Continuous low-rank tensor decompositions, with applications to stochastic optimal control and data assimilation

by

Alex Arkady Gorodetsky
B.S.E., University of Michigan (2010)
S.M., Massachusetts Institute of Technology (2012)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computational Science and Engineering
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 2017

Author .... Signature redacted .....
Department of Aeronautics and Astronautics
September 2, 2016

Certified by. Signature redacted .....
Sertac Karaman
Associate Professor, Aeronautics and Astronautics
Thesis Supervisor

Certified by... Signature redacted .....
Youssef M. Marzouk
Associate Professor, Aeronautics and Astronautics
Thesis Supervisor

Certified by... Signature redacted .....
Pierre Lermusiaux
Associate Professor, Mechanical Engineering
Thesis Committee

Accepted by .... Signature redacted .....
Youssef M. Marzouk
Associate Professor, Aeronautics and Astronautics
Co-Director, Computational Science and Engineering

Accepted by .. Signature redacted .....
Youssef M. Marzouk
Associate Professor, Aeronautics and Astronautics
Chair, Graduate Program Committee

Continuous low-rank tensor decompositions, with applications to stochastic optimal control and data assimilation

by

Alex Arkady Gorodetsky

Submitted to the Department of Aeronautics and Astronautics
on September 2, 2016, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computational Science and Engineering

Abstract

Optimal decision making under uncertainty is critical for control and optimization of complex systems. However, many techniques for solving problems such as stochastic optimal control and data assimilation encounter the curse of dimensionality when too many state variables are involved. In this thesis, we propose a framework for computing with high-dimensional functions that mitigates this exponential growth in complexity for problems with separable structure.

Our framework tightly integrates two emerging areas: tensor decompositions and continuous computation. Tensor decompositions are able to effectively compress and operate with low-rank multidimensional arrays. Continuous computation is a paradigm for computing with functions instead of arrays, and it is best realized by Chebfun, a MATLAB package for computing with functions of up to three dimensions. Continuous computation provides a natural framework for building numerical algorithms that effectively, naturally, and automatically adapt to problem structure.

The first part of this thesis describes a compressed continuous computation framework centered around a continuous analogue to the (discrete) tensor-train decomposition called the function-train decomposition. Computation with the function-train requires continuous matrix factorizations and continuous numerical linear algebra. Continuous analogues are presented for performing cross approximation; rounding; multilinear algebra operations such as addition, multiplication, integration, and differentiation; and continuous, rank-revealing, alternating least squares.

Advantages of the function-train over the tensor-train include the ability to *adaptively* approximate functions and the ability to compute with functions that are parameterized differently. For example, while elementwise multiplication between tensors of different sizes is undefined, functions in FT format can be readily multiplied together.

Next, we develop compressed versions of value iteration, policy iteration, and multilevel algorithms for solving dynamic programming problems arising in stochastic optimal control. These techniques enable computing global solutions to a broader set of problems, for example those with non-affine control inputs, than previously

possible. Examples are presented for motion planning with robotic systems that have up to seven states. Finally, we use the FT to extend integration-based Gaussian filtering to larger state spaces than previously considered. Examples are presented for dynamical systems with up to twenty states.

Thesis Supervisor: Sertac Karaman
Title: Associate Professor, Aeronautics and Astronautics

Thesis Supervisor: Youssef M. Marzouk
Title: Associate Professor, Aeronautics and Astronautics

Committee Member: Pierre Lermusiaux
Title: Associate Professor, Mechanical Engineering

Acknowledgments

This work would not have been possible without the support of so many people.

First, I thank my advisors, Youssef Marzouk and Sertac Karaman. Over the last six years, Youssef has provided me valuable guidance and the freedom to pursue my ideas and academic interests. I also thank him for giving me the opportunity to travel to conferences and do summer internships. They were invaluable experiences. I thank Sertac for introducing me to a new field and providing me the opportunity to really push and extend my research. His encouragement and excitement enabled me to persevere. I also thank Professor Pierre Lermusiaux, Dr. Tamara Kolda, and Professor John Tsitsiklis for providing valuable feedback on my thesis.

