# Dimension reduction techniques for multi-dimensional numerical integrations based on Fourier-cosine series expansion

MSc Final Thesis

# Dimension reduction techniques for multi-dimensional numerical integrations based on Fourier-cosine series expansion

MSc Final Thesis

Zhimin Cheng

Master of Science Thesis in Applied Mathematics

| | |
|---|---|
| Student ID: | 5247381 |
| Project Duration: | 2022.02.07 - 2022.10.21 |

Thesis committee:

| | |
|---|---|
| Dr. F. Fang | TU Delft, supervisor |
| Dr. X.Shen | FF Quant Advisory, supervisor |
| Prof. Dr. ir. C.Vuik | TU Delft, Responsible Full Professor |
| Prof. Dr. A. Papapantoleon | TU Delft, Full Professor |

# Contents

# Abstract

A wide range of practical problems involve computing multi-dimensional integrations. However, in most cases, it is hard to find analytical solutions to these multi-dimensional integrations. Their numerical solutions always suffer from the 'curse of dimension', which means the computational complexity grows exponentially with respect to the dimension.

There is an existing approach that approximates multivariate functions by a tensor of truncated multi-dimensional Fourier series coefficients, and uses the Stochastic Gradient Descent method to solve the lower-rank CPD model, which is used to reduce the computational complexity of the coefficient tensor. In contrast to this work, this thesis project extends its application to solve multi-dimensional integrations, utilizing the Fourier-cosine series expansion to represent the integrand. This project also replaces the SGD method with the Conjugate Gradient method, which improves the function matching accuracy significantly and also has great integration accuracy. The computational cost is reduced regardingly as well.

This thesis also tests an expectation operator related to the COS method, which can be used to compute the expectation of functions of several random variables with much less computational complexity. This method filters out insignificant Fourier-cosine basis functions of the marginal distribution functions, and uses the selected 'principal' basis functions to compute Fourier-cosine coefficients of the joint density function by the high-dimensional COS method. The test results show that more than half of the Fourier-cosine series terms can be dropped per dimension while the expectation accuracy is kept, and the correlation does not influence the expectation accuracy significantly for target functions of normally distributed random variables.

# PREFACE

When at the end of this thesis project, it still feels like the beginning was just yesterday. It could be the reason that I really concentrated on this project and learned a lot from it, thanks to the positive research environment provided by FF Quant Advisory.

I'm very thankful that Dr. F. Fang gave me this opportunity to do my final thesis project under her supervision. She helped me to build up the way of thinking for numerical analysis, and encouraged me to try out my own thoughts with an open mind. And this is how the CPD-CG method came up. Her passion for research also motivated me a lot, and she was always willing to answer my questions in detail and give me feedback on my updates. I really appreciate the positive change she has brought to me.

I also want to thank other people who accompanied me during this meaningful time. A special thanks to Marnix, I wouldn't improve my programming understanding so much without your help, and we can always have nice discussions. Also for Gijs, honestly, I was influenced by your hard-working so I tried to work harder, and you always helped me patiently, I really appreciated it. And thanks to Xiaoyu, who gave me many valuable suggestions for improving my academic thinking. Lastly, I'd like to express my love for my friends, Jingya, Yueran, and my lovely family, you have always been cheering me up in difficult times and celebrated my good results, which gave me the power to move on to the destination.

*Zhimin Cheng*
*Delft, October 2022*

# Abbreviations

CPD : Canonical Polyadic Decomposition

ALS: Alternative Least Square

GD: Gradient Descent

SGD: Stochastic Gradient Descent

CG: Conjugate Gradient

DCTs: Discrete Cosine Transforms

COS: short for replacing the probability density function by its Fourier-cosine series expansion

CC: Clenshaw-Curtis

MC: Monte Carlo

MSE: Mean Square Error

PPF: Percent Point Function

SPD: Symmetric Positive Definite

# 1 Introduction

This chapter aims to give the background and the main motivation for this research. We start with a general dilemma of computational complexity for high-dimensional integrations, followed by several well-known numerical integration methods and some potential dimension-reduction solutions, including the specific approach that this research focuses on. Lastly, the contributions of this thesis are summarized in a concise conclusion.

## 1.1. Problem Description

Numerical integration methods are often utilized when analytically we fail to derive a closed-form solution or the analytical expression is so sophisticated that a straightforward implementation of it does not outperform the numerical solution. The topic of numerical integration has attracted great interest from mathematicians who then created an abundance of methods over a hundred years since the name 'numerical integration' first appeared in 1915 [1].

Note that the numerical integration is needed to solve the integrations not only in one dimension, but also in many cases, over more than one variable. Problems like the valuation of financial derivatives under certain model assumptions, the simulation of physics phenomenons, the evaluation of health big data with many features, etc, often require numerically solving high-dimensional integrals. The computation can be extremely complex and can inevitably lead to the so-called "curse of dimension". That is, the cost to compute the integral numerically with a prescribed accuracy grows exponentially w.r.t. the number of dimensions, which not only makes the calculation time unbearably long but can also lead to memory overflow.

There have been a lot of efforts in literature endeavored to improve the performance of numerical integration, but those methods usually suit one-dimensional situations better. When solving a multi-dimensional integration problem, we often need methods or strategies for avoiding the curse of dimension.

What we are aiming for in this project is to find efficient methods to alleviate the curse of dimension. Some existing solutions in literature designed for this purpose will be briefly discussed in Chapter 2.

## 1.2. OVERVIEW OF NUMERICAL METHODS FOR INTEGRATION

The classical but still actively studied methods for numerical integration are quadrature rules. The simplest ones of this type make use of an equidistant grid for discretizing the integration interval, such as the rectangle rule and the trapezoidal rule [2]. The trapezoidal rule is exact for linear functions; since it utilizes first-order polynomials, or in other words, the error convergence is of the second order, i.e. $ah^2 + O(h^3)$, whereby $h$ is the gird size and $a$ is a constant. Another simple method is Simpson's rule, which is exact for the integration of the 3rd-order polynomial. Next to the simple methods, there are more advanced quadrature methods, such as Gaussian quadrature formulas and Clenshaw-Curtis quadrature rule[3]. These methods no longer rely on equidistant grid points and usually have exponential error convergence when the integrand is infinitely differentiable.

Nowadays, efficient implementation of these methods is available in different programming languages. For example, NumPy and Scipy from Python have built-in functions to calculate integrations, which provide fast and satisfactory results for one-dimensional integrations. However, it's tougher to compute integrations in higher dimensions because the number of grid points, and thus the computational complexity involved, grows exponentially with respect to the number of dimensions. Quadrature rules applied in higher dimensions result in multiplications of a hyper-cube with a number of vectors, which consumes a considerable amount of memory and time. Hence, researchers are exploring means to break or alleviate the dimensionality bottleneck. There are effective dimension-reduction techniques developed in recent literature, especially the low-order tensor based methods, such as Canonical Polyadic Decomposition (CPD) method. These are detailed later in Section 2.2 and are the main research target of this thesis project.

A second type of numerical method often used in practice for computing integrations is the Monte Carlo (MC) simulation method, which is in particular suitable for multi-dimensional integrations. The computational complexity only grows linearly w.r.t. the number of dimensions, and thus, the MC method can be conveniently applied to many fields. The main drawback of the MC method is in the calculation accuracy, for which improvements have been made in literature, e.g. by importance sampling method [4]. What's more, machine learning methods are also engaged in improving the performance of MC simulation, such as boosted decision trees, GANs [5], and neural network based methods.

## 1.3. RESEARCH OBJECTIVES

This report aims to reduce the computational complexity of numerical integrations, especially by exploring the methods based on Fourier-cosine series expansion. The main idea is to replace the integrand by its Fourier-cosine series expansion and then exchange the order of summation and integration, which transforms the original integration into a summation of weighted sine functions. Hence, this approach in essence transforms the original integration problem into the problem of computing the Fourier-cosine series coefficients of the integrand. And in multidimensional cases, the number of coefficients grows exponentially w.r.t. the number of dimensions. Hence, it is crucial to find an efficient way to solve the coefficients without too much computational cost, which is the main goal of this thesis project and we have combined a lower-rank tensor decomposition technique with a supervised machine learning method to achieve this goal.

In order to be in a position to compare the computed value with an exact value, we test on integrals that can be evaluated analytically. Although this may give an impression of undesirable simplicity, more practical examples are chosen to illustrate the applicability of this method for more general integrand functions. For the proof-of-concept, we use the Gaussian probability density function as the integrand since its analytical solution of the integration is one exactly.

## 1.4. CONTRIBUTIONS

This thesis targets at developing new but efficient solutions for solving high-dimensional numerical integrations. As mentioned earlier, the main idea is to transform the integration problem into the problem of solving the Fourier-cosine coefficients of the integrand. We contribute two dimension-reduction methods along this line of research.

The key idea of the generic solution method we developed in this thesis is to decompose the hyper-cube of the coefficients into lower-rank factor matrices by tensor decomposition technique and employ a supervised machine learning method to solve these lower-rank matrices directly. Near the end of this thesis research, we also explored a method for a special but very often-seen case of multi-dimensional integration, namely the expectation operator. It is the combination of the high-dimensional COS method and our innovative idea that one can select principal Fourier terms and leave out the "non-important" cosine basis functions.

To be more precise, our contributions have two folds:

1. A generic dimension-reduction method: We reproduced the methods in [6] and then improved their methods by means of finding the lower-rank representation of Fourier coefficients by the Conjugate Gradient (CG) method, instead of the originally adopted Stochastic Gradient Descent (SGD) solution. This improvement has been tested to outperform the original methods significantly.

2. A dimension-reduction method for an often-seen special case - expectation operator: Our innovative approach is to filter out non-important cosine basis functions in the high-dimensional COS method. It has been tested to greatly reduce computational complexity.

## 1.5. THESIS OUTLINE

This chapter provides a general introduction to this thesis. A literature review related to the keywords of this thesis is given in Chapter 2, such as the Fourier series, Tensor approaches, and appropriate Machine Learning methods. Chapter 3 gives basic definitions of tensors and a few useful tensor operations. Moreover, a tensor-based decomposition method, the Canonical Polyadic Decomposition (CPD) method, is introduced. We use a 2D toy example to illustrate how CPD is done based on Fourier-cosine series expansion.

In chapter 4, we apply a machine learning method to compute the lower-rank tensors resulted from CPD. More insights into this method are also discussed. A further attempt of employing the machine learning method to find the lower-rank tenors is illustrated in Chapter 5, following a method developed in [6], we further replace the SGD method with CG as a new solver. The model efficiency is tested in both 2D and 3D, which shows superior performance to the SGD method. Chapter 6 describes our innovative idea of founding the ’principal’ cosine terms based on 1D Fourier expansion to reduce computational complexity. Finally, Chapter 7 gives a conclusion of the project and a summary of the two methods we have developed and tested, as well as future research work.

# 2

# Ingredients of Our Solution

Multi-dimensional numerical integration methods suffer from the curse of dimension, as we mentioned in the previous chapter. This chapter provides an overview of three ingredients involved in our solution to this problem. The first one is Fourier-cosine series expansion, which transforms the integration problem into the computation of Fourier-cosine coefficients as we mentioned before. As the second ingredient, some tensor-based dimension-reduction techniques which are popular in literature are introduced. One of such techniques is employed to decompose the multi-dimensional Fourier coefficients expressed as a high-order tensor, by a few lower-rank factor matrices. At last, we present the third ingredient of a supervised machine learning method, which we utilize to solve the lower-rank factor matrices efficiently.

## 2.1. Fourier-cosine series expansion

Roughly speaking, the celebrated Fourier series expansion constructs any smooth function by cosine and sine basis functions, and the amplitude of the basis functions is determined by Fourier coefficients.

Fourier series is usually used to represent periodic functions, and those square-integrable functions even have a unique representation of Fourier form [7]. However, non-periodic functions with a compact support can also be resembled by Fourier series expansion. In that case, we actually reconstruct a periodic extension of the original function [6]. And our work focuses on the Fourier-cosine series expansion, which is obtained by applying Fourier series expansion on the even extension of the target function.

For smooth functions, the error of approximation converges exponentially w.r.t. the number of leading terms. And it is worth noting that in the context of calculating the expectation of the function of a few random variables, we need to truncate the integration range in the first place, to be able to apply Fourier series expansion on the integrand, which introduces the integration range truncation error. Moreover, there is a balance between the wideness of the integration truncation range and the number of needed terms of the series expansion: the larger the domain, the more terms in the series expansion are needed to reach a certain accuracy.

There are different means for solving coefficients of cosine terms. A straightforward approach is to apply advanced numerical integration methods such as Clenshaw-Curtis quadrature rule, and it is also possible to speed up the calculations by utilizing the algorithm of discrete cosine transforms (DCTs). For example, authors in [8] apply the Fourier-cosine series expansions for pricing several options in 2D. They use the 2D-COS method to compute Fourier-cosine coefficients for the conditional density function of which the characteristic function is known, and 2D Fourier coefficients of the payoff functions can be approximated by using DCTs if no exact representation is available. [9] extends the use of cosine series expansions in three-dimensional pricing problems with the 3D-COS method and DCTs to compute coefficients. In our solution, we resemble the coefficients via tensor-based Canonical Polyadic Decomposition of which the composition of lower-rank tensors is found via a supervised machine learning method. Details are given in the following sections.

## 2.2. TENSOR BASED DIMENSION-REDUCTION TECHNIQUES

Tensors are general forms of multidimensional arrays. We are familiar with their low-dimensional representations: matrices and vectors. Calculations involving higher-order tensors always raise troubles because of high dimensionality. In literature, there are various tensor decomposition techniques designed to tackle this problem. Two typical decomposition methods that are popular in literature are: Canonical Polyadic Decomposition (CPD) [10] and Higher-Order Singular Value Decomposition (HOSVD). The CPD reconstructs a tensor as a sum of rank-one tensors, and HOSVD is actually a principal component approach applied in high dimensions [11], with Tucker method [12] being a typical example of the latter.

In the last two decades, interests in tensor-based calculations not only rose in numerical linear algebra, but also in the field of signal processing (speech [13], communications [14], radar [15], biomedical [16]), data mining (handwritten digit [17], text representation [18], streams and graphs [19] [20]), Psychometrics, Chemometrics, and so on. At the same time, a number of improvements to the CPD and Tucker method have been made to cope with the particular needs of these different application fields.

Though the idea of CPD was first invented in 1927 [21], it did not become popular until 1970 when Carroll and Chang [22] brought it to the psychometrics community, in the form of the summation of rank-one tensors. They also proposed the Alternating Least Squares (ALS) [22] method to solve the CPD model as a linear problem. The standard ALS is generally powerful and outperforms many existing algorithms. However, the improvement of the ALS method is still ongoing. For instance, instead of alternatively updating factor matrices, damped Gauss-Newton and a variant named PMF3 can optimize all factor matrices simultaneously [23]. Recently, CPD has also been applied for solving large-scale, sparse tensors [24], not only via ALS, but also other adapted methods, like the generalized Rayleigh-Newton iteration method [25].

Tucker decomposition follows the idea of higher-order Principal Component Analysis. Unlike the CPD model, Tucker has a core tensor with matrices multiplied along each direction of the hypercube. And another essential difference is the way that elements interact with each other. For instance, a three-order tensor has Tucker factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, then each column of $\mathbf{A}$ interacts with every column of $\mathbf{B}$ and $\mathbf{C}$, with weights of these interactions recorded in the core tensor, while CPD only allows interactions between corresponding columns of **A**, **B** and **C**. Hence, the Tucker method fits the framework of HOSVD and thus can be implemented as a HOSVD [26].

Lastly, CPD holds uniqueness under certain weak conditions [27], while the uniqueness of Tucker decomposition is not guaranteed. This is because the core tensor can be modified without affecting the fit. And such flexibility leads to some transformations that simplify the core tensor, reducing as many elements of the core tensor to zero as possible. Since these elements represent the interactions between corresponding components, more zero improves the uniqueness (to create orthogonality).

## 2.3. MACHINE LEARNING(ML) METHODS

With the arising popularity of machine learning techniques, proposals for utilizing machine learning methods to solve tensor decomposition have been made in the literature. The problem of finding the best decomposition is in essence an optimization problem, with the object of minimizing the Mean Square Error (MSE) between the exact factor matrices and the approximated factor matrices.

The Gradient Descent (GD) method is a classical solver for Least Square Error problems, making use of the convex property of the loss function. For example, [28] derives a positive-preserving gradient descent algorithm for finding a non-negative $n$-dimensional tensor factorization, and applies it to the computer vision field. Another famous application of ML is to decompose the user(**A**) × item(**B**) × context(**C**) rating tensor [16], which is popular in the applications in recommendation systems.

In [16] the author also introduces a well-known ML approach, Stochastic Gradient Descent (SGD), which can be used even in solving non-convex optimization problems. This is a very inclusive algorithm and can handle missing data relatively well. However, its randomness is a problem for signal processing, and thus, needs intelligent caching strategies. [6] applies SGD to a generalized CPD method, and the authors demonstrate the SGD method outperforms many networks based on their datasets. However, the errors they show are still rather large and they do not conduct any error analysis. Our work starts by replicating their methods for a better understanding of the methods and then continues with improving their methods based on our insights and analysis.

# 3

# Overview of Tensor and Canonical Polyadic Decomposition (CPD)

This chapter gives the definitions and properties of tensors related to our research, and introduces Canonical Polyadic Decomposition (CPD). Tensor is a convenient representation for high dimensional data and CPD is a typical method to reduce the computational complexity by factorizing a tensor into a sum of lower-rank representations. At the end of the chapter, we give a two-dimensional example to illustrate how CPD works with Fourier-cosine coefficient tensors.

## 3.1. Tensor: Basics

This section provides a basic introduction to tensors, including definitions and structure of the tensor. In addition, we also introduce some commonly used matrix operations for the tensor calculation to build a necessary knowledge base for subsequent work.

### 3.1.1. Definition and Structure

***Definition 3.1*** An Nth-order **tensor** $\mathscr{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ is a real N-dimensional array, where the index range in the $k$-th mode is from 1 to $I_k$ [29].

Tensor is a general way of representing arrays. For example, a vector is a first-order tensor, a matrix is a second-order tensor, and a third-order or even higher-order array shares the name of the tensor. Figure 3.1 visualizes the structure of a third-order tensor.

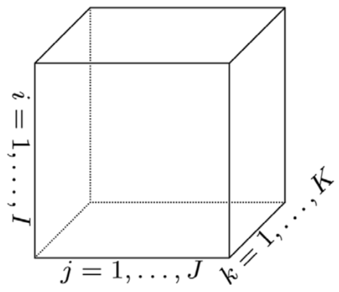


Figure 3.1: A third-order tensor: $\mathscr{X} \in \mathbb{R}^{I \times J. \times K}$[11], the element at position $(i, j, k)$ can be represented by $x_{ijk}$.

In the following, we collect the definitions of components of tensors[11] and a significant type of tensor, i.e. **rank-one tensors** .

***Entries*** : Similar as $a_{ij}$ is the $(i, j)$-th element of matrix $\mathbf{A}$, $\mathbf{x}_{ijk}$ denotes the $(i, j, k)$-th entry of a tensor.

***Fibers*** : Fibers are the higher-order analogue of matrix rows and columns. Third-order tensors have the column, row, and tube fibers, denoted by $\mathbf{x}_{:jk}$, $\mathbf{x}_{i:k}$, and $\mathbf{x}_{ij:}$.

***Slices***: Slices are 2-dimensional components of a tensor, and only one of the indices is fixed. For example, there are three dimensions in a third-order tensor, slices can be extracted from 3 directions, denoted by $\mathbf{X}_{\mathbf{i}::}$ (horizontal), $\mathbf{X}_{:\mathbf{j}:}$ (lateral), and $\mathbf{X}_{::\mathbf{k}}$ (frontal), respectively.

***Definition 3.2*** An Nth-order tensor $\mathscr{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ is **rank one** if it can be written as the outer product of N vectors, i.e.,

$$\mathscr{X} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \dots \circ \mathbf{a}^{(N)}. \tag{3.1}$$

here "$\circ$" denotes the outer product of vectors, which means that each entry of a tensor is represented by the product of the corresponding vector elements:

$$x_{i_1 i_2 \dots i_N} = a^{(1)}_{i_1} a^{(2)}_{i_2} \dots a^{(N)}_{i_N}, \text{ for all} 1 \leq i_n \leq I_n. \tag{3.2}$$

Figure 3.2 shows the construction of a rank-one tensor. Note that the CPD method assumes that a tensor can be approximated by the sum of rank-one tensors, which we will go into details later.

