conf. Data Mining*, pages 689–700, 2010.

J. Tu, C. Rowley, D. Luchtenberg, S. Brunton, and J. N. Kutz. On dynamic mode decomposition: Theory and applications. *Journal of Computational Dynamics*, 1:391–421, 2014.

M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. In *Proceedings on Computer Vision and Pattern Recognition*, pages 586–591. IEEE, 1991.

A. Uschmajew. Local convergence of the alternating least squares algorithm for canonical tensor approximation. *SIAM Journal on Matrix Analysis and Applications*, 33:639–652, 2012.

A. Vacavant, T. Chateau, A. Wilhelm, and L. Lequievre. A benchmark dataset for outdoor foreground/background extraction. In *Computer Vision–ACCV 2012 Workshops*, pages 291–300. Springer, 2013.

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S-PLUS*. Statistics and Computing. Springer-Verlag, 2002.

N. Vervliet and L. D. Lathauwer. A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors. *IEEE Journal of Selected Topics in Signal Processing*, 10:284–295, 2016.

N. Vervliet, O. Debals, L. Sorber, and L. D. Lathauwer. Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis. *IEEE Signal Processing Magazine*, 31:71–79, 2014.

V. Volkov and J. W. Demmel. Benchmarking gpus to tune dense linear algebra. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–11. IEEE, 2008.

S. Voronin and P.-G. Martinsson. RSVDPACK: Subroutines for computing partial singular value decompositions via randomized sampling on single core, multi core, and GPU architectures. *arXiv preprint arXiv:1502.05366*, 2015.

G. Wang, Y. Xiong, J. Yun, and J. R. Cavallaro. Accelerating computer vision algorithms using opencl framework on the mobile gpu-a case study. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 2629–2633. IEEE, 2013.

L. Wang and M. T. Chu. On the global convergence of the alternating least squares method for rank-one approximation to generic tensors. *SIAM Journal on Matrix Analysis and Applications*, 35:1058–1072, 2014.

R. Wang, F. Bunyak, G. Seetharaman, and K. Palaniappan. Static and moving object detection using flux tensor with split Gaussian models. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 420–424. IEEE, 2014a.

Y. Wang, P.-M. Jodoin, F. Porikli, J. Konrad, Y. Benezeth, and P. Ishwar. CDnet 2014: An expanded change detection benchmark dataset. In *IEEE Workshop on Computer Vision and Pattern Recognition*, pages 393–400. IEEE, 2014b.

Y. Wang, Z. Luo, and P.-M. Jodoin. Interactive deep learning method for segmenting moving objects. *Pattern Recognition Letters*, pages 1–10, 2016a. doi: 10.1016/j.patrec.2016.09.014.

Y. Wang, Z. Luo, and P.-M. Jodoin. Interactive deep learning method for segmenting moving objects. *Pattern Recognition Letters*, 2016b.

D. S. Watkins. *Fundamentals of Matrix Computations.* John Wiley & Sons, 2 edition, 2002.

W. H. Wen-Mei. *GPU computing gems emerald edition.* Elsevier, 2011.

H. Wickham. ***ggplot2**: Elegant Graphics for Data Analysis.* Springer-Verlag, 2009. URL http://had.co.nz/ggplot2/book.

R. Witten and E. Candes. Randomized algorithms for low-rank matrix factorizations: sharp performance bounds. *Algorithmica*, 72:264–281, 2015.

D. P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1-2):1–157, 2014. doi: 10.1561/0400000060.

F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert. A fast randomized algorithm for the approximation of matrices. *Journal of Applied and Computational Harmonic Analysis*, 25:335–366, 2008.

J. Wright, A. Ganesh, S. Rao, Y. Peng, and Y. Ma. Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization. In *Advances in Neural Information Processing Systems*, pages 2080–2088, 2009.

Y. Xu and W. Yin. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on Imaging Sciences*, 6(3): 1758–1789, 2013.

Y. Xu, J. Dong, B. Zhang, and D. Xu. Background modeling methods in video analysis: A review and comparative evaluation. *CAAI Transactions on Intelligence Technology*, 1(1):43–60, 2016. doi: 10.1016/j.trit. 2016.03.005.

C. Zhang, H. Tabkhi, and G. Schirner. A gpu-based algorithm-specific optimization for high-performance background subtraction. In *Parallel Processing (ICPP), 2014 43rd International Conference on*, pages 182–191. IEEE, 2014.

G. Zhou, A. Cichocki, and S. Xie. Decomposition of big tensors with low multilinear rank. *arXiv preprint arXiv:1412.1885*, 2014.

T. Zhou and D. Tao. Godec: Randomized low-rank & sparse matrix decomposition in noisy case. In *International Conference on Machine Learning*, pages 1–8. ICML, 2011.