ACDL has been a fantastic place to grow up as a researcher. It was a great source of feedback and ideas. In particular, I am indebted to Tarek Moselhy, Chad Lieberman, Doug Allaire, Alessio Spantini, and Daniele Bigoni for the remarkable help they have given me over the last years.

I also thank Fabian Riether for making my code fly, and John Alora, Chris Grimm, and Ezra Tal for their great and rewarding collaborations.

Finally, I thank my family for their support and encouragement. They gave me motivation to continue and provided meaning to my work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Numerical methods and algorithms that enable computational modeling are important drivers of advancement in science and engineering. They help increase the scope and capabilities of computational models that, in turn, have dramatic impacts on society. For example, computational modeling has played a primary role in discovering unseen planets in our solar system [5], has assisted in the creation of autonomous systems that are able to identify new biological mechanisms [111], and has led to novel airplane designs that are largely designed, tested, and verified using computational techniques [114]. Society also depends heavily on computational modeling for a diverse set of needs such as weather prediction, financial market management, evaluation of economic policies, medical data analysis, and more. All of these systems rely on fast, effective, and accurate numerical algorithms that underpin computational models.

The utility of computational modeling and simulation for design, analysis, and control has resulted in an ever growing desire to increase the complexity of modeled systems. As computational scientists continue to try to model real systems with increasing levels of reality, they begin to use more detailed equations and larger amounts of data. The resulting increase in model complexity causes trouble for many existing numerical algorithms.

In this thesis, numerical techniques are developed that strive to enable computation for the increasingly complex models that are encountered in practice. The

techniques are the result of an integration of two areas: tensor decompositions and function approximation. Tensor decompositions have been shown to mitigate the curse of dimensionality associated with storing large amounts of discrete data. Function approximation has become extremely effective for developing accurate simplifications of complex models to aid in their analysis. Integrating these two areas produces a framework that automatically adapts to problem structure, provides an efficient representation for computation, and mitigates the curse of dimensionality in a variety of application areas. Applications of the resulting techniques are shown for stochastic optimal control and data assimilation.

1.1 Motivation: decision making under uncertainty

This research is motivated to create computational algorithms that perform automated decision making under uncertainty. Arguably, decision making under uncertainty is one of the biggest factors motivating the development of computational science. Optimal decision making in the presence of real-world uncertainties can potentially revolutionize countless scientific, engineering, and societal endeavors. Algorithmic breakthroughs in this area have potential for being the enabling technology for autonomous robotic systems that can effectively explore and monitor dangerous environments on and off Earth. They can enable efficient resource allocation and robust performance of national infrastructure such as the energy grid and transportation networks. Finally, optimal automated decision making under uncertainty will become a critical tool for analyzing and regulating complex systems such as social networks; applications can include resource allocation for preventing the spread of contagious diseases and optimal sensor placements for safety monitoring of water networks.

Automated decision making under uncertainty, however, remains a challenging task that is plagued by the curse of dimensionality: its computational expense grows exponentially with the number of degrees of freedom. Indeed, all aspects of decision making under uncertainty including data analysis [39], Bayesian inference and uncertainty quantification [10, 34], and Markov decision processes (MDP) [102, 105] are

afflicted.

Richard Bellman, whose namesake is the well known Bellman equation, defined the term *curse of dimensionality* [8] in the context of optimization methods. He provides the following example about the difficulty of optimization by enumeration in tensor product spaces: if each dimension is discretized into 10 points, then a ten-dimensional problem would have a search space of 10^{10} points, a twenty-dimensional problem would have a search space of 10^{20}, and a hundred-dimensional problem would have a search space of 10^{100} dimensions – more than the number of estimated atoms in the universe ($10^{78} - 10^{82}$)!

The Bellman equation describes the solution of a dynamic programming (DP) problem. Mitigating the curse of dimensionality associated with DP would greatly enhance the capabilities of automated decision making under uncertainty since DP is a general and versatile framework for formulating decision problems [12,13,15,102]. For example, stochastic shortest path [14] problems, stochastic optimal control problems, and more generally, Markov decision processes can be formulated as dynamic programming problems. Parallel computing methods can sometimes reduce the computational time for finding its solution, but some results suggest that stochastic Markov decision processes are inherently sequential and may not benefit too greatly from parallelization [99]. Thus, fundamentally new algorithms are needed to enable solutions for DP for a wider variety of systems. The algorithms that we propose in Chapter 4 are indeed sequential,f but they exploit a particular type of problem structure that enables computational feasibility. Other areas related to MDPs also suffer in high dimensions. For example, partially observable Markov decision processes (POMDPs) [20,25,85,115], which can be formulated as an MDP with states that are probability distributions, are notoriously difficult to solve.