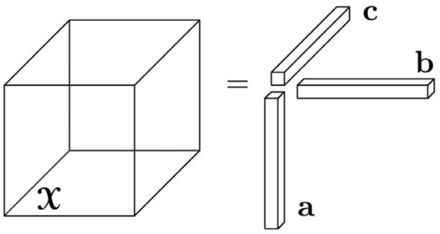


Figure 3.2: A rank-one model of a third-order tensor, $\mathscr{X} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$., The $(i,j,k)_{th}$ entry of $\mathscr{X}$ is calculated by $x_{ijk} = a_i \cdot b_j \cdot c_k$. [11]

### 3.1.2. MATHEMATICAL FOUNDATION

Tensor decomposition is a powerful technique to reduce computational complexity by finding lower-rank representations for tensors. Before we delve into decomposition approaches, let us review some matrix operations. Firstly, we introduce the unfolding technique that transforms a tensor into matrices. Then we give some common matrix products in the tensor calculation, including Kronecker product ($\otimes$), Khatri-Rao product ($\odot$), and Hadamard product ($\circledast$) [29], as well as the Frobenius norm, which is used to evaluate errors.

#### *Unfolding*

Unfolding a tensor is rearranging its slices into a matrix from different directions, also known as flattening. Compared to higher-order tensor calculations, matrix computations are much less abstract. Indeed, operations between tensors can be reformulated as matrix computations between unfolded matrices. There are various ways to flatten a tensor, but an essential type that is the most relevant to our discussion is the mode-$k$ unfolding.

Given a tensor $\mathscr{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, $I_1, I_2, \cdots, I_N \in I$, a mode-$k$ unfolding is defined by an $I_k \times (I/I_k)$ matrix whose columns are the mode-$k$ fibers [29]. To illustrate the concept clearer, we give an example of unfolding a third-order tensor $\mathscr{X} \in \mathbb{R}^{4\times3\times2}$. Then the 3 mode-$k$ unfolded matrices are constructed as

$$\mathscr{X}_{(1)} = \begin{bmatrix} x_{111} & x_{121} & x_{131} & x_{112} & x_{122} & x_{132} \\ x_{211} & x_{221} & x_{231} & x_{212} & x_{222} & x_{232} \\ x_{311} & x_{321} & x_{331} & x_{312} & x_{322} & x_{332} \\ x_{411} & x_{421} & x_{431} & x_{412} & x_{422} & x_{432} \end{bmatrix} \tag{3.3}$$

$$\mathscr{X}_{(2)} = \begin{bmatrix} x_{111} & x_{211} & x_{311} & x_{411} & x_{112} & x_{212} & x_{312} & x_{412} \\ x_{121} & x_{221} & x_{321} & x_{421} & x_{122} & x_{222} & x_{322} & x_{422} \\ x_{131} & x_{231} & x_{331} & x_{431} & x_{132} & x_{232} & x_{332} & x_{432} \end{bmatrix} \tag{3.4}$$

$$\mathscr{X}_{(3)} = \begin{bmatrix} x_{111} & x_{211} & x_{311} & x_{411} & x_{121} & x_{221} & x_{321} & x_{421} & x_{131} & x_{231} & x_{331} & x_{431} \\ x_{112} & x_{212} & x_{312} & x_{412} & x_{122} & x_{222} & x_{322} & x_{422} & x_{132} & x_{232} & x_{332} & x_{432} \end{bmatrix} \tag{3.5}$$

$\mathscr{X}_{(1)}$ is reshaped in a way that permutes the three slices of size $(4 \times 2)$ matrices side-by-side. $\mathscr{X}_{(1)}$ connects four $(3 \times 2)$ matrices and $\mathscr{X}_{(3)}$ is the reorganization of four $(2 \times 3)$ slices. Though there are such rules for unfolding, it does not have a strong physical meaning, for it is just used to simplify the computation process.

***Kronecker Product***

Kronecker product is widely used in tensor calculation since it provides a bridge to connect matrix computations and tensor computations. For matrices $A \in \mathbb{R}^{I \times J}, B \in \mathbb{R}^{K \times L}$, the Kronecker product denoted by $A \otimes B \in \mathbb{R}^{(IK) \times (JL)}$ gives a matrix as follows:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1J}B \\ a_{21}B & a_{22}B & \cdots & a_{2J}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}B & a_{I2}B & \cdots & a_{IJ}B \end{bmatrix} \tag{3.6}$$

Note that $A \otimes B \neq B \otimes A$, and $(A \otimes B)^T = A^T \otimes B^T$.

***Khatri-Rao Product***

Khatri-Rao and Hadamard products are actually submatrices of Kronecker products. An important instance of the Khatri-Rao product is based on column partitionings [29] and it requires input matrices to have the same number of columns. This operation is just compatible with tensor decomposition, and thus, this product is very practical in implementation. Given $A \in \mathbb{R}^{I \times K}, B \in \mathbb{R}^{J \times K}$, the Khatri-Rao product $A \odot B$ is defined by:

$$A \odot B = [a_1 \otimes b_1, a_2 \otimes b_2, \cdots, a_K \otimes b_K] \tag{3.7}$$

***Hadamard Product***

The Hadamard product is a pointwise (elementwise) product, and thus, matrices $A$ and $B$ need to be of the same size. For $A, B \in \mathbb{R}^{I \times J}$, the Hadamard product denoted by $A \circledast B \in \mathbb{R}^{I \times J}$ gives:

$$A \circledast B = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1J}b_{1J} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2J}b_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}b_{I1} & a_{I2}b_{I2} & \cdots & a_{IJ}b_{IJ} \end{bmatrix} \tag{3.8}$$

***Frobenius norm***

Given a tensor $\mathscr{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, its F-norm is defined as the square root of the sum of the squares of all its elements:

$$\|\mathscr{X}\|_F =< \mathscr{X}, \mathscr{X} >= \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x^2_{i_1 i_2 \cdots i_N}}. \tag{3.9}$$

Actually, many optimization problems for tensor decomposition often involve minimizing the sum of squares of the residual tensor elements, and the objective function is often written in the form of the square of the F-norm.

## 3.2. CANONICAL POLYADIC DECOMPOSITION(CPD)

Tensor models have overcome the limitation of matrix models for expressing high-dimensional data and have become an efficient tool in data analysis. Tensor decomposition techniques, which have also been well adopted in data mining, signal processing, and statistics [16], are proven to have more flexibility in finding the proper constraints that match with data properties and representative components in the data than matrix-based methods. And an effective application of tensor decomposition is dimension reduction.

In this section, we introduce Canonical Polyadic Decomposition (CPD), also known as Parallel Factor Analysis (PARAFAC) [30] in multilinear algebra, which has been applied successfully for modeling large-scale multi-dimensional and multi-relational data [31], [32].

### 3.2.1. CPD INTRODUCTION

***Definition 3.3*** A Polyadic Decomposition represents an Nth-order tensor $\mathscr{X} \in {}^{I_1 \times I_2 \times \cdots \times I_N}$ as a linear combination of rank-1 tensors in the form

$$\mathscr{X} \approx [\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(N)}]\!]_R \equiv \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \ldots \circ \mathbf{a}_r^{(N)}. \tag{3.10}$$

where $\mathbf{A}^{(n)}$ is a factor matrix, $\mathbf{a}_r^{(n)}$ is the $r$-th column of corresponding $\mathbf{A}^{(n)}$, $R$ is the tensor rank, which is a positive integer defined as the smallest value when the equation holds exactly. The minimum rank PD is called canonical PD (CPD) [16].

***Remark 3.1*** It is also noteworthy that the product of $\mathbf{AB}^T$, where $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{K \times R}$ is the same as the $R$ sum of rank-1 tensors of $\mathbf{A}, \mathbf{B}$. i.e

$$\begin{aligned}
\mathbf{AB}^T &= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1R} \\ a_{21} & a_{22} & \cdots & a_{2R} \\ \vdots & \vdots & \ddots & \vdots \\ a_{K1} & a_{K2} & \cdots & a_{KR} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{21} & \cdots & b_{K1} \\ b_{12} & b_{22} & \cdots & b_{K2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{1R} & b_{2R} & \cdots & b_{KR} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_K \end{bmatrix} \cdot \begin{bmatrix} \mathbf{b}_1^T & \mathbf{b}_2^T & \cdots & \mathbf{b}_K^T \end{bmatrix} \\
&= \mathbf{a}^{(1)} \circ \mathbf{b}^{(1)} + \mathbf{a}^{(2)} \circ \mathbf{b}^{(2)} + \cdots + \mathbf{a}^{(R)} \circ \mathbf{b}^{(R)}
\end{aligned}$$

where $\mathbf{a}_i, \mathbf{b}_i, i = 1, 2, 3, ..., K$, denote the rows of A and B, '$\cdot$' is the dot product. $\mathbf{a}^{(i)} \circ \mathbf{b}^{(i)}$ denotes the outer product of each column of A and B, an outer product is a rank-1 tensor, and the sum of $R$ rank-1 tensors forms up a tensor $\mathbf{AB}^T$.

***Remark 3.2*** As we can see from the definition, CPD can capture the low-dimension patterns of multi-dimensional data, as $R$ is generally smaller than the number of full columns of each factor matrix $\mathbf{A}_n$. Indeed, the lower-rank CPD model is widely applied to break the dimensionality curse. Another advantage of CPD is that under some mild conditions on factor matrices, the CPD is unique and can be found algebraically[27].

***Remark 3.3*** The tensor rank is a different concept from the matrix rank, as the former is the number of the rank-one tensors for the sum in the decomposition algorithms.

### 3.2.2. Alternating Least Square Algorithm (ALS)

Now we need to solve these factor matrices given a tensor, by evaluating the least square error between the product of $\mathbf{A}_n$ and tensor $\mathscr{X}$, i.e. the F-norm:

$$\min_{\{\mathbf{A}_n\}_{n=1}^N} ||\mathscr{X} - [\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(N)}]\!]_R||_F^2 \tag{3.11}$$

This optimization problem seems sophisticated since there are more than one matrix involved in the optimization objective. Hence, we will introduce an algorithm called Alternating Least Square (ALS) that transforms it into a linear problem.

To illustrate how ALS works, hereby we provide a 3D example. Considering a decomposition problem for the prototype tensor $\mathscr{X}$ and its factor matrices $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$, whereby $\mathbf{a}_r, \mathbf{b}_r$, and $\mathbf{c}_r$ are the $r$-th column, respectively:

$$\min_{\{\mathbf{A}_n\}_{n=1}^N} ||\mathscr{X} - \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r||_F^2 \tag{3.12}$$

Now we can combine the technique of tensor unfolding and Khatri-Rao product to build up a neat connection between tensor modes and factor matrices [29]:

$$\mathscr{X}_{(1)} = \mathbf{A} \cdot (\mathbf{C} \odot \mathbf{B})^T$$
$$\mathscr{X}_{(2)} = \mathbf{B} \cdot (\mathbf{C} \odot \mathbf{A})^T$$
$$\mathscr{X}_{(3)} = \mathbf{C} \cdot (\mathbf{B} \odot \mathbf{A})^T$$

Note that minimizing the F-norm of the above three equations above is equivalent to Eq (3.12). Hence, we can consider

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} ||\mathscr{X}_{(1)} - \mathbf{A} \cdot (\mathbf{C} \odot \mathbf{B})^T||_F^2 \tag{3.13}$$

This is actually a trilinear problem, for $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ are all unknown. However, if fixing $\mathbf{B}$ and $\mathbf{C}$, Eq (3.13) is linear w.r.t. $\mathbf{A}$. So it becomes possible to update each factor matrix one by one until the stopping criterion is satisfied, i.e.

$$\mathbf{A} \leftarrow \arg\min_{\mathbf{A}} ||\mathscr{X}_{(1)} - \mathbf{A} \cdot (\mathbf{C} \odot \mathbf{B})^T||_F^2$$

$$\mathbf{B} \leftarrow \arg\min_{\mathbf{B}} ||\mathscr{X}_{(2)} - \mathbf{B} \cdot (\mathbf{C} \odot \mathbf{A})^T||_F^2$$

$$\mathbf{C} \leftarrow \arg\min_{\mathbf{C}} ||\mathscr{X}_{(3)} - \mathbf{C} \cdot (\mathbf{B} \odot \mathbf{A})^T||_F^2$$

The above calculation process is the well-known ALS algorithm, turning a multilinear problem into highly structured least square problems., and it is a popular algorithm for solving factor matrices for lower-rank CPD models. It's interesting to see that ALS is actually very easy to implement but can help a lot with thinking in 'multi-dimension.'

## 3.3. GCPD WITH A 2D TOY EXAMPLE

In this section, we introduce a generalized form of CPD involving the Fourier-cosine series expansion, which is called GCPD in [6]. The method proposed by [6] only requires the prototype to be a multivariate function that has compact support and continuous derivatives, which can be well approximated by multidimensional Fourier series. Such generalization gives a strategy to decompose the Fourier series expansion of a function, which also endows the decomposition with more flexibility and the possibility to solve the decomposition via machine learning methods.

To illustrate the idea of GCPD, we implement an example of approximating two-dimensional Gaussian probability density functions in this section. The main process is to approximate the 2D Gaussian pdf by Fourier-cosine series expansion, whereby the series coefficients tensor is firstly computed by Clenshaw-Curtis quadrature rule [3], then is decomposed into lower-rank factor matrices via CPD.

### 3.3.1. GAUSSIAN 2D PDF

First of all, let us have a brief review of the Gaussian pdf, Fourier series, as well as its cosine expansion. Notably, the Gaussian pdf has a superior property that its integration is always one regardless of the number of dimensions, which can be used to directly check the accuracy level of the numerical integration.

***Definition 3.4*** The probability density function of a two-dimensional Gaussian distribution is given by:

$$f_X(x_1,x_2)=\left(2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}\right)^{-1} exp\left[-\frac{1}{2(1-\rho^2)}\left(\frac{(x-\mu_1)^2}{\sigma_1^2}-\frac{2\rho(x-\mu_1)(y-\mu_2)}{\sigma_1\sigma_2}+\frac{(y-\mu_2^2)}{\sigma_2^2}\right)\right] \tag{3.14}$$

$f_X(x_1,x_2)$ can also be denoted by $X \sim N(\mu,\Sigma)$, where $\mu=(\mu_1,\mu_2)$ is the mean vector, $\Sigma=\begin{bmatrix}\sigma_1^2 & \rho\sigma_1\sigma_2\\ \rho\sigma_1\sigma_2 & \sigma_2^2\end{bmatrix}$ is the covariance matrix, $\sigma_1$ and $\sigma_2$ are the standard deviation, and $\rho$ is the correlation coefficient.

### 3.3.2. 2D FOURIER SERIES

Since the Gaussian pdf decays to zero values when the variables move away from the mean values in each dimension, we can truncate the pdf function such that it is defined on a finite support. Let us denote the truncation domain as $\mathscr{D}=\{x_1\in[-l,l],x_2\in[-h,h]\}$, then the Fourier series of the truncated pdf can be written as:

$$\begin{aligned} f(x_1,x_2) = \sum_{m,n=0}^{\infty} \lambda_{mn}[A_{mn}\cos\frac{\pi m x_1}{l}\cos\frac{\pi n x_2}{h} + B_{mn}\sin\frac{\pi m x_1}{l}\cos\frac{\pi n x_2}{h} \\ + C_{mn}\cos\frac{\pi m x_1}{l}\sin\frac{\pi n x_2}{h} + D_{mn}\sin\frac{\pi m x_1}{l}\sin\frac{\pi n x_2}{h}]. \end{aligned} \quad (3.15)$$

And we need to introduce the series truncation error here since we do not compute the infinite sum of the series terms. Instead, we truncate the series by the first $K$ terms in each dimension without losing significant accuracy.

$$\begin{aligned} f(x_1,x_2) \approx \sum_{m,n=0}^{K} \lambda_{mn}[A_{mn}\cos\frac{\pi m x_1}{l}\cos\frac{\pi n x_2}{h} + B_{mn}\sin\frac{\pi m x_1}{l}\cos\frac{\pi n x_2}{h} \\ + C_{mn}\cos\frac{\pi m x_1}{l}\sin\frac{\pi n x_2}{h} + D_{mn}\sin\frac{\pi m x_1}{l}\sin\frac{\pi n x_2}{h}] \end{aligned} \quad (3.16)$$

$$\lambda_{mn} = \begin{cases} \frac{1}{4} & m = n = 0 \\ \frac{1}{2} & m > 0, n = 0, \text{or } m = 0, n > 0 \\ 1. & m > 0, n > 0 \end{cases}$$

the coefficients are double integrals of $f$ and basis functions:

$$A_{mn} = \frac{1}{lh}\int_{-l}^{l}\int_{-h}^{h} f(x_1,x_2)\cos\frac{\pi m x_1}{l}\cos\frac{\pi n x_2}{h}dx_1dx_2$$
$$B_{mn} = \frac{1}{lh}\int_{-l}^{l}\int_{-h}^{h} f(x_1,x_2)\sin\frac{\pi m x_1}{l}\cos\frac{\pi n x_2}{h}dx_1dx_2$$
$$C_{mn} = \frac{1}{lh}\int_{-l}^{l}\int_{-h}^{h} f(x_1,x_2)\cos\frac{\pi m x_1}{l}\sin\frac{\pi n x_2}{h}dx_1dx_2$$
$$D_{mn} = \frac{1}{lh}\int_{-l}^{l}\int_{-h}^{h} f(x_1,x_2)\sin\frac{\pi m x_1}{l}\sin\frac{\pi n x_2}{h}dx_1dx_2$$

In this section, we solve these integrals simply by an advanced numerical integration method, Clenshaw–Curtis quadrature rule [3], based on expanding the integrands regarding Chebyshev polynomials. As mentioned in the Introduction, the advantage of such advanced quadrature rules is that we do not need many points to ensure approximation accuracy.

However, the expansion above seems verbose and troublesome, and actually, it can be simplified by Fourier series expansion of the even extension $f(\cdot)$, also known as Fourier-cosine series expansion, i.e.

$$f(x_1,x_2) = \sum\nolimits_{m,n=0}^{\prime K}[A_{mn}\cos\frac{\pi m(x_1+l)}{2l}\cos\frac{\pi n(x_2+h)}{2h}] \quad (3.17)$$

And the computation formula for coefficients changes accordingly, i.e.

$$A_{mn} = \frac{1}{lh}\int_{-l}^{l}\int_{-h}^{h} f(x_1,x_2)\cos\frac{\pi m(x_1+l)}{2l}\cos\frac{\pi n(x_2+h)}{2h}dx_1dx_2 \quad (3.18)$$

Compared with Eq (3.16), this cosine expansion saves quite some computations. The prime here has the same effect as $\lambda_{mn}$, i.e., denoting the first term of the summation for $m$ and $n$ has a weight of 1/2.

We show the efficiency of Fourier-cosine series expansions in Figure 3.3. The right-hand side picture is the reconstruction of the function while the $K^2$ elements of the coefficient matrix are computed by the Clenshaw-Curtis quadrature rule. And for multi-dimensional functions, the number of Fourier coefficients grows exponentially. Hence the crucial task is to find a lower-rank representation of the Fourier coefficients in high-dimensional situations so that the computational cost can be reduced. We will look into such solutions based on CPD in the following chapters.

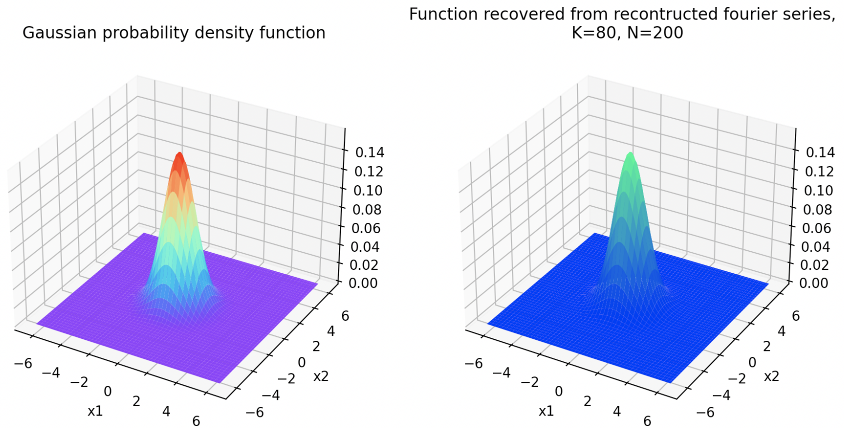


Figure 3.3: A Gaussian probability density function in 2D and a approximated one based on Fourier series with truncation number K = 80, Clenshaw-Curtis quadrature points T = 200, leading to $L_\infty$ error =5.2138004297695477e-11.

### 3.3.3. Implementation of CPD and sensitivity analysis

This section gives a 2D example of how to decompose a matrix filled with Fourier-cosine series coefficients via CPD.

Starting from the training dataset, given a 2D Gaussian distribution $N(x_1, x_2; \mu, \Sigma)$, the sample points are drawn uniformly on a 2D plane by constructing equidistant mesh points, and the target function values are obtained by matching the values on the sample points with a given probability density function.