X. Zhou, C. Yang, and W. Yu. Moving object detection by detecting contiguous outliers in the low-rank representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(3):597–610, 2013.

Z. Zivkovic. Improved adaptive gaussian mixture model for background subtraction. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pages 28–31. IEEE, 2004.

Appendix A

Proof of Theorem 1

In the following, we sketch a proof for Theorem 1, which yields an upper bound for the approximate basis for the range of a tensor. To assess the quality of the basis matrices $\{\mathbf{Q}_n\}_{n=1}^N$, we first show that the problem can be expressed as a sum of subproblems. First, let the residual error be defined as

$$\|\boldsymbol{\mathcal{E}}\|_F = \|\boldsymbol{\mathcal{X}} - \hat{\boldsymbol{\mathcal{X}}}\| = \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_1 \mathbf{Q}_1\mathbf{Q}_1^\intercal \times_2 \cdots \times_N \mathbf{Q}_N\mathbf{Q}_N^\intercal\|_F. \tag{A.1}$$

Note that the Frobenius norm of a tensor and its matricized forms are equivalent. Upon defining the orthogonal projector as $\mathbf{P}_n \equiv \mathbf{Q}_n\mathbf{Q}_n^\intercal$, we can reformulate Equation (A.1) more compactly as

$$\|\boldsymbol{\mathcal{E}}\|_F = \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_1 \mathbf{P}_1 \times_2 \cdots \times_N \mathbf{P}_N\|_F. \tag{A.2}$$

Proof. Assuming, that $\mathbf{P}_n$ yields an exact projection onto the column space of the matrix $\mathbf{Q}_n$, we need to show first that the error can be expressed as a sum of the errors of the n projections

$$\|\boldsymbol{\mathcal{E}}\|_F = \sum_{n=1}^{N} \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_n \mathbf{P}_n\|_F = \sum_{n=1}^{N} \|\boldsymbol{\mathcal{X}} \times_n (\mathbf{I} - \mathbf{P}_n)\|_F, \tag{A.3}$$

where $\mathbf{I}$ denotes the identity matrix. Following Drineas and Mahoney (2007), let us add and subtract the term $\boldsymbol{\mathcal{X}} \times_N \mathbf{P}_N$ in Equation (A.2) so that we obtain

$$\begin{aligned}
\|\boldsymbol{\mathcal{E}}\|_F &= \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_N \mathbf{P}_N + \boldsymbol{\mathcal{X}} \times_N \mathbf{P}_N - \boldsymbol{\mathcal{X}} \times_1 \mathbf{P}_1 \times_1 \cdots \times_N \mathbf{P}_N\|_F && \text{(A.4)}\\
&\leq \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_N \mathbf{P}_N\|_F + \|\boldsymbol{\mathcal{X}} \times_N \mathbf{P}_N - \boldsymbol{\mathcal{X}} \times_1 \mathbf{P}_1 \times_1 \cdots \times_N \mathbf{P}_N\|_F && \text{(A.5)}\\
&= \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_N \mathbf{P}_N\|_F + \|(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_1 \mathbf{P}_1 \times_1 \cdots \times_{N-1} \mathbf{P}_{N-1}) \times_N \mathbf{P}_N\|_F && \text{(A.6)}\\
&\leq \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_N \mathbf{P}_N\|_F + \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_1 \mathbf{P}_1 \times_1 \cdots \times_{N-1} \mathbf{P}_{N-1}\|_F. && \text{(A.7)}
\end{aligned}$$

The bound (A.5) then follows from the triangular inequality for a norm. Next, the common term $\mathbf{P}_N$ is factored out in Equation (A.6). Then, the bound (A.7) follows from the properties of orthogonal projectors. This is because the $range(\boldsymbol{\mathcal{X}} \times_1 \mathbf{P}_1 \times_1 \cdots \times_{N-1} \mathbf{P}_{N-1}) \subset range(\boldsymbol{\mathcal{X}} \times_1 \mathbf{P}_1 \times_1 \cdots \times_N \mathbf{P}_N)$, and then it holds that $\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_1 \mathbf{P}_1 \times_1 \cdots \times_N \mathbf{P}_N\|_F \leq \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_1 \mathbf{P}_1 \times_1 \cdots \times_{N-1} \mathbf{P}_{N-1}\|_F$. See Proposition 8.5 by Halko et al. (2011b) for a proof using matrices. Subsequently the residual error $\boldsymbol{\mathcal{E}}_{N-1}$ can be bounded

$$\|\boldsymbol{\mathcal{E}}_{N-1}\| \leq \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_{N-1} \mathbf{P}_{N-1}\|_F + \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_1 \mathbf{P}_1 \times_1 \cdots \times_{N-2} \mathbf{P}_{N-2})\|_F. \quad \text{(A.8)}$$