Beyond optimization, Donoho points out that this problem also exists in function approximation and numerical integration [39]. He notes that to approximate a function of d variables, when it is only known that the function is Lipschitz, requires on the order of $(1/\epsilon)^d$ evaluations on a grid to obtain a uniform approximation error of ϵ. Similarly, he notes that integrating a function of d variables also requires order

$(1/\epsilon)^d$ for an integration scheme to have an error of ϵ.

Despite these incredible demands, effective numerical optimization, function approximation, and numerical integration are vital to the pursuit of computational algorithms for decision making under uncertainty. Many uncertainty quantification methods such as uncertainty propagation and Bayesian inference can be argued to be exercises in integration. Consider that extracting information from probability distributions requires the evaluation of integrals. In order to be able to solve these problems effectively, numerical methods that exploit more problem structure than just Lipschitz continuity must be developed.

The Monte Carlo method is one example of a ubiquitous algorithm within uncertainty quantification and optimization that is relied upon to mitigate the curse of dimensionality. However, many Monte Carlo algorithms themselves run into the curse of dimensionality. For example, the particle filtering algorithm for data assimilation attempts to propagate a distribution represented by samples through a dynamical system and through Bayes' rule. Particle filtering notoriously runs into problems due to sample impoverishment [10, 34], and an exponentially growing number of samples are needed as the dimension of the system increases [110].

Furthermore, the standard Monte Carlo estimator (without particle filtering) can also run into the curse of dimensionality since the variance of the estimator is proportional to the variance of the random variable whose mean is being estimated. Consider the simple function $f_d(X_1, \ldots, X_d) = X_1 \cdots X_d$ where X_i are independent and identically distributed random variables with mean 0 and variance 2. In this case we have

$$\begin{aligned}
\mathrm{var}(f_d) &= \mathbb{E}\left[(X_1 \cdots X_d)^2\right] - \left(\mathbb{E}\left[X_1 \cdots X_d\right]\right)^2, \\
&= \mathbb{E}\left[X_1^2 \cdots X_d^2\right] - \left(\mathbb{E}\left[X_1\right]\right)^2 \cdots \left(\mathbb{E}\left[X_d\right]\right)^2, \\
&= \mathbb{E}\left[X_1^2\right] \cdots \mathbb{E}\left[X_d^2\right] - \left(\mathbb{E}\left[X_1\right]\right)^2 \cdots \left(\mathbb{E}\left[X_d\right]\right)^2, \\
&= \prod_{i=1}^{d} \left(\mathrm{var}(X_i) + \left(\mathbb{E}\left[X_i\right]\right)^2\right) - \prod_{i=1}^{d} \left(\mathbb{E}\left[X_i\right]\right)^2,
\end{aligned}$$

which for our case means

$$\text{var}(f_d) = 2^d.$$

Therefore, the number of samples required to estimate the expectation of f_d grows exponentially with dimension to achieve similar levels of accuracy. Practitioners typically hope that the variance of their outputs does *not* grow exponentially with the complexity of their models, so that the number of required samples (and therefore the computational cost of analysis) does not grow exponentially.

We are motivated to seek other methods that can mitigate the curse of dimensionality to develop algorithms that converge faster than the $\sqrt{N}$ rate associated with Monte Carlo. While Monte Carlo requires that the variance of our models does not grow too fast, the primary type of structure exploited in this research is that of output *separability*. The separability of a function refers to the notion that a multivariate function can be approximated by the sum of the products of univariate functions, e.g.,

$$f(x_1, \ldots, x_d) = \sum_{i=1}^{R} \phi_1(x_1) \ldots \phi_d(x_d).$$

If the *rank* R is "small", then f is considered to be a *low-rank* function. Low-rank functions can be integrated with complexity that is linear in dimension and polynomial with the rank. For example, the function f_d described above is rank 1, and it can be integrated with a computational cost that scales linearly with dimension. We hypothesize that separable structure is prevalent in many relevant application areas, and we show that its exploitation is indeed feasible for certain stochastic optimal control and data assimilation problems. The resulting methods can be viewed as complementary tools to Monte Carlo methods for high dimensional problems.