Then the coefficient matrix, also known as the 2D ‘tensor’, can be calculated by Eq(3.18). With this ‘tensor prototype’, we can obtain the lower-rank factor matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ via the ALS algorithm. And here we choose the Conjugate Gradient method to solve the least square problem.

The whole process is described as **Algorithm1** below. Since we need to populate the coefficient tensor explicitly in this algorithm, it is named Fourier Series Approximation with Explicit Tensor (FSA-ET).

**Algorithm 1** FSA-ET

```
Input: $\mathscr{X}, \mathbf{y}, R, K$
Initialize $\{\mathbf{A}_n \in \mathbb{R}^{K\times R}\}_{n=1}^{N}$
repeat
  for n=1 to N do
    Solve for each $\mathbf{A}_n$ by Conjugate Gradient method
    Update $\mathbf{A}_n$ by ALS as stated in section 3.2.2
  end for
until stopping criteria is satisfied
return $\mathbf{A_n}_{n=1}^{N}$
```

In this experiment, we define the integration truncation range by the percent point function (PPF), which can also be explained as the inverse of cdf (cumulative distribution function) percentiles: a lower input for PPF results in a broader truncation range.

Next, we conduct convergence analysis w.r.t. the number of cosine terms $K$, and quadrature points $T$ used for Clenshaw-Curtis, which helps in determining a proper value of each parameter. To clearly observe the error behavior, we need to suppress the error from $K$ (or $T$) as small as possible while testing $T$ (or $K$). The error convergence w.r.t. $K$ is plotted in Figure 3.4.

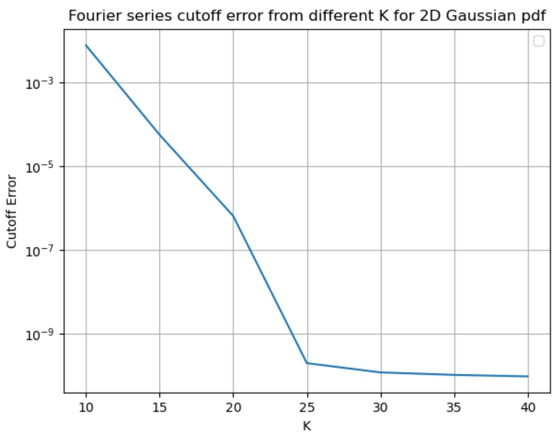


Figure 3.4: $L_\infty$ error among the increasing truncation terms, where the Fourier coefficients are computed by the Clenshaw-Curtis quadrature rule with 200 quadrature points, which is very adequate for the integration. And when $K$ is 25 the cutoff error reaches $10^{-10}$, which is consistent with the integration truncation range since it is chosen as ppf($10^{-10}$).

An exponential error convergence rate is observed, since the y-axis is in log-scale, which is consistent with the convergence theory of Fourier series expansion on sufficiently smooth functions. The elbow point at $K$ equals 25 means 25 Fourier-cosine expansion terms are enough to recover our target function in this example.

Another error source depends on the number of quadrature points $T$ when estimating the integration for each Fourier coefficient. Note that there is a much faster way to calculate the Fourier-cosine coefficients for density functions, i.e. the COS method [33], as we will emphasize in Chapter 6. However, we keep this generic method of solving the coefficients using a quadrature rule here, since the solution we develop here is designed for more general integrand functions than the density functions. Figure 3.5 shows that, when fixing $K$ at 50 to suppress the series truncation error, we obtain the error convergence w.r.t. $T$:

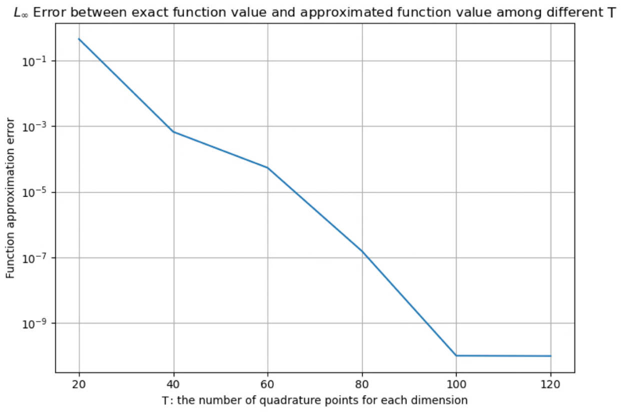


Figure 3.5: $L_\infty$ errors of the increasing quadrature points for integration, where $K$=50, which is supported by the conclusion from Figure 3.4. Again the error converges to $10^{-10}$.

Note that here the L-infinity error comes from comparing the exact function values and approximated function values recovered by Fourier-cosine series expansion, whereby the coefficients are computed numerically using the Clenshaw-Curtis quadrature rule.

The testing results above indicate that setting the parameters $(K, T)$ as (50, 100) is conservative enough to accurately recover the Gaussian density function from its Fourier-cosine series expansion using numerically computed coefficients $\mathcal{K}_{2D}$.

Now we are ready to apply CPD having the key $\mathcal{X}_{2D}$ in our hands. In other words, we aim to find the lower-rank factor matrices $\mathbf{A}_1$ and $\mathbf{A}_2$, which can minimize the objective function as below.

$$f = \frac{1}{2}||\mathcal{X}_{2D} - \mathbf{A}_1\mathbf{A}_2^T||_F^2. \tag{3.19}$$

Calculate for the partial derivative w.r.t. $\mathbf{A}_1$ and set it to 0, we have

$$\frac{\partial f}{\partial \mathbf{A}_1} = (\mathcal{X}_{2D} - \mathbf{A}_1\mathbf{A}_2^T)\mathbf{A}_2 \tag{3.20}$$

$$\mathbf{A}_1\mathbf{A}_2^T\mathbf{A}_2 = \mathcal{X}_{2D}\mathbf{A}_2. \tag{3.21}$$

Then we can solve for factor matrix $\mathbf{A}_1$ in Eq (3.22) via CG method, with $\mathbf{A}_2$ fixed. Then repeat the same solving process for $\mathbf{A}_2$ in the next round. $\mathbf{A}_1$ and $\mathbf{A}_2$ are updated alternatively until the reconstructed tensor is close enough to the original tensor.

Note that when $\mathbf{A}_1$ and $\mathbf{A}_2$ are of full rank, i.e. $\mathbf{A}_1, \mathbf{A}_2 \in \mathbb{R}^{K\times K}$, there is no error introduced in the CPD, and thus, the reconstruction error is dominated by the integration truncation error because we have chosen very conservative choices of $K$ and $T$. Next, we gradually reduce the rank $R$ so that $\mathbf{A}_1$ and $\mathbf{A}_2 \in \mathbb{R}^{K\times R}$, where $R \leq K$, aiming at finding the smallest R which ensures a sufficiently high level of accuracy. As Figure 3.6 suggests, we obtain $\mathbf{A}_1$ and $\mathbf{A}_2$ with rank of 15 instead of 50.

Meanwhile, the algorithm suggests that the computational complexity is $O(2KR)$, i.e. it is linear in $R$, which is confirmed by our experiment results in Figure 3.7. This reduction in computational costs is more prominent in higher dimensions, whereby the Fourier-cosine coefficients can be computed with complexity $O(NKR)$, instead of $O(K^N)$.

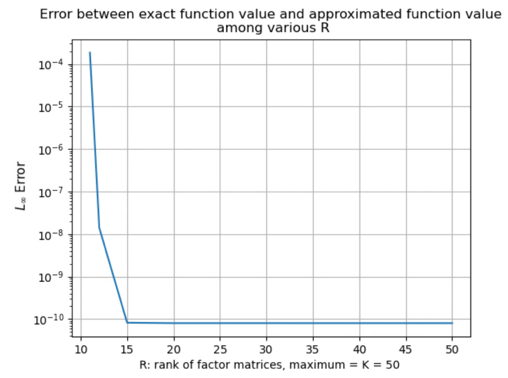


Figure 3.6: The error while lowering the rank of factor matrices. R less than 10 diverges

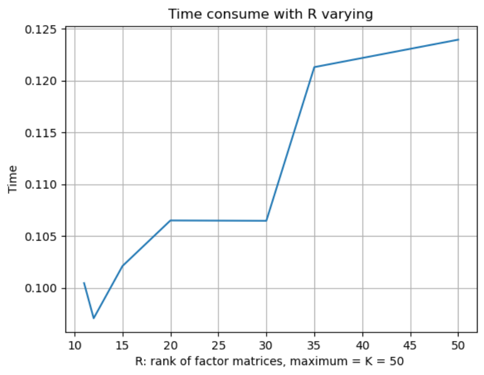


Figure 3.7: Time spend on reconstructing function values with coefficient tensor made of factor matrices, which is generally consistent with the computational complexity regarding to the change of R.

However, the downside of this approach is that we still need to generate the Fourier coefficient tensor explicitly first, which requires $K^N$ times of integrations by the Clenshaw-Curtis quadrature rule and is unbearable when $N$ is big. Hence we would like to find the lower-rank factor matrices directly without the need to populate the coefficient tensor in the next chapters.

# 4

# An Existing Method: Hidden Tensor Factorization and Machine Learning

## 4.1. Hidden Tensor Factorization

Recall that at the end of the last chapter, we provided a toy example of implementing CPD, which is not effective to reduce the computational complexity, since the population of the coefficient tensor still suffers from the curse of dimension. To solve this problem, the authors of [6] propose to minimize the distance between the function values approximated by truncated Fourier-cosine series expansion, of which the Fourier coefficient tensor is replaced by its representation using the lower-rank factor matrices, and exact function values. Since this approach bypasses the population of the explicit coefficient tensor, it is named 'Hidden Tensor Factorization' (HTF). Furthermore, they utilize a supervised machine learning method, Stochastic Gradient Descent (SGD), to solve the lower-rank matrices.

### 4.1.1. Replication of the existing model

Our work starts with reproducing the model of [6]. The authors aim to find optimized lower-rank factor matrices that can help recover function values the best. The objective function can be formulated as:

$$\frac{1}{M}\sum_{m=1}^{M} L(y_m - f(x_m)) + G(f). \tag{4.1}$$

Here $L(\cdot)$ denotes the loss function, $y_m$ is the exact function value at $x_m$, and $f(x_m)$ is an approximated function value based on the Fourier-cosine series expansion, $m$ denotes the index of sampling points, and $M$ is the total number of sampling points. $G(\cdot)$ is a regularization function, which is used to prevent overfitting.

Next, we we write out $f(x_m)$ explicitly. Let the cosine basis functions for $n$-th dimension evaluated at all $M$ sample points be collected in matrix $\mathbf{V}_n \in \mathbb{R}^{K\times M}$ as $\mathbf{V}_n[k,m] = \phi_k(x^m[n])$, whereby $K$ is again the number of leading terms in the series expansion, $\phi_k(\cdot)$ is the $k$-th cosine basis function. That is, there are $M$ columns in a $\mathbf{V}_n$, each representing a $K$-length vector of cosine basis functions evaluated at $x_m$. Let $\mathbf{A}_i$ be the factor matrix defined by CPD.

We choose the MSE as the loss function and rewrite the optimization problem as:

$$\min_{\{\mathbb{A}_n\}_{n=1}^N} \frac{1}{M}\|\mathbf{y} - (\odot_{n=1}^N \mathbf{V}_n)^T(\odot_{i=1}^N \mathbf{A}_i)\mathbf{1}\|_2^2 + \sum_{n=1}^N \rho\|\mathbf{A}_n\|_F^2 \tag{4.2}$$

where $\odot$ denotes the Khatri-Rao product, and $\rho$ is a regularization parameter. In fact, the regularization term is not involved in our application of integration, since the coefficients are used to solve the integration problem of the corresponding specific function, and thus the loss function does not need to be regularized over datasets and find the generally best fit.

Eq (4.2) combines the cosine basis functions $\mathbf{V}_n$ and factor matrices $\mathbf{A}_n$ exquisitely and briefly just by product operations, which can be seen as a representative example that there are many flexible and versatile uses of matrix products in tensor applications.

Note that the coefficient tensor reorganized by HTF is different from the one computed by the Clenshaw-Curtis quadrature rule, for the first terms of Fourier coefficients generated in HTF have already been scaled because they are fitted from the function value matching, while in the last chapter we still need to halve the first column/row of the coefficient tensor. The computation details of how the Khatri-Rao product connects the cosine basis functions and coefficient factor matrices can be found in the appendix.

And if we compute the function value pointwisely, we can also avoid the Khatri-Rao product by writing it in scalar form [6]:

$$\min_{\{\mathbf{A}_n\}_{n=1}^N} \frac{1}{M}\sum_{m=1}^M \left(y_m - \left(\circledast_{n=1}^N \left(\mathbf{V}_n[:,m]^T\mathbf{A}_n\right)\right)\mathbf{1}\right)^2 \tag{4.3}$$

where $\circledast$ denotes the Hadamard product. This scalar formula is helpful when implementing SGD. And we can also apply ALS to compute $\mathbf{A}_1$ and $\mathbf{A}_2$ respectively, which is [6]:

$$\min_{\mathbf{A}_n} \frac{1}{M}\sum_{m=1}^M \left(y_m - \mathbf{V}_n[:,m]^T\mathbf{A}_n\mathbf{Q}_n[:,m]\right)^2 \tag{4.4}$$

where $\mathbf{Q}_n = \left(\circledast_{i\neq n}\left(\mathbf{A}_i^T\mathbf{V}_i\right)\right) \in \mathbb{R}^{R\times M}$. This notation shows that the item needs to be updated each time is only $\mathbf{A}_n$, for other $\mathbf{A}_{i\neq n}$ are locked in $\mathbf{Q}_n$ temporarily. Now we are prepared to use the supervised machine learning method, SGD, to solve this optimization problem and investigate its performance.

## 4.2. MACHINE LEARNING APPROACH

In this section, we first use SGD to solve the optimization problem, i.e. Eq (4.4), and test the algorithm's convergence and parameter settings via experiments. Then we transform the original problem, which is function fitting, into our application need: integration. At the end of the section, we give some performance analysis of SGD and decide to improve the algorithm.

### 4.2.1. SOLVING LOSS FUNCTION WITH SGD

We begin by introducing the Gradient Descent (GD) method, a useful tool for solving convex linear equations. It is an iterative method, which updates the approximations by going a specific step toward the negative gradient:

$$x_{k+1} = x_k - \alpha_k \nabla \left( \sum_{i=1}^{M} f_m(x_k) \right) \tag{4.5}$$

And when the number of sampling points ($M$) is vast, we prefer using Stochastic Gradient Descent (SGD) to save computational cost. The difference is that GD updates all the approximations at one time, while SGD only updates one randomly chosen point so that it is capable of reducing the computational complexity while still utilizing the gradient information. Here we take an intermediate strategy that samples a batch of training data points stochastically each time and updates them toward gradient descent direction simultaneously with a prescribed step size $\alpha$, i.e.

$$\mathbf{A}_n \leftarrow \mathbf{A}_n - \alpha \mathbf{G}_n \tag{4.6}$$

where $\mathbf{G}_n$ is the gradient w.r.t $\mathbf{A}_n$:

$$\begin{aligned} \mathbf{G}_n = & \frac{1}{|\mathscr{F}|} \sum_{m \in \mathscr{F}} \left( \mathbf{V}_n[:, m]^T \mathbf{A}_n \mathbf{Q}_n[:, m] \right) \mathbf{V}_n[:, m] \mathbf{Q}_n^T[:, m] \\ & - \frac{1}{|\mathscr{F}|} \sum_{m=1}^{M} y_m \mathbf{V}[:, m] \mathbf{Q}^T[:, m]. \end{aligned} \tag{4.7}$$

$\mathscr{F}$ is a batch with size $|\mathscr{F}|$ and $m$ is the index of points in $\mathscr{F}$. So the computational complexity has been reduced to $O(KR|\mathscr{F}|)$ for each iteration.

Notably, SGD does not guarantee that each step goes toward the minimum since it updates a batch of randomly sampled points each time, so there are probably fluctuations in errors during the iteration process. And the choice of step size is also important because a wide step can cause the solver to miss the minimum and the error thus can bounce to infinity.

In this subsection, we implement two methods computing $\mathbf{A}_n$ with SGD in 2D: **Algorithm 2** and **Algorithm 3**, with the latter being a modification of the former.

**Algorithm 2** FSA-HTF (SGD)

```
Input: $\mathbf{X}, \mathbf{y}, R, K, |\mathscr{F}|$
Initialize $\{\mathbf{A}_n \in \mathbb{R}^{K\times R}\}_{n=1}^{N}$
repeat
    Sample $|\mathscr{F}|$ data points
    for n=1 to N do
        Update $\mathbf{A}_n$ via Eq(4.7)
        Update $\mathbf{Q}_n$ via Eq(4.8)
    end for
    Compute MSE using $\mathbf{X}, \mathbf{y}$
until $\max_{\text{iter}}$ is reached or MSE is not imporoved
Return $\mathbf{A}_n$
```

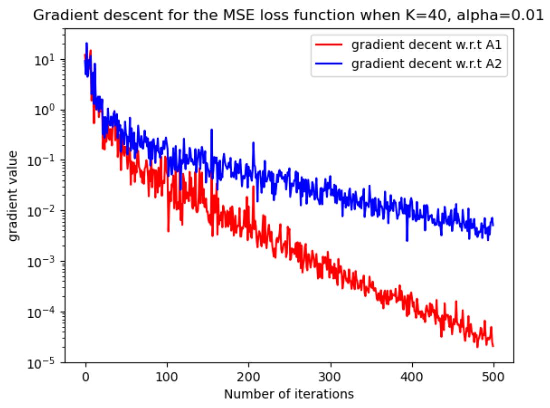


Figure 4.1: Gradient descent process when he number of cosine terms K = 40, step size $\alpha$= 0.01

**Algorithm 2** is based on a fixed number of $K$. Taking $K = 40$ as an example, we examined how the gradients w.r.t. $\mathbf{A}_1$ and $\mathbf{A}_2$ decrease in the SGD method.

Figure 4.1 shows the descent is not a smooth curve since each step is taken randomly, and the direction can be back and forth but goes down as the number of iterations increases. However, when $K$ is fixed during the whole training, it's not efficient to update the gradient since we begin from a large coefficient matrix, and we have to implement a relatively small step size to ensure the initial guess does not cause divergence. Moreover, $\mathbf{A}_1$ and $\mathbf{A}_2$ have different convergence rates, while simultaneous descent for both variables is expected.

Hence, we modify our implementation by starting with a small $K$, then increasing $K$ with a certain integer $b$ after enough iterations each time, and ending when $K$ reaches the prescribed maximum value. This modification is described in **Algorithm 3**.

**Algorithm 3** FSA-HTF (SGD-modified K)

```
Input: $\mathbf{X}, \mathbf{y}, R, K_0, b, K, |\mathscr{F}|$
Initialize $\{\mathbf{A}_n \in \mathbb{R}^{K_0 \times R}\}_{n=1}^{N}$
repeat
    Sample $|\mathscr{F}|$ data points
    for n=1 to N do
        Update $\mathbf{A}_n$ via Eq(4.7).
        Update $\mathbf{Q}_n$ via Eq(4.8).
    end for
    Compute MSE using $\mathbf{X}, \mathbf{y}$
    if $K_0 < K$ and MSE has not improved, then
        for n=1 to $N$ do
            $\mathbf{A}_n \leftarrow \begin{bmatrix} \mathbf{A}_n \\ \mathbf{0}^T \end{bmatrix}$
        end for
        $K_0 = K_0 + b$
    end if
until max$_{\text{iter}}$ is reached or MSE is not imporoved
Return $\mathbf{A}_n$
```

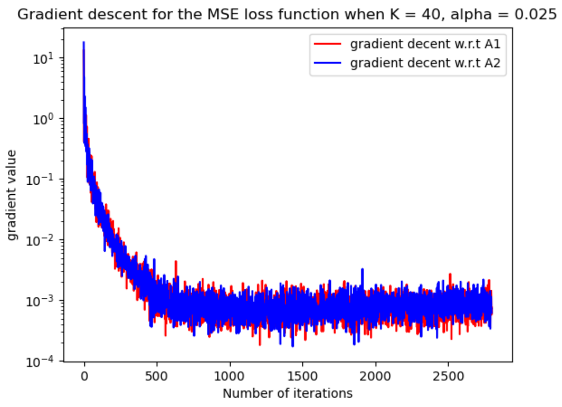
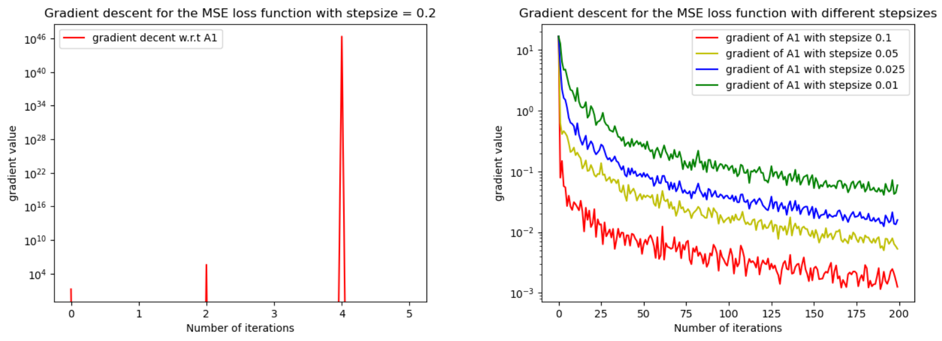


Figure 4.2: Gradient descent process with increasing K, step size = 0.025

In this way, the gradient drops synchronously in Figure 4.2, and we can have a relatively larger step size since we start from a smaller initial matrix.