From this inequality, Equation (A.3) follows. We take the expectation of Equation (A.3)

$$\mathsf{E}\|\boldsymbol{\mathcal{E}}\|_F = \mathsf{E}\left[\sum_{n=1}^{N} \|\boldsymbol{\mathcal{X}} \times_n (\mathbf{I} - \mathbf{P}_n)\|_F\right]. \quad \text{(A.9)}$$

Then, recall that Theorem 10.5 formulated by Halko et al. (2011b) states the following expected approximation error (formulated here using tensor notation)

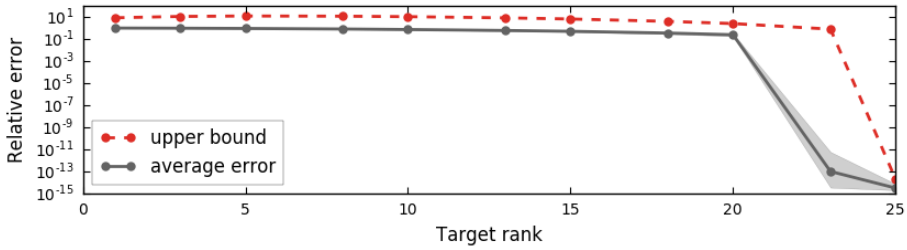
$$\mathsf{E}\|\boldsymbol{\mathcal{X}} \times_n (\mathbf{I} - \mathbf{P}_n)\|_F \leq \sqrt{1 + \frac{k}{p-1}} \cdot \sqrt{\sum_{j>k} \sigma_j^2}, \quad \text{(A.10)}$$

assuming that the sample matrix in Equation (6.4) is constructed using a standard Gaussian matrix $\boldsymbol{\Omega}$. Here σ_j denotes the singular values of the matricized tensor $\boldsymbol{\mathcal{X}}_{(n)}$ greater then the chosen target rank k. Combining Equations (A.9) and (A.10) then yields

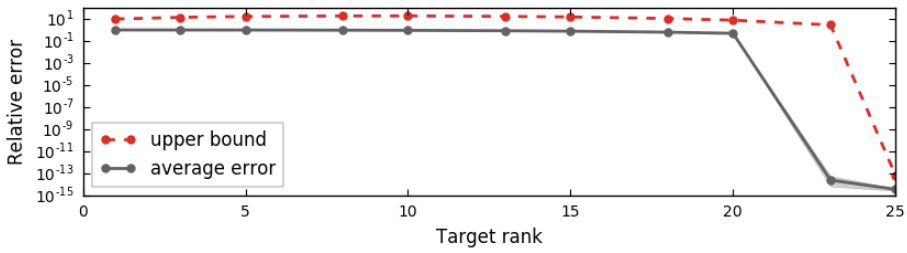
$$\mathsf{E}\|\boldsymbol{\mathcal{E}}\|_F \leq \sqrt{1 + \frac{k}{p-1}} \cdot \sqrt{\sum_{n=1}^{N} \sum_{j>k} \sigma_{nj}^2},$$

where the double sum in the right term sums the eigenvalues over the N matricized tensors. □

Figure A.1 and A.2 show a simulation over 100 runs to evaluate the performance of the theoretical upper bound.



(a) Tensor of dimension $50 \times 50 \times 50$.



(b) Tensor of dimension $50 \times 50 \times 50 \times 50$.

Fig. A.1 Given both a third and fourth order random low-rank $R = 25$ tensor, and assuming a fixed oversampling parameter $p = 2$, the performance of the theoretical upper bound for varying target ranks is bounding the average error faithfully.

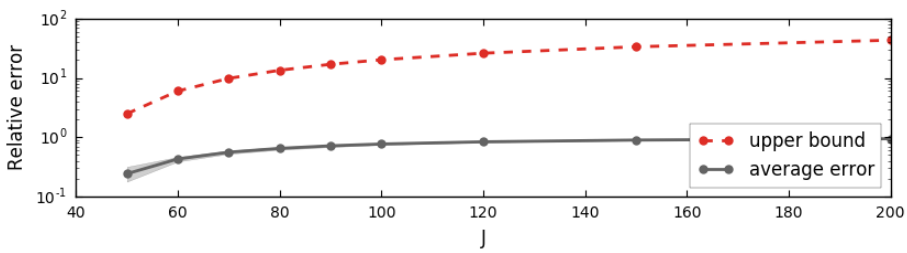


Fig. A.2 Given a low-rank $J/2$ tensor of dimension $J \times J \times J$, and assuming an oversampling parameter $p = 2$ and a fixed target rank $k = 20$, the performance of the theoretical upper bound is slightly overcautious with an increasing ratio between the intrinsic and target rank.