1.2 Computing with functions

One of the main ways to combat the computationally intensive nature of algorithms in high dimensions is through approximating the computationally expensive aspects of the problem in a simpler way. For example, one expensive aspect of Markov chain

Monte Carlo, a Monte Carlo variant that is useful for sampling from arbitrary distributions, is repeated evaluations of the likelihood function. These evaluations are particularly expensive when they involve solutions to partial differential equations (PDEs). A common strategy in these situations is to replace the likelihood with a surrogate model or emulator [31,88]. Then, instead of evaluating a PDE, the surrogate model is used within the evaluation of the likelihood. Surrogate models also play a role in solving Markov decision processes. Approximate dynamic programming [15, 102] often use surrogate modeling techniques to transform a computationally infeasible problem to a simpler one by using a finite dimensional representation for the approximation of a value function. In general, function approximation techniques for generating surrogate models play an important role in a wide variety of numerical algorithms.

This thesis is partially inspired by the pioneering view on the use of function approximation within computation developed by Trefethen, Battles, Platte, Townsend, and others, that is realized through the Chebfun package for the MATLAB computer language [4, 101, 117–119]. Their view is that end users of numerical software are typically interested in computing with *functions* instead of arrays. Users often have PDEs, optimization problems, or inference problems that are defined in continuous spaces in terms of functions. Extracting information from these functions, representing them on the computer, and performing operations with them should be within the domain of the software, not the user. The user should not have to specify that they would like to discretize their functions using a particular grid or represent them with a particular parameterization. Instead, the user should only need to specify the *accuracy* with which they want to compute or a *storage* level they would like to stay within.

The fundamental backbone that allows for such a beautiful possibility lies in interpreting continuous objects on a computer not as being *discretized* but rather *parameterized.* We purposely make a subtle, but vitally important, distinction between discretization and parameterization. For us, discretization implies that the computer can only "see" a function on a computer through its evaluation at a set of points. Pa-

Discretized

```
struct {
  int n;
  double * fvals;
} Function;
```

Parameterized

```
struct {
  int n;
  double * fvals;
} Function;
double evalf(struct Function f, double x);
double inner_product(struct Function f, struct Function g);
double add(struct Function f, struct Function g);
double multiply(struct Function f, struct Function g);
double max(struct Function f);
```

Figure 1-1: Discretized vs. parameterized functions.

rameterization implies that the computer can "see" a function on a computer through a finite set of parameters *and* a set of routines that map those parameters to outputs of interest.

As an example, consider a function discretized onto some grid of n points that results in the function values $\{f^1, \ldots, f^n\}$. A discretization viewpoint might store this function as an array of floating point values.

f^1	f^2	$\ldots$	f^n

The continuous framework utilizing the parameterized viewpoint, on the other hand, would store this function as an object that contains the function values *and* various routines for evaluating the function at arbitrary locations, computing inner products, finding its maximum, etc. These routines would be created by interpreting the discretized function values as, for instance, nodes of a spline or linear element approximation. The encoding for the discretized and the parameterized functions in the C programming language would then look similar to Figure 1-1.

The resulting effect on numerical algorithms is tremendous. Consider a matrix whose columns consists of m different functions. Performing matrix-factorization such as the QR or LU decompositions now takes on a new meaning. In the discretized framework, the m functions are each evaluated at n locations and the standard matrix QR and LU decompositions are performed. In the parameterized format the m functions may still be evaluated at n locations, but the QR and LU decomposi-

tions are modified because the inner product is a *continuous* inner product in the QR decomposition and the pivots for the LU are chosen using a *continuous* maximization technique. Put another way, in the discretized framework linear algebra algorithms can only work with discrete inner products, discrete maximization, elementwise addition, elementwise multiplication, etc. These discrete algorithms use no other information about the function that is actually being represented.