Indeed, a reasonably larger step size is beneficial to speed up the descent process. However, it can't be arbitrarily large, for the landing point may jump over the minimum and end up with an infinitely large error. To study this sensitivity, we test the descent speed with various step sizes. The plot on the left shows the gradient can be incredibly large if an overly wide step size is chosen. And it also does not mean that we need to set the length of steps as small as possible for guaranteed convergence, since a too-small step size decelerates converge speed obviously, as shown in Figure 4.3 (b).



(a) Exploded gradient with a large step size (b) Gradient curves with different step sizes

Figure 4.3: Various step sizes show different descent speeds

Now we have a basic idea of how to make SGD work with appropriate parameters. Next, we explain how it can be applied to compute the integration.

### 4.2.2. APPLY SGD TO INTEGRATION AND RESULTS ACHIEVED

Recall that we approximate the integrand by its Fourier-cosine series expansion, and integrate this expansion on a truncated domain, then the Fourier coefficients and cosine series are finite and thus can be exchanged by Fubini's Theorem [34]:

$$\begin{aligned}
&\int_{a_1}^{b_1}\int_{a_2}^{b_2} {\sum}'^{K}_{k_1,k_2=0} \mathbf{A}_{k_1k_2} \cos k_1x_1 \cos k_2x_2 dx_1 dx_2 \\
&\overset{Fubini}{=} {\sum}'^{K}_{k_1,k_2=0} \mathbf{A}_{k_1k_2} \int_{a_1}^{b_1}\int_{a_2}^{b_2} \cos k_1x_1 \cos k_2x_2 dx_1 dx_2 \\
&= \frac{1}{4}(b_2-a_2)(b_1-a_1)\mathbf{A}_{00} \\
&= (b_2-a_2)(b_1-a_1)\mathbf{A}'_{00}
\end{aligned} \tag{4.8}$$

Note that $\mathbf{A}_{00}$ denotes the coefficient computed in the traditional way, while $\mathbf{A}'_{00}$ is gained by the machine learning method so it is already scaled as we have explained before. Eq (4.8) proves the integration can be transferred to the computation of Fourier-cosine coefficients. Moreover, it is interesting to see there is a great simplification in this application: although all the coefficients are involved in training, only $\mathbf{A}_{00}$ is used for the integration need. Mathematically, this seems a detour, as $\mathbf{A}_{00}$ by definition corresponds to the integral we aim to solve. However, linking an integral to the coefficient of the first term in the Fourier-cosine series expansion of the integrand is the key to utilize machine learning methods to solve the integral.

Hence, we are already at the destination after we solve the Fourier-cosine coefficients by SGD, and this desirable connection between numerical integration and machine learning is made via Fourier-cosine series expansion and CPD. Now, we can look into the results of SGD. Firstly we will look at the convergence speed of the algorithm.

Usually, machine learning algorithms are used to learn an unknown mapping from the given data, then it's better to have as much information from the training and validation as possible with acceptable time consumption. And we followed the convention of machine learning in previous experiments, with relatively large datasets, which corresponds to a lengthy training process. However, we can not just let the machine run for a longer time and ignore the possible optimization of time consumption.

Note that we already know the probability density function of the Gaussian distribution, and what we need to solve is the coefficient matrix of size $K \times R$, which contains $KR$ unknown elements. Based on the number of unknowns, we design experiments to figure out whether the training cost can be reduced.

To suppress the error from Fourier series truncation, we choose $K = 55$ following the conclusion from Chapter 3. And because of the stochastic nature of the algorithm, we repeat the test 10 times and record the average of the 10 measurements.

The test results in Table 4.1 show the decrease of sampling points does not increase the function error, while the computational time is indeed reduced. Notice that the integration error change is not regular, which may be caused by the function not being fitted accurately enough, because such irregularity disappears when we use a better optimizer in the next chapter. The reason for taking $K^2$ into account is that the unknown coefficients are in the form of a matrix with size $K \times K$ when R = K, so we just draw $K^2$ samples in the integration truncation domain.

K = 55, R = 55 iteration = 1000

| $T^2$ | function error | integration error | CPU time(sec) |
|---|---|---|---|
| $200^2$ | 0.0016875335 | 0.009405922 | 425.73384984 |
| $150^2$ | 0.0016985815 | 0.004916333 | 235.49313409 |
| $100^2$ | 0.0016884715 | 0.007633706 | 106.11341623 |
| $K^2$ | 0.0016958506 | 0.005084954 | 40.97256137 |

Table 4.1: The number of grid points in each dimension is $T$, with total iteration times = 1000.

Based on the number of sampled points being $K^2$, which does not introduce the lower-rank error yet, we decrease the number of iterations monotonically in order to abandon unnecessary iterations.

| | | K = 55, R = 55 $T^2 = K^2$ | |
|---|---|---|---|
| iteration | function error | integration error | CPU time(sec) |
| 1000 | 0.0016958506 | 0.005084954 | 40.97256137 |
| 800 | 0.00169183901 | 0.007014089 | 34.64054515 |
| 600 | 0.0017084173 | 0.004420803 | 26.75869117 |
| 400 | 0.0019311009 | 0.015517829 | 16.5032237 |

Table 4.2: The iteration numbers vary when the number of grid points in each dimension is $T = K = 55$.

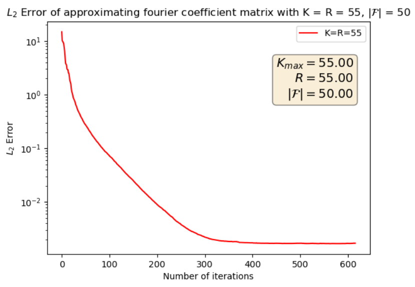
Compared with 1000 iterations, the function error does not increase significantly when iterating 600 times in Table 4.2, while the calculation time is nearly halved. There is always a subtle trade-off between accuracy and time consumption. Since the sacrifice on error is still acceptable, we choose $K^2$ samplings and 600 iterations as the baseline for the following experiments.

Considering that we adjust the algorithm to draw a batch of points stochastically each time, of which the size is $|\mathscr{F}|$, we can also test the effect of various batch sizes.

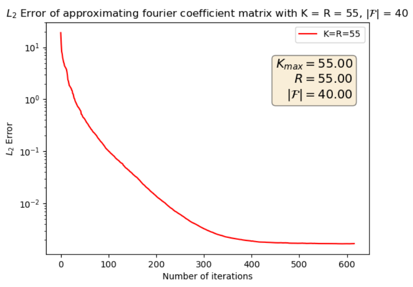
| | | K = 55, R = 55 iteration = 600 | |
|---|---|---|---|
| $\|\mathscr{F}\|$ | function error | integration error | CPU time(sec) |
| 50 | 0.0017084173 | 0.004420803 | 26.75869117 |
| 40 | 0.0017327550 | 0.007414279 | 23.32856766 |
| 30 | 0.0017623425 | 0.007873943 | 22.27834607 |
| 20 | 0.0018239328 | 0.007128875 | 20.6408789 |
| 15 | 0.0285444075 | 0.425194181 | 20.06425794 |
| 10 | 0.0203284150 | 0.430241075 | 19.11257073 |

Table 4.3: Implementation results with $55 \times 55$ samplings and 600 iterations for each $\mathscr{F}$.

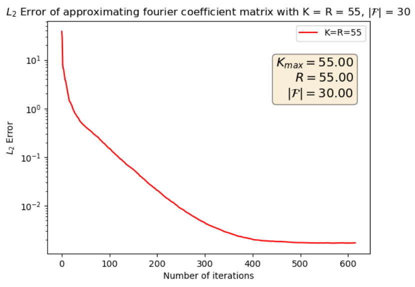
As seen from Table 4.3, the computational time decreases as $|\mathscr{F}|$ reduces, without influencing the function error. However, the convergence collapses when $|\mathscr{F}|$ is too small (less than 20). The following plots visualize the function error drawn from the table. To this end, it is better to choose $|\mathscr{F}|$ around 20 to 30.



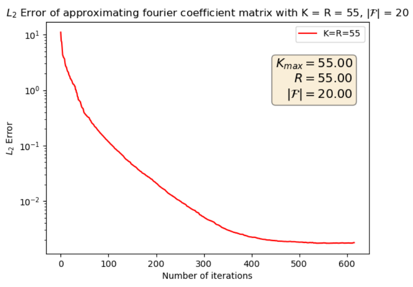
(a) sample 55 × 55 points, batch size = 50



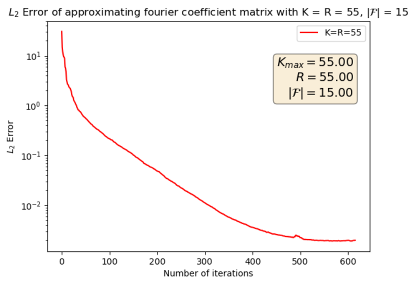
(b) sample 55 × 55 points,batch size = 40



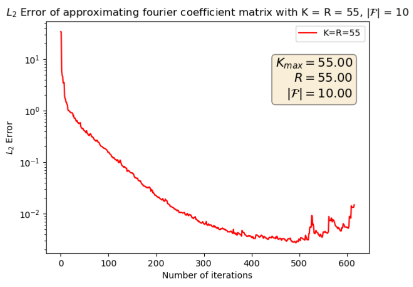
(c) sample 55 × 55 points, batch size = 30



(d) sample 55 × 55 points, batch size = 20



(e) sample 55 × 55 points, batch size = 15



(f) sample 55 × 55 points, batch size = 10

Figure 4.4: $L_2$ function error when $|\mathscr{F}|$ decreases.

We continue with the lower-rank test, based on the experimental results that reducing training data size to $K \times K$ does not lose the accuracy. We fix the values of other parameters than the tensor rank according to the tests before and vary the tensor rank value. We can fix $|\mathscr{F}| = 30$ with 600 iterations considering the outcomes from Table 4.2 and Table 4.3.

| Sampling strategy: $K \times K$ | K = 55, $\vert\mathscr{F}\vert = 30$ iteration = 600 | | |
|---|---|---|---|
| R | function error | integration error | CPU time(sec) |
| R = K= 55 | 0.001755357 | 0.005539071 | 23.58637767 |
| 45 | 0.001786917 | 0.005718527 | 27.01278413 |
| 40 | 0.001952586 | 0.009259844 | 29.25595766 |
| 35 | 0.002818757 | 0.010615879 | 29.69673709 |
| 30 | 0.003348965 | 0.015767389 | 30.40672392 |
| 25 | 0.0066356291 | 0.0404400067 | 29.79918631 |
| 20 | 0.011169203 | 0.035130067 | 29.80676714 |
| 15 | 0.036395637 | 0.224068644 | 26.81936796 |
| 10 | 0.070233826 | 0.210576595 | 25.03277316 |

Table 4.4: Implementation of reduced rank with 55 × 55 samplings and 600 iterations for $|\mathscr{F}|= 30$.

The decrease of R indicates a loss of information from the real coefficient tensor. The decreasing trend of accuracy is also consistent with the results we have in the toy example. However, the best rank for tensor decomposition is an NP-hard problem [35], which means there is no algorithm for determining the best tensor rank. Hence in practice, the rank can be estimated by fitting different numbers of rank-1 tensors by the CPD model.

Lastly, since now Fourier-coefficient matrix with lower rank is of size $K \times R$, we can test another time for only sampling $K \times R$ points.

| Sampling strategy: $K \times R$, | K = 55, $\vert\mathscr{F}\vert = 30$ iteration = 600 | | |
|---|---|---|---|
| R | function error | integration error | CPU time(sec) |
| R = K= 55 | 0.001755357 | 0.005539071 | 23.58637767 |
| 45 | 0.001799183 | 0.010366755 | 21.14877568 |
| 40 | 0.001855471 | 0.012386979 | 20.49000502 |
| 35 | 0.002761207 | 0.018168613 | 18.86270292 |
| 30 | 0.003582007 | 0.022662432 | 18.32986945 |
| 25 | 0.006448219 | 0.040881785 | 17.31500931 |
| 20 | 0.013507681 | 0.09761292 | 15.82239724 |
| 15 | 0.048813471 | 0.619796172 | 12.96152817 |
| 10 | 0.071886444 | 0.60369236 | 11.59706865 |

Table 4.5: Implementation results with $55 \times R$ samplings and 1000 iterations for $|\mathscr{F}| = 30$.

Compared with Table 4.4, the new sampling strategy in Table 4.5 indeed reduces the time cost without losing the corresponding function accuracy but the integration error increases compared to the same rank level in Table 4.4.

From all these tables, we can see the function match accuracy given by SGD is at most $10^{-3}$, which is undesirable, and we can also see that the integration accuracy is unstable and irregular, which is caused by the SGD solver. This will become clear in the next chapter.

It's meaningful that we make attempts in speeding up the algorithm, which helps us to understand it better. However, we also know that it is not the best solution in essence. Figure 4.5 visualizes the results achieved by SGD, which indicates that the SGD solution does not fit the prototype function very well. This inaccuracy can influence the lower-rank analysis greatly since the error in estimating the lower-rank tensors is still dominant. Hence, in the next section, we delve into the SGD algorithm itself to have more insights.

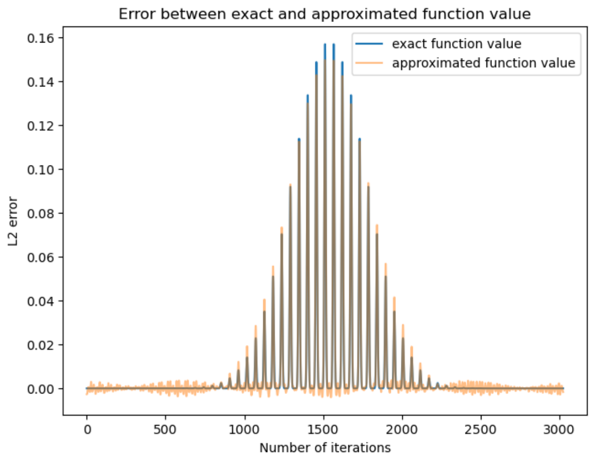


Figure 4.5: Comparison between exact function values and approximated function values obtained by CPD-SGD algorithm.

### 4.2.3. INSIGHTS INTO SGD

Although SGD is quite popular for solving optimization problems in machine learning, especially for multivariate functions, it still has apparent deficiencies. Recall that SGD is derived from the Gradient Descent method. Let us see why it is effective under proper circumstances.

The loss function Eq (4.4) is in the quadratic form, which can be generalized as Eq (4.9), where $\mathbf{A}_n$ is solved as $x$:

$$f(x) = \frac{1}{2}x^T\mathbf{A}x - b^T x + c \tag{4.9}$$

The minimum of $f(x)$ is also the solution to the linear equation $\mathbf{A}x = b$ if $\mathbf{A}$ is symmetric positive definite (SPD), because $f'(x) = 0$ is equivalent with $\mathbf{A}x = b$, and the SPD property of $\mathbf{A}$ ensures the stationary point is the global minimum. Figure 4.6 shows an example of a quadratic function in which **A** is SPD, and the lowest line of the surface is the solution of $\mathbf{A}x = b$.

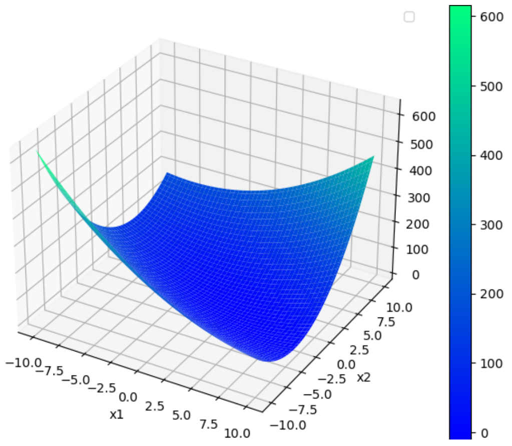


Figure 4.6: The 3D image of a quadratic function with **A** symmetric positive definite, the solution is the lowest part of this convex surface.

However, in practice, **A** can be more general with irregular shapes, and GD is likely to be trapped in the local minimum since the gradient is also small there. In fact, SGD introduces the possibility to escape from the local minimum since the gradient computation is based on stochastically chosen batches, but still has the limitation of GD itself.

There are two main aspects of SGD: search direction and step size. Hence we would like to go a step further to know more about both.

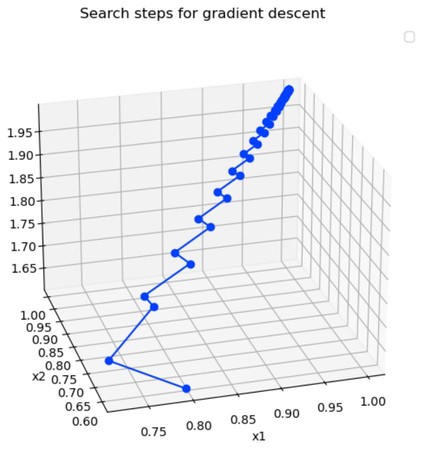


Figure 4.7: The search process of Gradient Descent method.

On the one hand, such a negative gradient may not be the optimal search direction since it is just a first-order approximation by tangent lines in the local area, which limits global accuracy and has comparatively poor robustness. For a general quadratic function, if the condition number $\kappa$(ratio of maximum eigenvalue to minimum eigenvalue) of **A** is big, then it would lead to unnecessarily tortuous searching steps, as shown in Figure 4.7.

A better approximation for search direction could be the second order derivative with the Hessian matrix, which leads to the Newton method or quasi-Newton method. There are also other methods adapting the search direction [36]. But second-order methods often face serious problems such as heavy dependence on the initial guess.

On the other hand, a commonly used strategy for choosing the step size for SGD is fixing a prescribed $\alpha$. However, it is difficult to pick a good value, not only for each descent step, but also in the initial step since individual steps are not checked for robustness at the time they are taken. Actually many methods of adapting search directions also have such high sensitivity on the step size, such as the celebrated Adagrad algorithm [37].

To conclude, the model needs to be modified for a better performance. And based on the analysis above, we deicide to use the Conjugate Gradient (CG) method to overcome the problem existing in search directions and step size.

# 5

# Our 1st solution: CPD with Conjugate Gradient (CG) method

## 5.1. A further step: CG solver

We mentioned at the end of the last chapter that there are different ways to improve SGD, a famous one that is commonly used in Neural Networks is to add momentum to the gradient. This physics idea helps the descent process escape from the local minimum or plateau. And the authors of [38] point out that the momentum method is a time-invariant version of the Conjugate Gradient (CG) method. Although there are lots of papers analyzing choices for step sizes and momentum, applying the named 'optimal parameter' tuned for the momentum algorithm is just the same as using the CG method.

In this chapter, we replace SGD with CG since the latter uses conjugate search directions with adaptive step sizes computed by exact line search, which may help us to find the solution more accurately and faster. The result achieved by CG is indeed promising, compared to the 'Gibb's phenomenon' in Figure 4.5, Figure 5.1 shows a more precise fitting. Moreover, it even needs much less time to obtain such good matching.

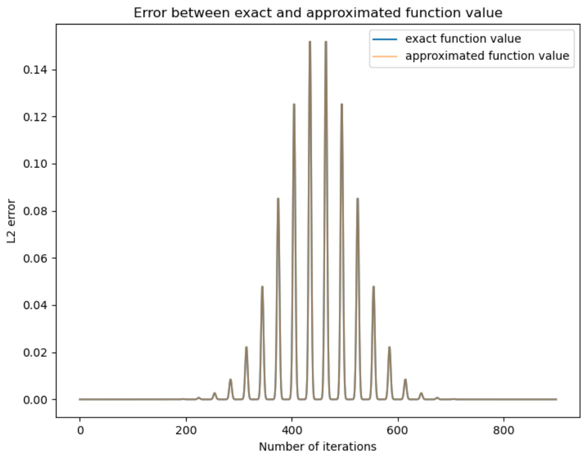


Figure 5.1: The match results of Conjugate Gradient method.

### 5.1.1. Analysis about the standard CG

Figure 4.7 shows that there are many repetitive search steps in the GD method, which means the corresponding iterations do not completely eliminate the component of the error vector in this direction. CG manages to prevent this problem by requiring conjugacy among search directions.