There are both theoretical and practical advantages to using this continuous framework. Theoretically, one can begin to link the convergence of matrix factorizations such as the QR, LU, and SVD decompositions to properties of the original functions [118]. The practical advantages include automated adaptation and better error control for operations with functions. For example, if an algorithm calls for multiplying two functions together, the resulting function often needs to be parameterized by a larger set of parameters to maintain accuracy. Instead, if functions are simply multiplied using elementwise multiplication of their discretizations, then the resulting product may not accurately represent the continuous product between the original functions. Furthermore, performing multiplication, addition, and taking the inner product between functions that are discretized in different ways is ill-defined. Such operations would effectively *force* the user to provide the computer with more information.

While continuous computation has been effectively developed for univariate and bivariate functions by the Chebfun package, theory and routines for representing and operating with general multivariate functions has up to now been lacking. The major goal in this thesis is to develop a framework for continuous computation for high-dimensional functions. One realization of this framework is the new numerical computing software package *Compressed Continuous Computation (C^3)* [54] that is used for most of the numerical examples within this thesis.

The new framework relies on representing functions in a compressed format that is a continuous extension of low-rank tensor decompositions. In contrast to other low-rank functional approaches [26, 40, 103], however, our framework provides more flexibility and generality. The previous approaches all rely on converting a low-rank

functional approximation problem to a low-rank tensor approximation problem, where the tensor represents certain parameters of the function approximation scheme that are linearly related to the output. Our framework never requires this conversion, in fact, the resulting algorithms for high-dimensional computation are independent from the underlying parameterization. In the end, our ability to incorporate a wider variety of, even nonlinear, parameterizations results in a more general framework than these previous approaches.

1.3 Objectives and contributions

The primary objective of this research is to develop a framework for continuous computation with multivariate functions. Two secondary objectives are to develop scalable algorithms for (1) stochastic optimal control and (2) data assimilation. The solution to the primary objective enables the design of novel algorithms for tackling the secondary objectives. The major high-level contributions of this research are then

1. A framework for continuous computation with multivariate functions that exploits low-rank tensor decompositions,

2. Application of a low-rank framework for the solution of dynamic programming equations arising in stochastic optimal control, and

3. Application of continuous computation to integration-based Gaussian filtering.

Numerous lower-level contributions were necessary for the realization of these objectives. For the development of continuous computation, we make the following contributions:

- Development of maximum-volume based CUR/skeleton decompositions of vector-valued functions,

- Extension of continuous matrix factorizations to the case of QR and LU factorization of matrix-valued functions, and

- Continuous versions of cross approximation, rounding, and alternating least squares for a continuous tensor-train decomposition — called the function-train (FT) decomposition.

For the application of the low-rank framework to stochastic optimal control and data assimilation, we make the following contributions:

- Utilization of the function-train decomposition within value and policy iteration for solving Bellman's equation,
- Utilization of the function-train decomposition for multi-level schemes by enabling evaluations of optimal value functions and policies in continuous domains, and
- Utilization of the function-train for multivariate integration within integration-based Gaussian filtering.

1.4 Thesis outline

The thesis is organized as follows. In Chapter 2, the continuous linear algebra background is provided. In that chapter, all of the continuous linear algebra that is used throughout this thesis is described and discussed. Chapter 3 describes the novel low-rank functional decomposition, the function-train (FT). The FT is shown to be the continuous analogue of the discrete tensor-train decomposition. All the continuous analogues of operations performed with the discrete tensor-train are then described. These operations include cross approximation, rounding, multilinear algebra, and alternating least squares. Chapter 4 applies the framework described in Chapters 2 and Chapter 3 to the problem of stochastic optimal control and introduces low-rank algorithms for value iteration and policy iteration. Chapter 5 applies the framework to the problem of Gaussian filtering. Chapter 6 is a summary and conclusion.

Chapter 2

Continuous linear algebra

Continuous linear algebra and, specifically, continuous matrix factorizations form the basis for a framework to design flexible, efficient, and adaptive function approximation algorithms. One theoretical benefit of this framework is that function properties can inform convergence properties of corresponding factorizations, e.g., determine the decay of singular values from a property on the regularity of the function. A practical algorithmic benefit of designing continuous algorithms is that they are naturally adaptive and efficient. For instance, elementwise multiplication of discretized functions may not yield the best representation of the product; however, the errors involved with multiplying two functions represented in a basis of polynomials can be well controlled.

In effect, designing computational algorithms based on continuous linear algebra requires carrying the knowledge of how a discretization represents a function, e.g., a vector of floating numbers represents coefficients of a polynomial or the nodes of a spline, through the entire algorithm. This knowledge can then be automatically used to maintain accuracy through every step of a numerical algorithm. Examples of software developed using these principles include the MATLAB-based Chebfun [101], the Julia-based ApproxFun [92], and the author's own C-based Continuous Compressed Computation (C^3) toolbox [54].

In this chapter, we discuss three topics: interpreting scalar-, vector-, and matrix-valued functions as continuous analogues to vectors and matrices; defining and per-

forming continuous matrix factorizations such as the LU, QR, and CUR decompositions; and computing with a specific realization of this framework where the functions are represented as an expansion of orthonormal basis functions. These topics form the foundation for the low-rank multivariate algorithms that are later described in Chapter 3. This chapter is an adaptation of the first part of [57].

2.1 Scalar-, vector-, and matrix-valued functions

Just as the primary elements of discrete linear algebra are vectors and matrices, the primary elements of continuous linear algebra are scalar-, vector-, and matrix-valued functions. These elements appear in both the theory and algorithms behind the low-rank functional decompositions that are discussed in Chapter 3. The difference between these elements lies in the *type of output.* Corresponding to the names of these elements, for a fixed input x, the output of a scalar-valued function is a scalar, the output of a vector-valued function is a vector, and the output of a matrix-valued function is a matrix.

The inputs to these functions typically lie in a d-dimensional input space $\mathscr{X} \subset \mathbb{R}^d$ that is formed through the tensor product of sets $\mathscr{X}_i \subset \mathbb{R}$ according to $\mathscr{X} = \mathscr{X}_1 \times \cdots \times \mathscr{X}_d$. Unless explicitly stated otherwise, each of the subsets $\mathscr{X}_i$ are closed intervals $\mathscr{X}_i = [a_i, b_i]$ with $b_i > a_i$. Furthermore, each element of the input space $x \in \mathscr{X}$ refers to a tuple $x = (x_1, x_2, \ldots, x_d)$ where $x_i \in \mathscr{X}_i$.

It is sometimes useful to think about functions that take d-inputs as equivalent to functions that take two inputs such that, in the second function, each input is a grouping of a subset of the original d inputs. For example, consider a six-dimensional input space $\mathscr{X} = \mathscr{X}_1 \times \mathscr{X}_2 \times \mathscr{X}_3 \times \mathscr{X}_4 \times \mathscr{X}_5 \times \mathscr{X}_6$ then we can group the first three and last three variables according to

$$z \equiv \{x_1, x_2, x_3\} \text{ and } y \equiv \{x_4, x_5, x_6\} \text{ such that}$$
$$f(z, y) = f(x_1, x_2, x_3, x_4, x_5, x_6),$$

where on the right side of the second line we treat the function $f : \mathscr{X}_1 \times \cdots \times \mathscr{X}_6 \to \mathbb{R}$ as taking six inputs, and on the left side of second line we define the sets $\mathscr{Z} = \mathscr{X}_1 \times \mathscr{X}_2 \times \mathscr{X}_3$ and $\mathscr{Y} = \mathscr{X}_4 \times \mathscr{X}_5 \times \mathscr{X}_6$ and treat $f : \mathscr{Z} \times \mathscr{Y} \to \mathbb{R}$ as taking two inputs.