#### Conjugate direction

We first introduce the conjugate direction assuming **A** is symmetric positive definite, again for the linear equation

$$\mathbf{A}x = b \tag{5.1}$$

using the iterative method, i.e. :

$$x_{k+1} = x_k + \alpha_k \cdot d_k \tag{5.2}$$

where $x_k$ is the current approximation, $\alpha_k$ is the next step size, and $d_k$ is the next move direction. Then we can derive the following facts from Eq (5.2) and Eq (5.1):

| | |
|---|---|
| Error vector: | $e_k = x_k - x$ |
| Residual vector: | $r_k = b - \mathbf{A}x_k$ |
| And: | $\mathbf{A}e_k = r_k$ |

If the error vector $e_{k+1}$ is always orthogonal to the search direction $d_k$, i.e.

$$d_k^T \cdot e_{k+1} = 0 \tag{5.3}$$

which avoids searching in this direction again later, and also provides the solution to the step size. The step size $\alpha_k$ can be computed by substituting Eq (5.3) into Eq (5.2):

$$\alpha_k = \frac{d_k^T e_k}{d_k^T d_k} \tag{5.4}$$

However, it is almost impossible to compute $\alpha_k$ using Eq (5.4) as the exact solution $x$ is unknown, thus $e_k$ is not determined. Instead, we use '**A**-orthogonal' to replace 'orthogonal', which also means conjugate:

$$d_k^T \mathbf{A} e_{k+1} = 0 \tag{5.5}$$

This is also the definition of **conjugate directions**: $d_i$ and $d_j$ are conjugate to each other if $d_i^T \mathbf{A} d_j = 0, i \neq j$. And note that $r_{k+1} = \mathbf{A} e_{k+1}$, we can rewrite Eq(5.5) as:

$$d_k^T r_{k+1} = 0 \tag{5.6}$$

Now we are able to derive the Conjugate Gradient (CG) method. The CG is an approach to realizing conjugate directions with the help of gradient information.

Recall we have mentioned that for GD method the gradient $f'(x)$ is exactly the residual, where $f'(x) = \mathbf{A}x - b$. So for CG, starting from an initial condition $x_0$, search the directions that are conjugate to each other with the line search determining the step size, then the gradient at $x_{k+1}$, i.e. $g_{k+1}$, is orthogonal to all the previous search directions $d_1, d_2, d_3, ..., d_{k-1}$, which spans the Krylov subspace step by step (The proof can be found in [29]). That is:

$$g_{k+1}^T d_j = 0, \; j = 0, ..., k \tag{5.7}$$

Next, we give the derivation of the search direction and the step size for the CG method.

**Search direction**

Following the requirement that all the search directions are conjugate to each other, and new search direction $d_{k+1}$ is the linear combination of $g_k$ and $d_k$, then $d_{k+1}$ can also be expressed in an iterative way:

$$d_{k+1} = -g_{k+1} + \beta_k d_k \tag{5.8}$$

Then the only task left is to determine $\beta_k$ since the first search direction $d_0$ is given by $g_0$. According to the conjugate direction definition, i.e. $d_k^T \mathbf{A} d_{k+1} = 0$, we can multiply both sides with $d_k^T \mathbf{A}$ then we have:

$$\beta_k = \frac{g_{k+1}^T \mathbf{A} d_k}{d_k^T \mathbf{A} d_k} \tag{5.9}$$

Moreover , since $g_{k+1} - g_k = \mathbf{A}(x_{k+1} - x_k) = \alpha_k \mathbf{A} d_k$, Eq (5.9) can be rewritten as :

$$\beta_k = \frac{g_{k+1}^T (g_{k+1} - g_k)}{d_k^T (g_{k+1} - g_k)} \tag{5.10}$$

We can modify this expression further to ensure there is only gradient information in $\beta$. Transposing Eq (5.8) and multiplying $g_{k+2}$ on both sides we have:

$$d_{k+1}^T g_{k+2} = -g_{k+1}^T g_{k+2}^T + \beta_k d_k^T g_{k+2} \tag{5.11}$$

According to the fact that the new gradient is orthogonal to all the previous search directions, we have:

$$d_{k+1}^T g_{k+2} = d_k^T g_{k+2} = 0 \tag{5.12}$$

This also means that in the CG method, the gradients obtained from two adjacent searches are orthogonal to each other. In the end, we have the following expression for search directions, which is also known as Fletcher Reeves formula [39].

$$\beta_k = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k} \tag{5.13}$$

**Step size**

Now we can easily compute the value of step size based on the equations we derived above. Reuse $g_{k+1} - g_k = \mathbf{A}(x_{k+1} - x_k) = \alpha_k \mathbf{A} d_k$, we multiply $d_k^T$ to the both sides to have:

$$\alpha_k = \frac{d_k^T (g_{k+1} - g_k)}{d_k^T \mathbf{A} d_k} = \frac{g_k^T g_k}{d_k^T \mathbf{A} d_k} \tag{5.14}$$

It is efficient and convenient to update the approximations using the CG method to solve linear equations when $\mathbf{A}$ is symmetric positive definite, but Eq (5.13) also makes it possible to apply the CG method in more general practices. Nevertheless, it still needs some modification to completely fit our problem. Since in our case, the solution is in the form of matrices $\in \mathbb{R}^{K\times R}$, not the standard vector $x \in \mathbb{R}^n$.

### 5.1.2. Adjust CG to fit matrix equations

The minimization problem Eq (4.4) is in fact not a standard form of a linear equation $\mathbf{A}x = b$, whereby the solution $x$ should be an $n$-length vector, and $\mathbf{A} \in \mathbb{R}^{n\times n}$. Eq (4.4) can be seen as a matrix equation and we derive the matrix equation formula for the CG method by unfolding the matrix into vectors, as we introduced in Chapter 3. This is indeed a common-used tool in tensor decomposition. For a $K \times R$ factor matrix $\mathbf{A}_n$, define

$$\text{vec}(\mathbf{A}_n) = \begin{bmatrix} \mathbf{A}_n(:,1) \\ \mathbf{A}_n(:,2) \\ \mathbf{A}_n(:,3) \\ \vdots \\ \mathbf{A}_n(:,R) \end{bmatrix}_{KR\times 1} \tag{5.15}$$

which is equivalent to stacking all the columns of $\mathbf{A}_n$ vertically in order. Next, recall the Kronecker Product that we introduced in Chapter 3, denoting the column of matrices $\mathbf{A}$ and $\mathbf{B}$ by $\mathbf{a}$ and $\mathbf{b}$, there is a helpful property that $\mathbf{b}^T \otimes \mathbf{a} = \mathbf{a}\mathbf{b}^T$. Applying the definition of vec(·), it follows that $\text{vec}(\mathbf{a}\mathbf{b}^T) = \mathbf{b} \otimes \mathbf{a}$.

Consider the matrix product $\mathbf{V}_n[:,m]^T\mathbf{A}_n\mathbf{Q}_n[:,m]$ in Eq (4.5), where $\mathbf{V}_n[:,m]^T \in \mathbb{R}^{1\times K}$, $\mathbf{A}_n \in \mathbb{R}^{K\times R}$, $\mathbf{Q}_n[:,m] \in \mathbb{R}^{R\times 1}$, and

$$\mathbf{V}_n[:,m]^T\mathbf{A}_n\mathbf{Q}_n[:,m] = \sum_{k=1}^{K}\sum_{r=1}^{R}(\mathbf{V}_n[:,m]^T)(:,k)\mathbf{A}_n(k,r)\mathbf{Q}_n[:,m](:,r). \tag{5.16}$$

Applying $\text{vec}(\mathbf{ab}^T) = \mathbf{b}\otimes\mathbf{a}$ and the linearity of the $\text{vec}(\cdot)$ operator, we have:

$$\begin{aligned}\text{vec}(\mathbf{V}_n[:,m]^T\mathbf{A}_n\mathbf{Q}_n[:,m]) &= \sum_{k=1}^{K}\sum_{r=1}^{R}\mathbf{A}_n(k,r)\mathbf{Q}_n[:,m](:,r)^T\otimes(\mathbf{V}_n[:,m]^T)(:,k)\\ &= (\mathbf{Q}_n[:,m]^T\otimes\mathbf{V}_n[:,m]^T)\text{vec}(\mathbf{A}_n)\end{aligned} \tag{5.17}$$

where $(\mathbf{Q}_n[:,m]^T\otimes\mathbf{V}_n[:,m]^T)$ is of size $(1\cdot 1\times K\cdot R)$. In this way, we represent Eq(4.4) in a more familiar form, which is

$$\min_{\mathbf{A}_n}||y_m - (\mathbf{Q}_n[:,m]^T\otimes\mathbf{V}_n[:,m]^T)\text{vec}(\mathbf{A}_n)||_2^2 \tag{5.18}$$

That is, we have generalized the matrix equation to the normal form $\mathbf{A}x = b$, and $\text{vec}(\mathbf{A}_n)$ is the $x\in\mathbb{R}^{KR}$ that we try to solve, and $(\mathbf{Q}_n[:,m]^T\otimes\mathbf{V}_n[:,m]^T)$ is $\mathbf{A}$.

Write the matrix equation in a generalized form as:

$$f(\mathbf{A},\mathbf{X}) = \mathbf{B} \tag{5.19}$$

And now we can give the algorithm of vectorized CG and the modified CPD-CG algorithm:

**Algorithm 4** Vectorized CG

```
Initialize: $\mathbf{X_0},\mathbf{g}_0 = \text{vec}(\mathbf{B}) - f(\mathbf{A},\mathbf{X}_0), \mathbf{d}_0 = \mathbf{g}_0,\ \epsilon > 0$
for k=0 to ... do
    $\alpha_k = \frac{\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{d}_k^T f(\mathbf{A},\mathbf{D}_k)}$, where $\mathbf{D}_k$ recovers $\mathbf{d}_k$ back to matrix.
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{d}_k$
    $\mathbf{r}_{k+1} = \mathbf{r}_k + f(\mathbf{A},\mathbf{D}_k)$
    if $||\mathbf{r}_{k+1}||/||\text{vec}(\mathbf{B})|| < \epsilon$ then
        stop
    else
        $\beta_k = \frac{\mathbf{g}_{k+1}^T\mathbf{g}_{k+1}}{\mathbf{g}_k^T\mathbf{g}_k}$
        $\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k\mathbf{d}_k$
    end if
end for
```

Theoretically, CG is able to converge within at most $KR$ steps if '$\mathbf{A}$' here is positive definite, for the optimal $\text{vec}(\mathbf{A}_n)^* \in \mathbb{R}^{KR}$. However, as it is derived from different $m$, i.e., each sample point has a corresponding '$\mathbf{A}$', it is impossible to ensure every '$\mathbf{A}$' is positive

**Algorithm 5** CPD-CG

```
Input: $\mathbf{X_0}, \mathbf{y}, R, K, \epsilon_{\text{int}} > 0$, integration error = 1
while Integration error > $\epsilon_{\text{int}}$ do
    Initialize $\{\mathbf{A}_n \in \mathbb{R}^{K\times R}\}_{n=1}^{N}$
    for n=1 to N do
        Update $\mathbf{A}_n$ and $\mathbf{Q}_n$ via vectorized CG
        if The iteration number reaches $K \times R$ then
            stop
        end if
    end for
    Reconstruct the coefficient tensor by $\mathbf{A}_n$.
    Compute the integration by coefficient tensor.
    Update the integration error.
end while
```

definite. Though the standard requirement for the quadratic coefficient matrix may not be satisfied, our experiments still show that CG finally converges to an ideal minimum.

Note that in the last chapter we choose the number of sampling points for each dimension to be the same as $K$ intuitively, and it's hard to check this sampling strategy efficiently because the model error is dominant. However, since the CG method provides much better performance on the accuracy, we are able to explore the sampling strategy now.

Here we fix $K$ as 30 to suppress the Fourier series truncation error. Next, as the cosine basis functions are represented discretely by mesh points, also known as training data, there can be a risk that the solution does not cover the non-training data, i.e. over-fitting. So we test a range of numbers of sampling points of each dimension, $T$, to see the fitting effect for both training data and non-training data.

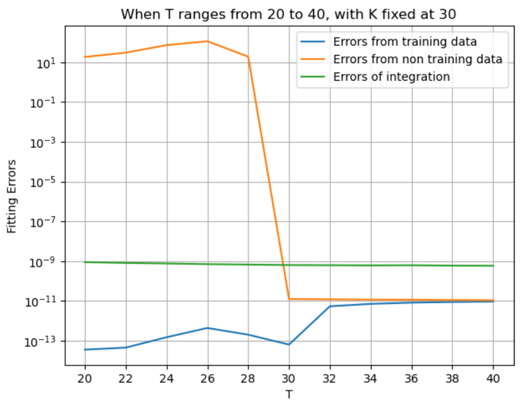


Figure 5.2: Fitting results for different numbers of the mesh point of each dimension.

It is obvious that there are 2 different fitting behaviors for training data and non-training data when $T$ is greater than $K$ and $T$ is smaller than $K$, while the integration error is always consistent with the accuracy obtained by training data. As we visualize the function plots via Figure 5.3 and Figure 5.4, they present that when the sampling number T is less than $K$, then it is not sufficient to reconstruct a correct probability density function, as the computer rather stops early at the pointwise fitting level than looking for a completely correct curve. So it is necessary to feed the solver with more training points, at least as many as $K$. so that a global solution can be found as in Figure 5.4, and the minimum of $T$ can be determined to be the same as $K$ as shown in Figure 5.2.

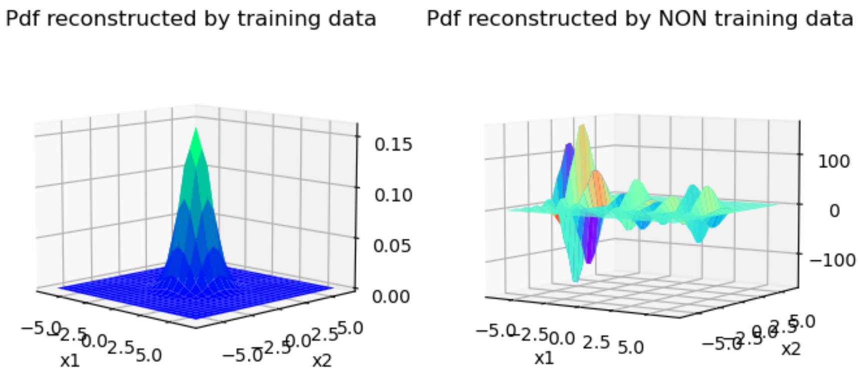


Figure 5.3: Pdf reconstructed by Fourier-cosine series expansion when T = 25, K= 30.

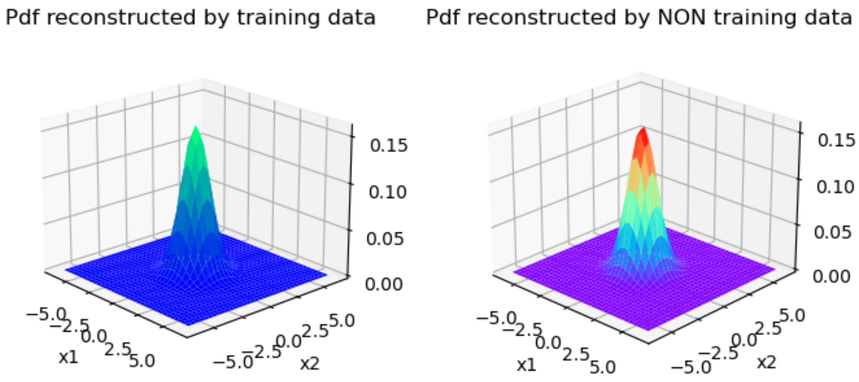


Figure 5.4: Pdf reconstructed by Fourier-cosine series expansion when T = 35, K= 30.

### 5.1.3. Results achieved by CG

For the CG method, there are 2 layers of stopping criteria. The out layer limits the maximum number of iterations, which is the size of the vectorized solution, $K \times R$, while the inner layer is used to stop the iteration when there is little update for each factor matrix, which is also when it converges. For the ALS part, the stopping criterion can be set as little progress in updating the integration. However, as the integral of the example function, Gaussian pdf, is exactly one, we can also measure the distance between the approximated integral and one. So in our tests, ALS stops when the distance is smaller than the prescribed accuracy, which is $10^{-8}$.

Starting from testing CPD-CG for a two-dimensional Gaussian pdf, the number of points sampled for each dimension of the training dataset is determined by the conclusion before, with $T = K = 30$. And CG stops while the improvement for each factor matrix is less than $10^{-10}$. After all the factor matrices converge, the integral is computed by the coefficient tensor that is synthesized by these factor matrices, and then it is compared with 1 to check whether the integration accuracy criterion is met. The following plot shows that under such stopping criteria this algorithm is able to find an accurate solution for each rank.

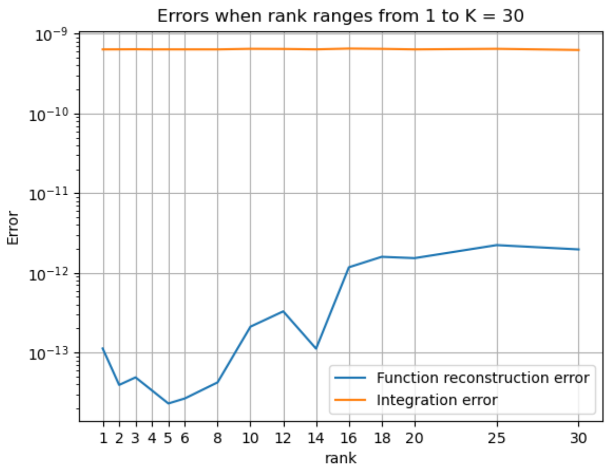
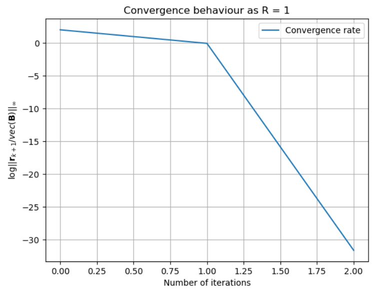
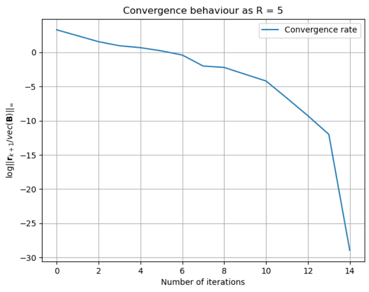


Figure 5.5: Function error and Integration error under different ranks.

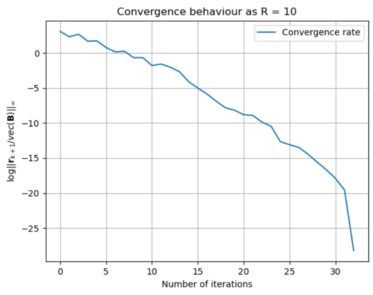
Next, we visualize the convergence behavior when the rank of $\mathbf{A}_1$ = 1, 5, 10,15. The standard CG method, which solves linear equation $\mathbf{A}x = b$, with $\mathbf{A}$ being SPD, has a linear convergence behaviour, or even superlinear convergence. In our application, the convergence rate can be seen as an approximation of linear convergence or superlinear convergence.



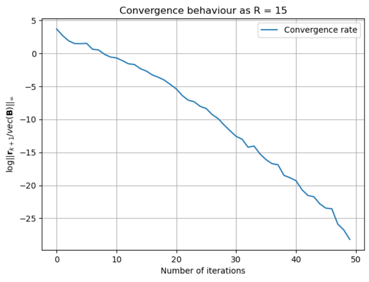
(a) Convergence rate of rank = 1



(b) Convergence rate of rank = 5



(c) Convergence rate of rank = 10



(d) Convergence rate of rank = 15

Figure 5.6: Convergent behaviours when rank R varies.

The CPD-CG algorithm can not only find the solution every time, but also solve the integration much faster than the SGD method, especially for very low-rank factor matrices, which only takes around 0.003s. Indeed, CG needs more iteration times as the number of ranks grows. Hence, the computational time also increases. For 2D Gaussian pdf, it costs more iterations for the first factor matrix, $\mathbf{A}_1$ when the rank is higher, and $\mathbf{A}_2$ converges rapidly after $\mathbf{A}_1$ is ready. However, both of them never achieve the maximum iteration number $K \times R$, which is labeled in the yellow bars in Figure 5.7. The total iteration numbers (after all factor matrices are iterated to convergence) for each rank are presented in Figure 5.8, including the specific CPU time.