Such an interpretation has two advantages. First, many continuous matrix factorizations have been defined only for bivariate functions, and we would still like to use them for general multivariate functions. Second, low-rank decompositions of multidimensional arrays and multivariate functions often rely on computing with two-dimensional objects, e.g., matrices obtained through reshaping of a tensor. The particular type of grouping that is most useful in this work is that of splitting an input domain at the kth variable. To this end, we will often refer to the following groupings of variables

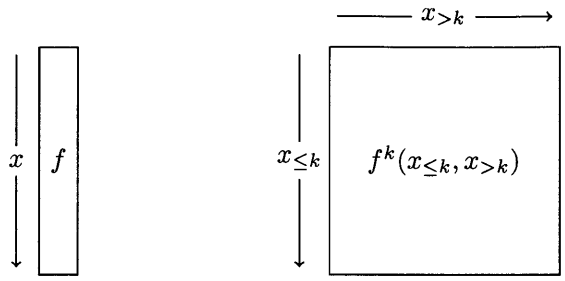
$$x_{\leq k} \in \mathscr{X}_{\leq k} \text{ where } \mathscr{X}_{\leq k} = \mathscr{X}_1 \times \cdots \times \mathscr{X}_k,$$
$$x_{>k} \in \mathscr{X}_{>k} \text{ where } \mathscr{X}_{>k} = \mathscr{X}_{k+1} \times \cdots \times \mathscr{X}_d.$$

We also consider the integration of certain functions through out this dissertation. To this end, let μ_i denote the Lebesgue measure over the interval $[a_i, b_i]$ and $\mu = \mu_1 \times \mu_2 \cdots \times \mu_d$ denote the Lebesgue measure over $\mathscr{X}$. The integral of a scalar-valued function, $f : \mathscr{X} \to \mathbb{R}$, denoted as $\int f dx$, is always with respect to this Lebesgue measure. For example, for $x_i \in [a_i, b_i]$ and $x \in \mathscr{X}$ we have $\int f(x_i) dx_i \equiv \int f(x_i) \mu(dx_i)$, and for the product space we have $\int f(x) dx \equiv f(x) \mu(dx)$.

Scalar-, vector-, and matrix-valued functions and the relationships between these elements are now described in more detail.

2.1.1 Definitions and interpretations

Formally, a scalar-valued function, $f : \mathscr{X} \to \mathbb{R}$ is a map from $\mathscr{X}$ to the reals and is denoted by a lowercase letter. For the purposes of continuous linear algebra, it is useful to think of various "vector" and "matrix" representations of f. Two such interpretations are provided in Figure 2-1. The first interpretation, shown in Figure 2-



(a) "$\infty \times 1$" column vector (b) "$\infty \times \infty$" matrix of the separated form f^k

Figure 2-1: Vector and matrix interpretations of scalar-valued functions

1a views f as vector with an (uncountably) infinite number of rows, where each "row" is indexed by $x \in \mathscr{X}$. This interpretation motivates the inner product between scalar-valued function as a continuous analogue to the inner product between discrete vector, e.g., let $g : \mathscr{X} \to \mathbb{R}$, then

$$\langle f, g \rangle = \int_{\mathscr{X}} f(x)g(x)dx,$$

where the sum of the discrete inner product is effectively "replaced" with an integral for the continuous inner product.

A second interpretation of a scalar-valued function, shown in Figure 2-1b, is as an "$\infty \times \infty$" matrix obtained through a separated form, or unfolding, of f obtained by the variable splittings $\mathscr{X}_{\leq k}$ and $\mathscr{X}_{>k}$. The superscript k will often be added to a function to encourage this interpretation, e.g. f^k denotes that the variables are separated after the kth variable. Formally, we have $f^k : \mathscr{X}_{\leq k} \times \mathscr{X}_{>k} \to \mathbb{R}$ where

$$f^k(x_{\leq k}, x_{>k}) = f^k(\{x_1, \ldots, x_k\}, \{x_{k+1}, \ldots, x_d\}) = f(x_1, \ldots, x_d). \tag{2.1}$$