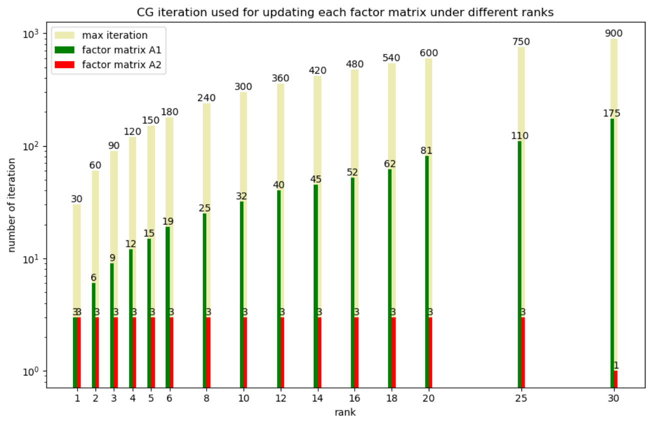


Figure 5.7: Iterations used for updating factor matrices, while the maximum of iteration is determined by $K \times R$.

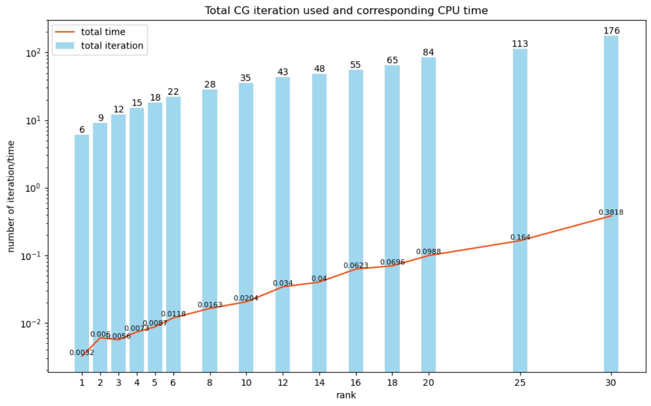


Figure 5.8: Total iterations used for each rank and the corresponding time consumed.

In general, though each rank gives a desirable solution for the factor matrices, we need to stick to the lower-rank representation goal. And following the results of our experiments, the performance of rank-one factor matrices is good enough. In this way, we decrease the computing complexity from $O(K^2)$ to at least $O(K \times 1 \times 6)$ for the coefficient tensor, for there are only 6 CG iterations, which is indeed a great reduction for the computational cost.

## 5.2. 3D COUNTERPART MODEL

In the previous sections, we discuss different methods of reducing the computational complexity for computing the coefficient matrix for Fourier-cosine series expansion in 2D. However, to really solve the high-dimensional problem, we at least need to move on to 3D. In this section, we turn to 3D space and implement the CPD-CG algorithm to see its efficiency in a higher dimensional situation.

The Fourier-cosine series expansion in 3D can be generated in a similar way as in 2D, which also needs a sufficient integration truncation range $[a_1,b_1]\times[a_2,b_2]\times[a_3,b_3]\subset\mathbb{R}^3$.

$$f(\mathbf{x}) = {\sum}'^{K}_{k_1=0}{\sum}'^{K}_{k_2=0}{\sum}'^{K}_{k_3=0}\mathbf{A}_{k_1,k_2,k_3}(\mathbf{x})\cos\left(k_1\pi\frac{x_1-a_1}{b_1-a_1}\right)\cos\left(k_2\pi\frac{x_2-a_2}{b_2-a_2}\right)\cos\left(k_3\pi\frac{x_3-a_3}{b_3-a_3}\right) \tag{5.20}$$

And the coefficient tensor is organized as a cube in 3D, which can be computed as Eq( 5.21), and again after the exchange of summation and integration, the key behind is $\mathbf{A}_{000}$.

$$\mathbf{A}_{k_1,k_2,k_3}(\mathbf{x}) \equiv \left(\prod_{i=1}^{3}\frac{2}{b_i-a_i}\right)\int_{a_1}^{b_1}\int_{a_2}^{b_2}\int_{a_3}^{b_3} f(\mathbf{x})\cos\left(k_1\pi\frac{x_1-a_1}{b_1-a_1}\right)\times \cos\left(k_2\pi\frac{x_2-a_2}{b_2-a_2}\right)\cos\left(k_3\pi\frac{x_3-a_3}{b_3-a_3}\right)d\mathbf{x} \tag{5.21}$$

Firstly we give the convergence curve of the 3D Fourier-cosine series truncation terms, to determine the value of $K$ that we use in 3D tests.

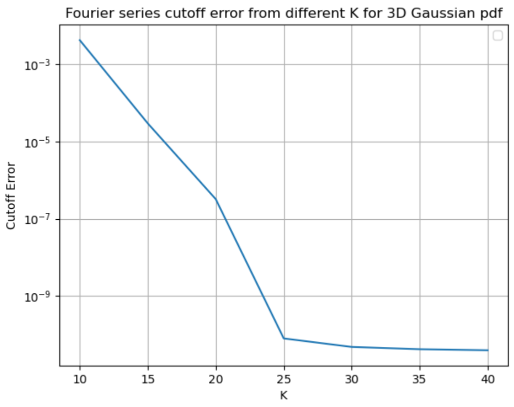


Figure 5.9: Convergence of series truncation error for Fourier-cosine series expansion as K increases.

This supports that choosing $K$ to be 30 is also safe for 3D Fourier-cosine series expansion for our testing function, i.e. Gaussian density. And the following plot verifies the same sampling conclusion drawn from 2D: the error is global when $T$ is equal to or larger than $K$.

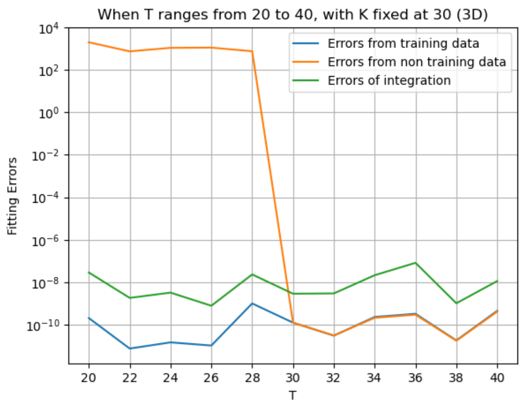


Figure 5.10: Fitting results for different sizes of the training set.

Next, we continue searching for the best experimental rank by looking at their corresponding integration accuracy. We test a range of ranks from 1 to 10, as shown in Figure 5.11. The accuracy is quite good enough, so it is not necessary to raise the rank to a very high level.

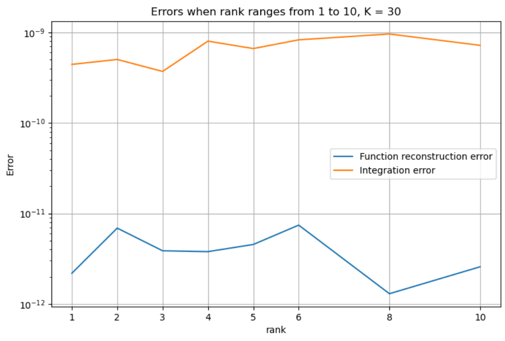


Figure 5.11: Function error and Integration error under different ranks.

We also give the number of iterations for each factor matrix as it satisfies the stopping criterion. Again none of the factor matrices actually iterates to $K \times R$ times. The specific time costs are shown in Figure 5.13, which indicates that rank-one factor matrices are again the fastest fitting with high accuracy for our testing function, i.e. Gaussian density. And the computational complexity is reduced from $O(K^3)$ to $O(K \times 1 \times 10)$ regarding the coefficient tensor.

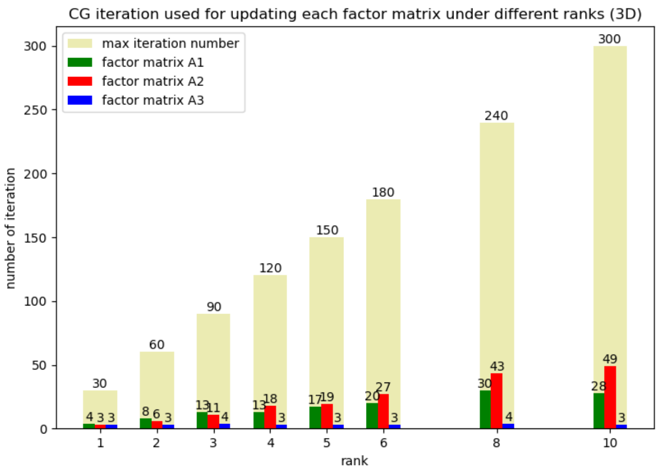


Figure 5.12: Function error and Integration error under different ranks.

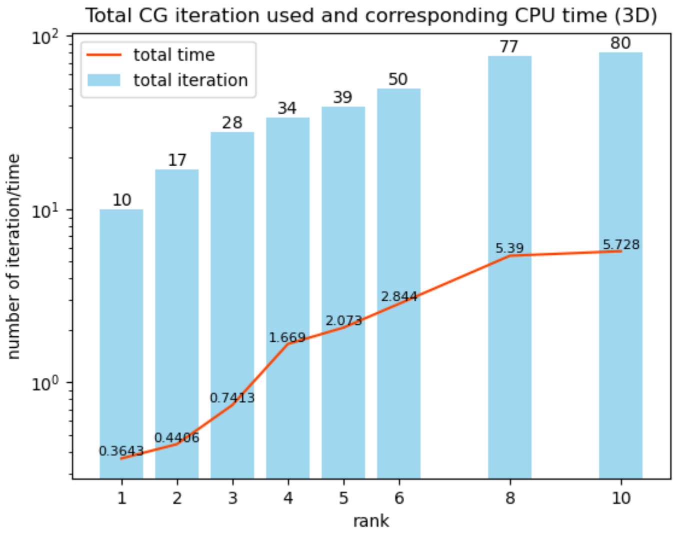


Figure 5.13: Function error and Integration error under different ranks.

### 5.2.1. Application to compute expectations

As this new method has a good performance in computing the integration of the smooth Gaussian pdf, we continue to apply it to compute the expectation of some general functions whose variables are normally distributed. Given the formula of function expectation:

$$\mathbb{E}(g(x)) = \int_{a_1}^{b_1} \int_{a_2}^{b_2} g(\mathbf{x}) f_X(\mathbf{x}) dx_1 dx_2 \tag{5.22}$$

we compute the expectation of an example function $g(\mathbf{x}) = \frac{e^{x_1}+e^{x_2}}{2}$, where $(x_1, x_2) \sim N(\mathbf{0}, \mathbf{1})$. This testing function is chosen because it represents the payoff function of a basket option in the field of quantitative finance.

The analytical expectation of $g(x)$ can be simply computed by Moment Generating Function. Since for normally distributed $x$, $M_X(t) = \mathbb{E}(e^{tX}) = e^{t\mu + \frac{t^2}{2}\sigma^2}$, and here $t = 1$, $\mathbb{E}(g(\mathbf{x})) = e^{\frac{1}{2}}$.

Next, we compute with applying its expectation by the CPD-CG integration method and compare this numerical solution with the analytical solution as the integration error in Table 5.1, from which we can see it is sufficient and fast to compute the integration when rank = 1.

K = 30, T = 30, 2D

| rank | expectation error | CPU time(sec) |
|---|---|---|
| 1 | 9.875774531487025e-08 | 0.004648 |
| 2 | 9.86047701267978e-08 | 0.005195 |
| 3 | 9.861229099961122e-08 | 0.008275 |
| 4 | 9.861528904586692e-08 | 0.008991 |
| 5 | 9.860876226674975e-08 | 0.010080 |

Table 5.1: Accuracy and time cost of computing expectation of $\frac{e^{x_1}+e^{x_2}}{2}$ by CPD-CG method.

Samely, for $g(\mathbf{x}) = \frac{e^{x_1}+e^{x_2}+e^{x_3}}{3}$, where $\mathbf{x} \sim N(\mathbf{0}, \mathbf{1})$ in 3D, we summarize the results in Table 5.2. The table shows the algorithm needs a bigger rank to give an accurate result in 3D than in 2D, but the iteration number does not grow monotonically with rank, which again indicates the real rank of tensors is NP-hard and needs to be searched by decomposition algorithms.

K = 30, T = 30, 3D

| rank | expectation error | CPU time(sec) |
|---|---|---|
| 1 | 0.0030854176068879635 | 0.81815 |
| 2 | 0.0006333410086563074 | 2.21082 |
| 3 | 3.550936298335472e-06 | 4.19175 |
| 4 | 9.908346121356715e-08 | 2.38794 |
| 5 | 9.91124169402724e-08 | 3.71954 |

Table 5.2: Accuracy and time cost of computing expectation of $\frac{e^{x_1}+e^{x_2}+e^{x_3}}{3}$ by CPD-CG method.

# 6 Our 2nd solution: Principal-COS method

In the previous chapters, we make efforts to find a lower-rank representation of a multi-dimensional tenor containing Fourier-cosine coefficients. The method of HTF with machine learning gives the insight that a numerical integration problem can be transformed into another one that can be learned by machine, via applying Fourier-cosine series expansion on the integrand. A further analysis on the Fourier-cosine series coefficients of the example function we used before, i.e. the Gaussian density function, reveals that only a fraction of the coefficients are significantly different from zero, which we name as principal cosine terms from here onwards. This observation inspires us to only use the principal cosine basis functions to generate a smaller coefficient hyper-cube in high dimensions.

Based on this idea, we developed a second solution method which is designed to compute the expectations of functions of several random variables. Recall that expectations are mathematically defined as multi-dimensional integrations, of which integrands are the multiplications of the target functions and the joint density functions of relevant random variables. This solution method works as follows: instead of representing the joint density function with normal $K^N$ Fourier-cosine basis functions, whereby $N$ is the number of total dimensions, we only use the 'principal' cosine basis functions to reconstruct the joint density function. Here the 'principal' terms are referred to those cosine basis functions of which the Fourier coefficients are greater than a pre-defined threshold. We further take the assumption that the non-principal terms of marginal distributions do not become principal terms in the Fourier expansion of the joint density function.

More precisely, we first compute the Fourier coefficients of the marginal distributions of the involved random variables by the COS method [33], then select the 'principal' basis functions of which the Fourier-cosine expansion coefficients are larger than a certain threshold, and at last those principal basis functions are used in the reconstruction of multi-dimensional Fourier-cosine series expansion of the joint density function. Hence, this way we effectively utilize much smaller-sized vectors that contain the principal cosine basis functions and a much dense tensor (hyper-cube) consists of the Fourier-cosine coefficients. We compute these coefficients by the high-dimensional COS method [8], and thus, we name this innovative idea as the Principal-COS method. Now getting back to the goal of calculating the expectation of a function defined for a few random variables, we then follow the same derivation as in [33]. That is, by inserting this modified reconstruction of the joint density function into the integration that defines the expectation, and interchanging the order of integration and the summation, we effectively transform the inner product of the target function and the joint density function into the inner product of their Fourier-cosine coefficients. Since the Fourier-cosine coefficients of the joint density can be accurately approximated via the COS method, what is remaining is to compute the Fourier-cosine coefficients of the target function.

Here we already give some justification of the assumption made in this method, namely, why we can assume that the 'non-principal' terms as identified in the construction of the marginal distributions can be deemed non-important in the construction of the joint density function. Those 'unimportant' cosine basis functions usually have coefficients less than $10^{-10}$, which means such coefficients can greatly dampen the corresponding Fourier-cosine basis terms of the target function via the inner product of the two sets of coefficients. As a result, we reduce the number of cosine basis functions not only for the joint density function, but also for the function of random variables, which leads to smaller coefficient tensors. The multi-dimensional Fourier coefficients of general functions are again computed by Clenshaw-Curtis quadrature rule, although in many cases analytical solution might exist.

## 6.1. Reconstruction for Gaussian PDF

We still implement conceptual tests with the two-dimensional Gaussian probability density function. Firstly, we compute the Fourier-cosine coefficients for the one-dimensional marginal distribution of Gaussian density by the COS method:

$$f(x) = {\sum}'^{\infty}_{k=0} A_k \cdot \cos\left(k\pi \frac{x-a}{b-a}\right) \tag{6.1}$$

$$A_k = \frac{2}{b-a} \operatorname{Re}\left\{\varphi\left(\frac{k\pi}{b-a}\right) \cdot \exp\left(-i\frac{ka\pi}{b-a}\right)\right\} \tag{6.2}$$

where $(a, b)$ is the integration truncation interval, $A_k$ denotes the vector of Fourier-cosine coefficients, and $\varphi(\cdot)$ is the characteristic function of random variables. Re$(\cdot)$ means only taking the real part of the complex input.

Then we can select the 'principal' Fourier-cosine coefficients that are larger than various thresholds, such as $10^{-8}$, $10^{-6}$, etc. Note that after this step the indices of cosine basis functions are not continuous anymore, since those insignificant ones are filtered out. Next, we can use these selected basis functions to generate the Fourier coefficients of the original Gaussian pdf, utilizing the COS method.

As a result, we replace the full-sized Fourier coefficient tensor with a smaller principal coefficient tensor, and thus, the computational complexity is reduced.

As we only use marginal distribution functions as one-dimensional representations for high-dimensional joint density functions, we tested three sampling strategies to check whether the sampling method matters for the correlation.

### I. Uniform sampling

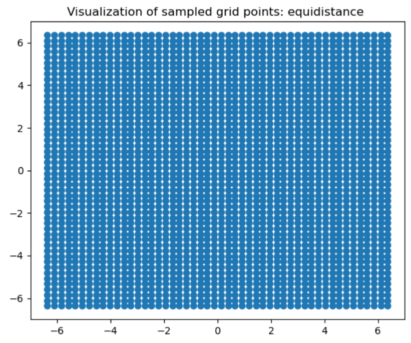


Figure 6.1: The uniform sampling strategy with equidistant mesh grids

| $\mu_{2D} = [1,1]$, $\Sigma_{2D} = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$, correlation = 0, Nr. of series truncation terms K = 50. | | |
|---|---|---|
| ‘principal’ threshold | Number of principal terms | $L_\infty$ error |
| 1.00E-01 | (2,2) out of (50,50) | 0.111738992 |
| 1.00E-02 | (5,5) out of (50,50) | 0.007782515 |
| 1.00E-03 | (7,7) out of (50,50) | 0.000376692 |
| 1.00E-04 | (8,8) out of (50,50) | 5.83E-05 |
| 1.00E-08 | (12,12) out of (50,50) | 3.10E-09 |
| 1.00E-12 | (15,15) out of (50,50) | 1.21E-10 |

Table 6.1: Select cosine basis functions from marginal distributions $N(1,1)$ and $N(1,1)$ to reconstruct joint distribution $N(\mu_{2D}, \Sigma_{2D})$ when the correlation is 0. $L_\infty$ error denotes the error between exact 2D distribution values and values approximated by principal Fourier-cosine basis functions.

| $\mu_{2D} = [1,1]$, $\Sigma_{2D} = \begin{bmatrix} 1.0 & 0.05 \\ 0.05 & 1.0 \end{bmatrix}$, correlation = 0.05, Nr. of series truncation terms K = 50. | | |
|---|---|---|
| ‘principal’ threshold | Number of principal terms | $L_\infty$ error |
| 1.00E-01 | (2,2) out of (50,50) | 0.11193687 |
| 1.00E-02 | (5,5) out of (50,50) | 0.007845235 |
| 1.00E-03 | (7,7) out of (50,50) | 0.003141096 |
| 1.00E-04 | (8,8) out of (50,50) | 0.002955253 |
| 1.00E-08 | (12,12) out of (50,50) | 0.002932286 |
| 1.00E-12 | (15,15) out of (50,50) | 0.002932285 |
| 1.00E-20 | (29,29) out of (50,50) | 1.78E-07 |

Table 6.2: Select cosine basis functions from marginal distributions $N(1,1)$ and $N(1,1)$ to reconstruct joint distribution $N(\mu_{2D}, \Sigma_{2D})$ when the correlation is 0.05.

| $\mu_{2D} = [1,1]$, $\Sigma_{2D} = \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}$, correlation = 0.05, Nr. of series truncation terms K = 50. | | |
|---|---|---|
| 'principal' threshold | Number of principal terms | $L_\infty$ error |
| 1.00E-01 | (2,2) out of (50,50) | 0.136216191 |
| 1.00E-03 | (7,7) out of (50,50) | 0.034943017 |
| 1.00E-05 | (9,9) out of (50,50) | 0.034809537 |
| 1.00E-10 | (14,14) out of (50,50) | 0.034803765 |
| 1.00E-15 | (17,17) out of (50,50) | 0.034803764 |
| 1.00E-20 | (23,23) out of (50,50) | 1.56E-05 |
| 1.00E-30 | (41,41) out of (50,50) | 2.41E-08 |

Table 6.3: Select cosine basis from marginal distributions $N(1,1)$ and $N(1,1)$ to reconstruct joint distribution $N(\mu_{2D}, \Sigma_{2D})$ when the correlation is 0.5.

Table 6.1 shows when there is no correlation in the joint distribution, the reconstruction is accurate as the integration truncation error ($10^{-10}$) even with only $15^2$ Fourier-cosine basis functions, while the full size is $50^2$, which is a great reduction for the computational cost. However, the error is only at $10^{-7}$ with $29^2$ cosine basis functions in Table 6.2, when a small correlation is engaged. Table 6.3 presents that a larger correlation results in more principal basis functions to reconstruct the correlated joint distribution function. Note that here we didn't present the relation between the number of sampling points $T$ and the ultimate error on non-training data, and we just used the conclusion we drew in Chapter 5 that $T$ should be greater or equal to $K$ to avoid overfitting.

To verify whether the way of sampling the points may have an influence on the reconstruction efficiency, next we continue testing two more different sampling strategies. The first one puts more weights on the central area and, since the numbers of points are assigned according to the weights, generates more points in the central area, which is visualized in Figure 6.2. The weight is computed by the integration value on each equally distant interval. However, this sampling means gives the same error performance as the uniform sampling.

II. More points in the center, fewer toward the boundary

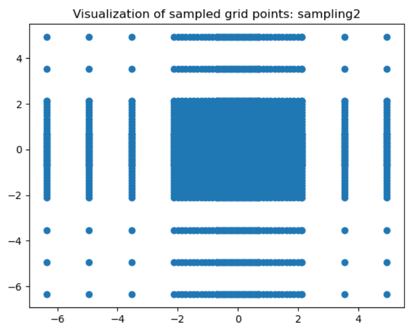


Figure 6.2: More points in the center, fewer points toward the boundary.

| $\mu_{2D} = [1,1]$, $\Sigma_{2D} = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$, correlation = 0, Nr. of series truncation terms K = 50. | | |
|---|---|---|
| 'principal' threshold | Number of principal terms | $L_\infty$ error |
| 1.00E-01 | (2,2) out of (50,50) | 0.111738992 |
| 1.00E-02 | (5,5) out of (50,50) | 0.007782515 |
| 1.00E-03 | (7,7) out of (50,50) | 0.000376692 |
| 1.00E-04 | (8,8) out of (50,50) | 5.83E-05 |
| 1.00E-08 | (12,12) out of (50,50) | 3.10E-09 |
| 1.00E-12 | (15,15) out of (50,50) | 1.21E-10 |

Table 6.4: Select cosine basis from marginal distributions $N(1,1)$ and $N(1,1)$ to reconstruct joint distribution $N(\mu_{2D}, \Sigma_{2D})$ when the correlation is 0.

| $\mu_{2D} = [1,1]$, $\Sigma_{2D} = \begin{bmatrix} 1.0 & 0.05 \\ 0.05 & 1.0 \end{bmatrix}$, correlation = 0.05, Nr. of series truncation terms K = 50. | | |
|---|---|---|
| 'principal' threshold | Number of principal terms | $L_\infty$ error |
| 1.00E-01 | (2,2) out of (50,50) | 0.11193687 |
| 1.00E-02 | (5,5) out of (50,50) | 0.007845235 |
| 1.00E-03 | (7,7) out of (50,50) | 0.003141096 |
| 1.00E-04 | (8,8) out of (50,50) | 0.002955253 |
| 1.00E-08 | (12,12) out of (50,50) | 0.002932286 |
| 1.00E-12 | (15,15) out of (50,50) | 0.002932285 |
| 1.00E-20 | (29,29) out of (50,50) | 1.78E-07 |

Table 6.5: Select cosine basis from marginal distributions $N(1,1)$ and $N(1,1)$ to reconstruct joint distribution $N(\mu_{2D}, \Sigma_{2D})$ when the correlation is 0.05.

| $\mu_{2D} = [1,1]$, $\Sigma_{2D} = \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}$, correlation = 0.05, Nr. of series truncation terms K = 50. | | |
|---|---|---|
| 'principal' threshold | Number of principal terms | $L_\infty$ error |
| 1.00E-01 | (2,2) out of (50,50) | 0.136216191 |
| 1.00E-03 | (7,7) out of (50,50) | 0.034943017 |
| 1.00E-05 | (9,9) out of (50,50) | 0.034809537 |
| 1.00E-10 | (14,14) out of (50,50) | 0.034803765 |
| 1.00E-15 | (17,17) out of (50,50) | 0.034803764 |
| 1.00E-20 | (23,23) out of (50,50) | 1.56E-05 |
| 1.00E-30 | (41,41) out of (50,50) | 2.41E-08 |

Table 6.6: Select cosine basis from marginal distributions $N(1,1)$ and $lN(1,1)$ to reconstruct joint distribution $N(\mu_{2D}, \Sigma_{2D})$ when the correlation is 0.5.

The third sampling method is a reverse of method II by an intuitive inspiration. We use the logarithm of weights computed in sampling strategy II to inverse the weight and multiply a certain scale to control the number of points.

III. Fewer points in the center, more toward the boundary

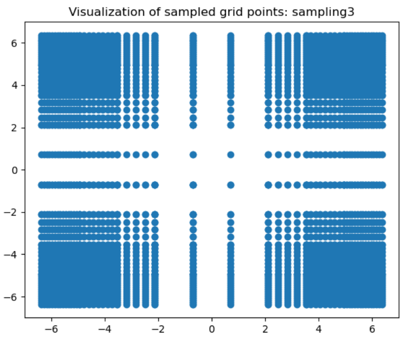


Figure 6.3: Fewer points in the center, more toward the boundary

| $\mu_{2D} = [1,1]$, $\Sigma_{2D} = \begin{bmatrix}1.0 & 0.0\\0.0 & 1.0\end{bmatrix}$, correlation = 0, Nr. of series truncation terms K = 50. | | |
|---|---|---|
| 'principal' threshold | Number of principal terms | $L_\infty$ error |
| 1.00E-01 | (2,2) out of (50,50) | 0.052740431 |
| 1.00E-02 | (5,5) out of (50,50) | 0.003803434 |
| 1.00E-03 | (7,7) out of (50,50) | 0.00023323 |
| 1.00E-04 | (8,8) out of (50,50) | 4.30E-05 |
| 1.00E-08 | (12,12) out of (50,50) | 1.30E-09 |
| 1.00E-12 | (15,15) out of (50,50) | 9.431E-11 |

Table 6.7: Select cosine basis from marginal distribution $N(1,1)$ and $N(1,1)$ to reconstruct joint distribution $N(\mu_{2D},\Sigma_{2D})$ when the correlation is 0.

| $\mu_{2D} = [1,1]$, $\Sigma_{2D} = \begin{bmatrix}1.0 & 0.05\\0.05 & 1.0\end{bmatrix}$, correlation = 0.05, Nr. of series truncation terms K = 50. | | |
|---|---|---|
| 'principal' threshold | Number of principal terms | $L_\infty$ error |
| 1.00E-01 | (5,5) out of (50,50) | 0.006555928 |
| 1.00E-04 | (8,8) out of (50,50) | 0.002760237 |
| 1.00E-06 | (10,10) out of (50,50) | 0.002760120 |
| 1.00E-10 | (14,14) out of (50,50) | 0.002759815 |
| 1.00E-15 | (17,17) out of (50,50) | 0.002759815 |
| 1.00E-18 | (25,25) out of (50,50) | 5.76E-05 |
| 1.00E-20 | (29,29) out of (50,50) | 2.21E-07 |

Table 6.8: Select cosine basis from marginal distribution $N(1,1)$ and $N(1,1)$ to reconstruct joint distribution $N(\mu_{2D},\Sigma_{2D})$ when the correlation is 0.05.

| $\mu_{2D} = [1,1]$, $\Sigma_{2D} = \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}$, correlation = 0.05, Nr. of series truncation terms K = 50. | | |
|---|---|---|
| 'principal' threshold | Number of principal terms | $L_\infty$ error |
| 1.00E-03 | (7,7) out of (50,50) | 0.032412315 |
| 1.00E-05 | (9,9) out of (50,50) | 0.032028207 |
| 1.00E-10 | (14,14) out of (50,50) | 0.032027205 |
| 1.00E-15 | (17,17) out of (50,50) | 0.032027205 |
| 1.00E-20 | (29,29) out of (50,50) | 1.50E-05 |
| 1.00E-25 | (35,35) out of (50,50) | 4.31E-09 |
| 1.00E-30 | (41,41) out of (50,50) | 4.08E-09 |

Table 6.9: Select cosine basis from marginal distribution $N(1,1)$ and $N(1,1)$ to reconstruct joint distribution $N(\mu_{2D},\Sigma_{2D})$ when the correlation is 0.5.

For the third sampling strategy, there are almost no differences between the 0 correlation and 0.05 correlation situations. When the correlation increase to 0.5, this sampling method needs fewer cosine basis functions to reach a small error. For example, the former two sampling methods need about $40^2$ basis functions to reach the accuracy level of $10^{-8}$, while the third sampling method takes only $35^2$ basis functions to have a smaller error of $4.3^{-9}$.

Note that these conclusions are just based on experimental results of reconstructing the two-dimensional Gaussian density function with the principal Fourier-cosine basis functions of its marginal distributions. We need to continue exploring this idea and finding more theoretical support to make this method a solid solution method. FF Quant Advisory B.V. is researching further into this direction in another Master thesis at the time being.

Next, we repeat the same tests to recover the 3D Gaussian joint density function, with the uniform sampling strategy. And the correlation still plays a significant role in affecting the reconstruction accuracy.

IV. 3D implementation

| $\mu_{3D} = [1,1,1]$, $\Sigma_{3D} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.1 \end{bmatrix}$, correlation = 0.0, K = 50. | | |
|---|---|---|
| 'principal' threshold | Number of principal terms | $L_\infty$ error |
| 1.00E-01 | (2,2,2) out of (50,50,50) | 0.054856728 |
| 1.00E-03 | (8,8,8) out of (50,50,50) | 0.000308028 |
| 1.00E-06 | (12,12,12) out of (50,50,50) | 2.63E-07 |
| 1.00E-08 | (14,14,14) out of (50,50,50) | 3.31E-09 |
| 1.00E-10 | (16,16,16) out of (50,50,50) | 2.26E-11 |
| 1.00E-14 | (19,19,19) out of (50,50,50) | 8.40E-12 |

Table 6.10: Select cosine basis from 3 marginal distributions $N(1,1)$ to reconstruct joint distribution $N(\mu_{3D},\Sigma_{3D})$ when the correlation is 0.

| $\mu_{3D} = [1,1,1]$, $\Sigma_{3D} = \begin{bmatrix} 1.0 & 0.05 & 0.05 \\ 0.05 & 1.0 & 0.05 \\ 0.05 & 0.05 & 0.1 \end{bmatrix}$, correlation = 0.05, K = 50. | | |
|---|---|---|
| 'principal' threshold | Number of principal terms | $L_\infty$ error |
| 1.00E-01 | (2,2,2) out of (50,50,50) | 0.055284133 |
| 1.00E-03 | (8,8,8) out of (50,50,50) | 0.002116377 |
| 1.00E-06 | (12,12,12) out of (50,50,50) | 0.00229184 |
| 1.00E-08 | (14,14,14) out of (50,50,50) | 0.002291992 |
| 1.00E-10 | (16,16,16) out of (50,50,50) | 0.002291993 |
| 1.00E-14 | (19,19,19) out of (50,50,50) | 0.002291993 |
| 1.00E-18 | (30,30,30) out of (50,50,50) | 2.35E-05 |

Table 6.11: Select cosine basis from 3 marginal distributions $N(1,1)$ to reconstruct joint distribution $N(\mu_{3D},\Sigma_{3D})$ when the correlation is 0.05.

| $\mu_{3D} = [1,1,1]$, $\Sigma_{3D} = \begin{bmatrix} 1.0 & 0.05 & 0.0 \\ 0.05 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.1 \end{bmatrix}$, correlation = 0.05 between 2 of the dimensions ,K = 50. | | |
|---|---|---|
| 'principal' threshold | Number of principal terms | $L_\infty$ error |
| 1.00E-01 | (2,2,2) out of (50,50,50) | 0.055004485 |
| 1.00E-03 | (8,8,8) out of (50,50,50) | 0.00121238 |
| 1.00E-06 | (12,12,12) out of (50,50,50) | 0.001136214 |
| 1.00E-08 | (14,14,14) out of (50,50,50) | 0.001136099 |
| 1.00E-10 | (16,16,16) out of (50,50,50) | 0.001136097 |
| 1.00E-14 | (19,19,19) out of (50,50,50) | 0.001136097 |
| 1.00E-18 | (30,30,30) out of (50,50,50) | 9.98E-06 |

Table 6.12: Select cosine basis from 3 marginal distributions $N(1,1)$ to reconstruct joint distribution $N(\mu_{2D},\Sigma_{2D})$ when only 2 random variables are correlated with correlation 0.05.

The only difference between Table 6.11 and Table 6.12 is the number of correlated random variables. In Table 6.11, all random variables are mutually correlated, while there are only two random variables correlated in Table 6.12. And the former shows a relatively worse reconstruction accuracy than the less correlated situation.

## 6.2. A trial on one expectation operator

Now after testing the idea of utilizing the principal Fourier-cosine basis function of the marginal distributions in reconstructing the Gaussian joint density function, we continue with testing the often-seen application of multi-dimensional integration, the expectation operator.

### 6.2.1. METHODOLOGY

As we know, the expectation of a function $g(X)$, given that $X$ has a probability density function $f_X(x)$, is given by the inner product of $f$ and $g$:

$$\mathbb{E}[g(X)] = \int_{\mathbb{R}} g(x) f_X(x) dx \tag{6.3}$$

And for multi-variate functions, this formula still holds, and $f$ is the joint density function.

We then follow the same line of derivation as in [33] to derive the approximation formula for the 2D expectation operator: Truncate the ranges of integration into the domain $[a_1, b_1] \times [a_2, b_2]$ without lost of significant accuracy, and apply Fourier-cosine series expansion for the joint density function $f(x_1, x_2)$, to yield:

$$\mathbb{E}[g(x_1,x_2)] \approx \int_{a_2}^{b_2}\int_{a_1}^{b_2} g(x_1,x_2) \sum_{k_1=0}^{\infty}\sum_{k_2=0}^{\infty} \mathbf{A}_{k_1 k_2} \cos\left(k_1\pi\frac{x_1-a_1}{b_1-a_1}\right)\cos\left(k_2\pi\frac{x_2-a_2}{b_2-a_2}\right) dx_1 dx_2 \tag{6.4}$$

As proved in Chapter 4, we can exchange the summation and the integration to move $\mathbf{A}_{k_1 k_2}$ outside the integration. Then the integration part becomes the Fourier-cosine coefficients of $g(x_1, x_2)$ if multiplied with $\frac{2}{b_1-a_1}\frac{2}{b_2-a_2}$:

$$\mathbf{B}_{k_1 k_2} := \frac{2}{b_1-a_1}\frac{2}{b_2-a_2}\int_{a_2}^{b_2}\int_{a_1}^{b_2} g(x_1,x_2)\cos\left(k_1\pi\frac{x_1-a_1}{b_1-a_1}\right)\cos\left(k_2\pi\frac{x_2-a_2}{b_2-a_2}\right) dx_1 dx_2 \tag{6.5}$$

Truncating the series expansion with the number of cosine terms $K$ gives:

$$\mathbb{E}[g(x_1,x_2)] \approx \frac{b_1-a_1}{2}\frac{b_2-a_2}{2}{\sum}'^{K}_{k_1=0}{\sum}'^{K}_{k_2=0}\mathbf{A}_{k_1 k_2}\mathbf{B}_{k_1 k_2} \tag{6.6}$$

The authors of [8] derive an analytical solution to compute $\mathbf{A}_{k_1 k_2}$, which is the 2D-COS method. We use $\mathbf{S}_{k_1 k_2}$ to denote the coefficients obtained by the 2D-COS method:

$$\mathbf{S}_{k_1 k_2} := \frac{2}{b_1-a_1}\frac{2}{b_2-a_2}\int\int_{\mathbb{R}^2} f(\mathbf{x})\cos\left(k_1\pi\frac{x_1-a_1}{b_1-a_1}\right)\cos\left(k_2\pi\frac{x_2-a_2}{b_2-a_2}\right) dx_1 dx_2 \tag{6.7}$$

2D-COS is based on goniometric relation that transforms the product of cosine functions into a summation of new cosine functions:

$$2\cos(\alpha)\cos(\beta) = \cos(\alpha+\beta) + \cos(\alpha-\beta) \tag{6.8}$$

Accordingly, $\mathbf{S}_{k_1 k_2}$ can also be written as:

$$2\mathbf{S}_{k_1 k_2} = \mathbf{S}^{+}_{k_1 k_2} + \mathbf{S}^{-}_{k_1 k_2} \tag{6.9}$$

where

$$\mathbf{S}^{\pm}_{k_1 k_2} := \frac{2}{b_1-a_1}\frac{2}{b_2-a_2}\int\int_{\mathbb{R}^2} f(\mathbf{x})\cos\left(k_1\pi\frac{x_1-a_1}{b_1-a_1} \pm k_2\pi\frac{x_2-a_2}{b_2-a_2}\right) dx_1 dx_2 \tag{6.10}$$

Next, the coefficients $\mathbf{S}^{\pm}_{k_1 k_2}$ can be computed as:

$$\begin{aligned}
&\mathbf{S}^{\pm}_{k_1 k_2} \\
&= \frac{2}{b_1-a_1}\frac{2}{b_2-a_2}\operatorname{Re}\left(\int\int_{\mathbb{R}^2} f(\mathbf{x})\exp\left(ik_1\pi\frac{x_1-a_1}{b_1-a_1} \pm ik_2\pi\frac{x_2-a_2}{b_2-a_2}\right)d\mathbf{x}\right. \\
&\qquad\qquad\qquad\qquad\qquad\qquad \left.\exp\left(ik_1\pi\frac{x_1-a_1}{b_1-a_1} \mp ik_2\pi\frac{x_2-a_2}{b_2-a_2}\right)\right) \\
&= \frac{2}{b_1-a_1}\frac{2}{b_2-a_2}\operatorname{Re}\left(\varphi\left(k_1\pi\frac{x_1-a_1}{b_1-a_1}, \pm k_2\pi\frac{x_2-a_2}{b_2-a_2}\right)\exp\left(ik_1\pi\frac{x_1-a_1}{b_1-a_1} \mp ik_2\pi\frac{x_2-a_2}{b_2-a_2}\right)\right)
\end{aligned} \tag{6.11}$$

where $\operatorname{Re}(\cdot)$ means taking the real part of the complex input, and $\varphi$ is the characteristic function of random variables. For example, if $X \sim N(\mu, \Sigma)$, then its characteristic function is:

$$\varphi(\mathbf{t}) = \mathbb{E}(e^{i\mathbf{t}X}) = e^{i\mathbf{t}^T\mu - \frac{1}{2}\mathbf{t}^T\Sigma\mathbf{t}} \tag{6.12}$$

With the help of the 2D-COS method, we can derive the principal coefficients for the joint distribution function quickly, since we know the selected indices. Then we can compute $\mathbf{B}_{k_1 k_2}$ with the same cosine basis functions by Clenshaw-Curtis quadrature rule. Then Eq (6.6) turns to:

$$\bar{\mathbb{E}}[g(x_1, x_2)] \approx \frac{b_1-a_1}{2}\frac{b_2-a_2}{2}{\sum}'^{\tilde{K}}_{\tilde{k}_1=0}{\sum}'^{\tilde{K}}_{\tilde{k}_2=0}\mathbf{A}_{\tilde{k}_1\tilde{k}_2}\mathbf{B}_{\tilde{k}_1\tilde{k}_2} \tag{6.13}$$

where $\tilde{k}_1, \tilde{k}_2$ denote the selected indices for principal basis functions and $\tilde{K}$ is the smaller total number of principal cosine basis functions. And $\bar{\mathbb{E}}[\cdot]$ is the approximated expectation.

### 6.2.2. Results

We give results of computation of two example function expectations by the Principal-COS method in this section.

Given $g_1(\mathbf{x}) = \frac{e^{x_1}+e^{x_2}}{2}$, where $\mathbf{x} \sim N(\mathbf{0}, \mathbf{1})$, of which expectation we have computed in Chapter 5 and showed its analytical solution. In this chapter, we use a different method to approximate its exact solution. We compute the baseline expectation by Eq (6.6) with $K = 100$, which is a very conservative choice, to ensure the accuracy of the expectation. Then we use a smaller but efficient $K$, and increase the threshold gradually to filter out ‘unimportant’ cosine basis functions and compute expectations via Eq (6.13).

| $g_1(\mathbf{x}) = (e^{x_1} + e^{x_2})/2$, $\mathbf{x} \sim N(\mathbf{0}, \mathbf{1})$, K = 30 | | | |
|---|---|---|---|
| 'principal' threshold | Number of principal terms | $L_\infty$ error of expectation | time |
| 0 | (30, 30) | 1.93667e-12 | 0.02829 |
| 1e-10 | (14, 14) | 7.57518e-11 | 0.007856 |
| 1e-09 | (13, 13) | 2.34097e-09 | 0.007119 |
| 1e-08 | (12, 12) | 5.72244e-08 | 0.006617 |
| 1e-07 | (11, 11) | 1.10796e-06 | 0.005544 |
| 1e-06 | (10, 10) | 1.70215e-05 | 0.004993 |
| 1e-05 | (9, 9) | 0.00020797 | 0.004382 |
| 1e-04 | (8, 8) | 0.002027 | 0.003673 |

Table 6.13: The expectation results of $(e^{x_1} + e^{x_2})/2$ by Principal COS method.

Table 6.13, shows only $14^2$ cosine basis functions are important for computing the expectation of $g_1(\mathbf{x}) = (e^{x_1} + e^{x_2})/2$, without losing significant accuracy. And it is a great computation reduction compared to the full $30^2$ basis functions. Note that $\mathbb{E}(g_1(\mathbf{x})) = (\mathbb{E}(e^{x_1}) + \mathbb{E}(e^{x_2}))/2$, so there is no influence of correlation to compute the function expectation by Principal-COS method. So next we give an example that involves correlation.

Given $g_2(\mathbf{x}) = e^{x_1} * e^{x_2}$, where $\mathbf{x} \sim N(\mathbf{0}, \mathbf{1})$,this time we make $x_1$ and $x_2$ correlated and test different correlations.

| $g_1(\mathbf{x}) = e^{x_1} * e^{x_2}$, $\mathbf{x} \sim N(\mathbf{0}, \mathbf{1})$, K = 30 | | | | | |
|---|---|---|---|---|---|
| prncipal terms | correlations | | | | |
| | corr = 0 | corr = 0.1 | corr = 0.2 | corr = 0.4 | corr = 0.6 |
| (30, 30) | 6.9980e-12 | 4.4498e-11 | 7.7605e-11 | 1.2399e-10 | 1.4938e-10 |
| (14, 14) | 2.5027e-10 | 1.6533e-09 | 2.8645e-09 | 4.6431e-09 | 5.6944e-09 |
| (13, 13) | 7.7163e-09 | 4.8142e-08. | 8.3558e-08 | 1.3713e-07 | 1.7046e-07 |
| (12, 12) | 1.8870e-07 | 1.1040e-06 | 1.9170e-06 | 3.1845e-06 | 4.0135e-06 |
| (11, 11) | 3.6534e-06 | 1.9947e-05 | 3.4617e-05 | 5.8158e-05 | 7.4330e-05 |
| (10, 10) | 5.6127e-05 | 0.0002842 | 0.0004923 | 0.0008357 | 0.001083 |
| (9, 9) | 0.0006858 | 0.003199 | 0.005519 | 0.009454 | 0.01242 |
| (8, 8) | 0.006680 | 0.02847 | 0.04882 | 0.08425 | 0.1122 |
| (7, 7) | 0.05244 | 0.2001 | 0.3414 | 0.5920 | 0.7985 |

Table 6.14: The expectation results of $e^{x_1} * e^{x_2}$ by Principal COS method.

From Table 6.14 we can find the correlation does have an impact on the expectation accuracy obtained by the Principal-COS method, and the higher the correlation, the bigger the influence. However, these undesirable effects are not significant when the number of principal terms is big enough, such as 13 basis functions per dimension, which is still a great reduction in the total computational complexity compared to using the full cosine series.

Now we can compare the performance of computing integration of two methods we developed in this thesis project, the CPD-CG and Principal-COS, with two traditional numerical integration methods, the Trapezoidal rule and the Clenshaw-Curtis quadrature rule.

| Comparision among different numerical integration methods in 2D cases | | |
|---|---|---|
| Numerical integration method | Integration error | Time cost |
| Trapezoidal rule | 4.0140e-10 | 0.00076465 |
| Clenshaw-Curtis quadrature rule | 4.9785e-10 | 0.00056746 |
| CPD-CG method | 9.8765e-08 | 0.0032097 |
| Principal COS method | 4.44089e-16 | 0.0021777 |

Table 6.15: Integration performance of different numerical integration methods.

Note that as the Principal-COS method is an expectation operator, the $g(\mathbf{x})$ hereby is set to be 1 when we use it to compute the integral of the joint density function, which is a 2D Gaussian pdf. Table 6.15 illustrates that the traditional methods are better in two-dimension for Gaussian pdf, while our new CPD-CG method also has good performance. The timing deficiency can also be caused by coding since there are mature python functions for these traditional numerical integration methods, while the codes for these new algorithms are written by ourselves. Moreover, these two new methods are still under development. The CPD-CG method can be extended to higher dimensions, with the modified coding strategy, which is expected to outperform the straight forward implementation of the quadrature rules in high dimensions. And we have focused on developing and testing the concept via experiments for the Principal-COS method in this thesis report, which needs to be researched further.

In the end, we also give the comparison of expectation computation performance and their computational complexity in Table 6.16 and Table 6.17 as a reference.

| Comparision among different numerical integration methods in 2D cases | | |
|---|---|---|
| Numerical integration method | Expectation error | Time cost |
| Trapezoidal rule | 6.9624e-11 | 0.001627 |
| Clenshaw-Curtis quadrature rule | 6.5701e-11 | 0.0006717 |
| CPD-CG method | 9.9176e-08 | 0.005125 |
| Principal COS method | 5.7224e-08 | 0.007186 |

Table 6.16: Expectation performance of different numerical integration methods.

| Computational Complexity | | |
|---|---|---|
| Numerical integration method | Theoretical complexity | Practical complexity |
| Trapezoidal rule | $O(T^N)$ | $O(600^2)$ |
| CC quadrature rule | $O(T\log T) + O(T^N)$ | $O(35\log 35) + O(35^2)$ |
| CPD-CG | $O(KRS) * O(m\sqrt{\kappa})$ | $O(30 \cdot 2 \cdot 21) * O(m\sqrt{\kappa})$ |
| Principal COS | $O(\tilde{K}^N \cdot T\log T) + O(\tilde{K}^N \cdot T^N)$ | $O(12^2 \cdot 40\log 40) + O(12^2 \cdot 40^2)$ |

Table 6.17: Computational complexity of different numerical integration methods.

Here $T$ denotes the number of quadrature points, $N$ is the number of dimensions, $K$

is the number of Fourier-cosine basis functions, $R$ is the lower rank, $S$ is the total iteration number used in CPD-CG. And for the CG method, the computational complexity is $O(m\sqrt{\kappa})$ [40], where $m$ is the number of nonzero entries for the coefficient matrix of the linear equation, and $\kappa$ is its condition number. For the Clenshaw-Curtis quadrature rule, the computation for its weights is $O(T\log T)$, and the integration part is $O(T^N)$. The $\tilde{K}$ for Principal COS is the number of principal cosine basis functions after filtering by the threshold $10^{-8}$. And the value of the practical $K$ is chosen based on the convergence tests in the previous chapters, and the convergence tests for $T$ here can be found in the appendix.

# 7 Conclusion and Future work

## 7.1. Conclusion

As multi-dimensional numerical integration methods always suffer from the curse of dimension, this thesis project aims to find possible solutions to alleviate this issue. Based on Fourier-cosine series expansion, we can transfer the integration problem into solving the Fourier-cosine coefficients, which can be solved via supervised machine learning methods and of which the first coefficient is all we need to approximate the integral.

The multi-dimensional array expression, tensor, is compatible with the form of Fourier-cosine coefficients in high-dimension cases. Hence, it is indeed a smart and natural idea to reduce the computation of coefficients by tensor decomposition techniques, such as CPD, which gives a lower-rank representation of coefficient tensors. From recent literature, we found a method that applies a supervised machine learning method, SGD, to solve CPD as an optimization problem, which avoids instantiating the coefficient tensor. We first replicated their work from approximating the multivariate function, analyzed the error behaviors, improved their approach and extended it to generically solving the multi-dimensional integration.

The function convergence error of SGD is only at $10^{-3}$, which, based on our analysis, is the limitation in the original method and may be caused by the insufficient robustness of the search direction and the inflexible step size. We, therefore, proposed to use the CG as the solver instead of SGD, which has been tested to greatly improve the function approximation accuracy to at most $10^{-14}$, and the integration error is as low as $10^{-8}$. This improved method can decompose the Fourier coefficient tensor for multi-dimensional Gaussian distribution into rank-1 factor matrices, which reduces the computational complexity from $K^N$ to $KN$ in this case. We can also apply the CPD-CG method to compute general integrations such as the expectation of a function of random variables, which has also been tested to have good performance with accuracy at $10^{-8}$.

In the end, we developed a second method, the Principal-COS method, as a fast numerical solver for an expectation operator. It utilizes the fast decay property of Fourier-cosine coefficients of smooth density functions and the fact that a big fraction of those

Fourier coefficients can be smaller than $10^{-10}$. Thus the principal Fourier coefficients of the joint density function also dominate the Fourier coefficients of the target function of random variables.

By assuming that the non-important basis functions identified in the construction of marginal distributions remain unimportant in the construction of the joint density function, we manage to reduce computational complexity by only using the principal basis functions for each of the marginal distributions to compute the Fourier coefficients of the multi-dimensional joint density function, combined with high-dimensional COS method. The tests prove that we can drop over half of the cosine basis functions per dimension to speed up the computation. And we also see good results while testing the function with correlation. However, in this thesis project, we merely prove that this idea is conceptually feasible, and more future work needs to be done to really put this method into practice.

## 7.2. FUTURE WORK

We only apply the CPD-CG method at most in 3D situations in this thesis project, and it is far from enough to be really 'multi-dimensional'. At the moment, another thesis at FF Quant is already ongoing, which has already extended the CPD-CG method to high dimensions such as 6 or more with a detailed theoretical and experimental error analysis. A working paper will be drafted soon, which combines the results of this thesis and the continued analysis from that project and introduces the CPD-CG method as an efficient numerical solver for high-dimensional integration.

For the Principal-COS method, there are still a few important questions to be answered. For example, how big is the error when assuming the non-important cosine basis functions as seen in the construction of the marginal distributions remain unimportant for the construction of the joint density function? Is there room to improve the calculation of the Fourier coefficients of the target function, such as using CPD-CG or utilizing FFT plus an equidistant quadrature rule?

These future works will be picked up by other Msc thesis projects within FF Quant Advisory.

# 8 APPENDIX

## CONVERGENCE TESTS FOR $N$ IN TABLE 6.17

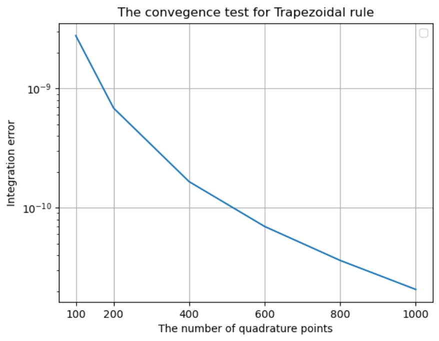


Figure 8.1: Quadrature points convergence test for Trapezoidal rule.

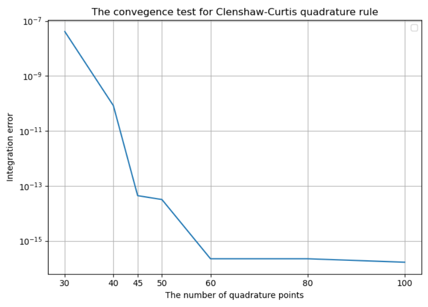


Figure 8.2: Quadrature points convergence test for Clenshaw-Curtis quadrature rule.

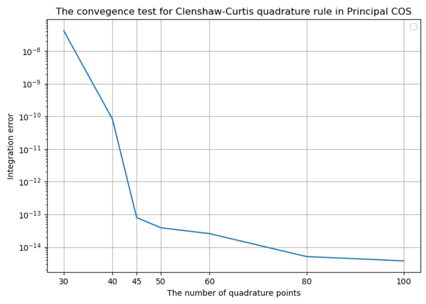


Figure 8.3: Quadrature points convergence test for Clenshaw-Curtis quadrature rule in Principal-COS.

## KHATRI-RAO PRODUCT DETAILS

$$
V_1{}' \odot V_2{}' = \begin{bmatrix} \frac{1}{2}cos0x^1[1] & cos0x^2[1] & \cdots & cos0x^M[1] \\ cos1x^1[1] & cos1x^2[1] & \cdots & cos1x^M[1] \\ cosk_1x^1[1] & cosk_1x^2[1] & \cdots & cosk_1x^M[1] \\ \vdots & \vdots & \ddots & \vdots \\ cosKx^1[1] & cosKx^2[1] & \cdots & cosKx^M[1] \end{bmatrix}
$$

$$
\odot \begin{bmatrix} cos0x^1[2] & cos0x^2[2] & \cdots & cos0x^M[2] \\ cos1x^1[2] & cos1x^2[2] & \cdots & cos1x^M[2] \\ cosk1x^1[2] & cosk1x^2[2] & \cdots & cosk1x^M[2] \\ \vdots & \vdots & \ddots & \vdots \\ cosKx^1[2] & cosKx^2[2] & \cdots & cosKx^M[2] \end{bmatrix}
$$

$$
= \begin{bmatrix} \frac{1}{2}cos0x^1[1]\cdot\begin{bmatrix} \frac{1}{2}cos0x^1[2] \\ cos1x^1[2] \\ \vdots \\ cosKx^1[2] \end{bmatrix} & \cdots & \frac{1}{2}cos0x^M[1]\cdot\begin{bmatrix} \frac{1}{2}cos0x^M[2] \\ cos1x^M[2] \\ \vdots \\ cosKx^M[2] \end{bmatrix} \\ cos1x^1[1]\cdot\begin{bmatrix} \frac{1}{2}cos0x^1[2] \\ cos1x^1[2] \\ \vdots \\ cosKx^1[2] \end{bmatrix} & \cdots & cos1x^M[1]\cdot\begin{bmatrix} \frac{1}{2}cos0x^M[2] \\ cos1x^M[2] \\ \vdots \\ cosKx^M[2] \end{bmatrix} \\ \vdots & \ddots & \vdots \\ cosKx^1[1]\cdot\begin{bmatrix} \frac{1}{2}cos0x^1[2] \\ cos1x^1[2] \\ \vdots \\ cosKx^1[2] \end{bmatrix} & \cdots & cosKx^M[1]\cdot\begin{bmatrix} \frac{1}{2}cos0x^M[2] \\ cos1x^M[2] \\ \vdots \\ cosKx^M[2] \end{bmatrix} \end{bmatrix}
$$

# BIBLIOGRAPHY

[1] B. David Gibb M.A., "A course in interpolation and numerical integration for the mathematical laboratory (edinburgh mathematical tracts, no. 2)," *Journal of the Institute of Actuaries*, vol. 50, no. 1, 1916.

[2] P. Davis and P. Rabinowitz, *Methods of Numerical Integration*. Dover Publications, 2007.

[3] C. W. Clenshaw and A. R. Curtis, "A method for numerical integration on an automatic computer," *Numerische Mathematik*, vol. 2, no. 1, pp. 197–205, 1960. DOI: 10.1007/BF01386223. [Online]. Available: https://doi.org/10.1007/BF01386223.

[4] F. E. James, "Monte carlo phase space," 1968.

[5] J. Bendavid, "Efficient monte carlo integration using boosted decision trees and generative deep neural networks," *arXiv preprint arXiv:1707.00028*, 2017.

[6] N. Kargas and N. D. Sidiropoulos, "Supervised learning and canonical decomposition of multivariate functions," *IEEE Transactions on Signal Processing*, vol. 69, pp. 1097–1107, 2021.

[7] G. Plonka, D. Potts, G. Steidl, and M. Tasche, *Numerical Fourier Analysis*. Springer, 2018.

[8] M. J. Ruijter and C. W. Oosterlee, "Two-dimensional fourier cosine series expansion method for pricing financial options," *SIAM Journal on Scientific Computing*, vol. 34, no. 5, B642–B671, 2012.

[9] T. Pellegrino and P. Sabino, "Pricing and hedging multiasset spread options using a three-dimensional fourier cosine series expansion method," *The Journal of Energy Markets*, vol. 7, pp. 71–92, Jun. 2014. DOI: 10.21314/JEM.2014.117.

[10] R. A. Harshman *et al.*, "Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis," *University of California at Los Angeles Los Angeles*, 1970.

[11] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009. DOI: 10.1137/07070111X. eprint: https://doi.org/10.1137/07070111X. [Online]. Available: https://doi.org/10.1137/07070111X.

[12] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966. DOI: 10.1007/BF02289464. [Online]. Available: https://doi.org/10.1007/BF02289464.

[13] D. Nion, K. N. Mokios, N. D. Sidiropoulos, and A. Potamianos, "Batch and adaptive parafac-based blind separation of convolutive speech mixtures," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, no. 6, pp. 1193–1207, 2009.

[14] N. D. Sidiropoulos, G. B. Giannakis, and R. Bro, "Blind parafac receivers for ds-cdma systems," *IEEE Transactions on Signal Processing*, vol. 48, no. 3, pp. 810–823, 2000.

[15] D. Nion and N. D. Sidiropoulos, "Tensor algebra and multidimensional harmonic retrieval in signal processing for mimo radar," *IEEE Transactions on Signal Processing*, vol. 58, no. 11, pp. 5693–5705, 2010.

[16] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017.

[17] B. Savas, "Analyses and tests of handwritten digit recognition algorithms," *LiTH-MAT-EX-2003-01, Link^ping University, Department of Mathematics*, 2003.

[18] N. Liu, B. Zhang, J. Yan, *et al.*, "Text representation: From vector to tensor," in *Fifth IEEE International Conference on Data Mining (ICDM'05)*, IEEE, 2005, 4–pp.

[19] J. Sun, S. Papadimitriou, and S. Y. Philip, "Window-based tensor analysis on high-dimensional and multi-aspect streams," in *Sixth International Conference on Data Mining (ICDM'06)*, IEEE, 2006, pp. 1076–1080.

[20] J. Sun, D. Tao, and C. Faloutsos, "Beyond streams and graphs: Dynamic tensor analysis," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 374–383.

[21] F. L. Hitchcock, "The expression of a tensor or a polyadic as a sum of products," *Journal of Mathematics and Physics*, vol. 6, no. 1-4, pp. 164–189, 1927.

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

[23] P. Paatero, "A weighted non-negative least squares algorithm for three-way 'parafac'factor analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 38, no. 2, pp. 223–242, 1997.

[24] T. G. Kolda, B. W. Bader, and J. P. Kenny, "Higher-order web link analysis using multilinear algebra," in *Fifth IEEE International Conference on Data Mining (ICDM'05)*, IEEE, 2005, 8–pp.

[25] T. Zhang and G. H. Golub, "Rank-one approximation to high order tensors," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 2, pp. 534–550, 2001.

[26] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[27] I. Domanov and L. D. Lathauwer, "Canonical polyadic decomposition of third-order tensors: Reduction to generalized eigenvalue decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 35, no. 2, pp. 636–660, 2014.

[28] A. Shashua and T. Hazan, “Non-negative tensor factorization with applications to statistics and computer vision,” in.

[29] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Fourth. The Johns Hopkins University Press, 1996.

[30] R. A. Harshman, “Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis,” *UCLA Working Papers in Phonetics*, vol. 16, pp. 1–84, 1970.

[31] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, “Gigatensor: Scaling tensor analysis up by 100 times - algorithms and discoveries,” *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2012.

[32] N. Vervliet, O. Debals, L. Sorber, and L. De Lathauwer, “Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis,” *IEEE Signal Processing Magazine*, vol. 31, no. 5, pp. 71–79, 2014.

[33] F. Fang, “The cos method: An efficient fourier method for pricing financial derivatives,” 2010.

[34] M. d. E. De Giorgi, “Sulla differenziabilitae l’analiticita delle estremali degli integrali multipli regolari,” *Ennio De Giorgi*, p. 167, 1957.

[35] J. Håstad, “Tensor rank is np-complete,” *Journal of Algorithms*, vol. 11, no. 4, pp. 644–654, 1990.

[36] A. P. George and W. B. Powell, “Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming,” *Machine Learning*, vol. 65, no. 1, pp. 167–198, 2006. DOI: 10.1007/s10994-006-8365-9. [Online]. Available: https://doi.org/10.1007/s10994-006-8365-9.

[37] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization.,” *Journal of machine learning research*, vol. 12, no. 7, 2011.

[38] A. Bhaya and E. Kaszkurewicz, “Steepest descent with momentum for quadratic functions is a version of the conjugate gradient method,” *Neural Networks*, vol. 17, no. 1, pp. 65–71, 2004, ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(03)00170-9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608003001709.

[39] R. Fletcher and C. M. Reeves, “Function minimization by conjugate gradients,” *The computer journal*, vol. 7, no. 2, pp. 149–154, 1964.

[40] J. R. Shewchuk *et al.*, “An introduction to the conjugate gradient method without the agonizing pain,” *Carnegie-Mellon University. Department of Computer Science Pittsburgh*, 1994.