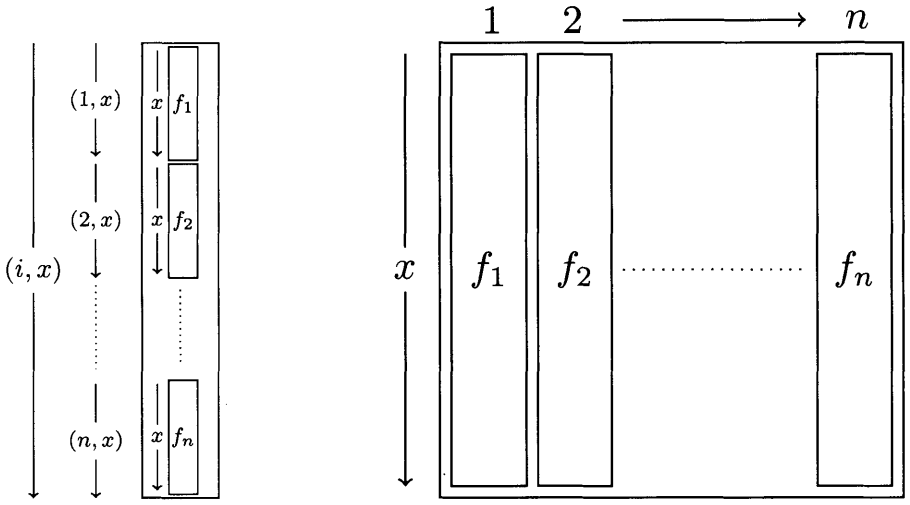
In this interpretation, the rows of the "matrix" are indexed by $x_{\leq k} \in \mathscr{X}_{\leq k}$ and the columns are indexed by $x_{>k} \in \mathscr{X}_{>k}$.

Formally, a vector-valued function $F : \mathscr{X} \to \mathbb{R}^n$, where $n \in \mathbb{Z}_+$, is a map from $\mathscr{X}$ to a vector with n elements, and it is denoted by an uppercase letter. Analogously to the scalar-valued function case, various "vector" and "matrix" interpretations of F are useful. These interpretations arise from viewing a vector-valued function as an array

of scalar-valued functions. Consider a set of n scalar-valued functions $f_i : \mathscr{X} \to \mathbb{R}$. Then, the i-th output of F can be indexed according to

$$F[i](x) = f_i(x), \quad i = 1, \ldots, n.$$

The particular interpretation of a vector-valued function depends on the arrangement of the scalar-valued functions. Three visualizations that are important in this work are provided in Figures 2-2 and 2-3. The first interpretation of a vector-valued function



(a) "$\infty \times 1$" column vector (b) "$\infty \times n$" quasimatrix

Figure 2-2: Vector-valued functions as vectors and matrices in continuous linear algebra

provided by Figure 2-2a is that of a vector with an infinite number of elements. This vector is formed by the concatenation, or vertical stacking, of the scalar-valued functions f_i. Due to this concatenation, the rows of this vector are now indexed by a tuple $(i, x) \in \{1, \ldots, n\} \times \mathscr{X}$. This viewpoint motivates the following definition for an inner product. Let $G : \mathscr{X} \to \mathbb{R}^n$ be another vector-valued function whose outputs are referred to by the scalar-valued functions $g_i : \mathscr{X} \to \mathbb{R}$, then the inner product between F and G is

$$\langle F, G \rangle = \sum_{i=1}^{n} \langle F[i](x), G[i](x) \rangle = \sum_{i=1}^{n} \langle f_i(x), g_i(x) \rangle. \tag{2.2}$$

The second interpretation of a vector-valued function is that of an "$\infty \times n$" *quasi-matrix*[1], and is shown by Figure 2-2b. The rows of the quasimatrix are indexed by $x \in \mathscr{X}$ and the columns are indexed by $i = 1, \ldots, n$ such that each column corresponds to the scalar-valued function f_i. This interpretation of a vector-valued function arises in the context of continuous matrix factorizations such as the continuous LU or continuous QR decomposition of Sections 2.2 and 2.2, respectively.

The final interpretation comes from considering separated forms, or unfoldings, of vector-valued functions. These unfoldings F^k are obtained by splitting the input space at the kth input. Thus, an unfolding $F^k : \mathscr{X}_{\leq k} \times \mathscr{X}_{>k} \to \mathbb{R}^n$ takes values

$$F^k(x_{\leq k}, x_{>k}) = F^k(\{x_1, \ldots, x_k\}, \{x_{k+1}, \ldots, x_d\}) = F(x_1, \ldots, x_d).$$

A visualization of this representation as an "$\infty \times \infty$" matrix is provided in Figure 2-3. In Figure 2-3, the rows are indexed by an index-value pair $(i, x_{\leq k})$ where $i = 1, \ldots, n$,

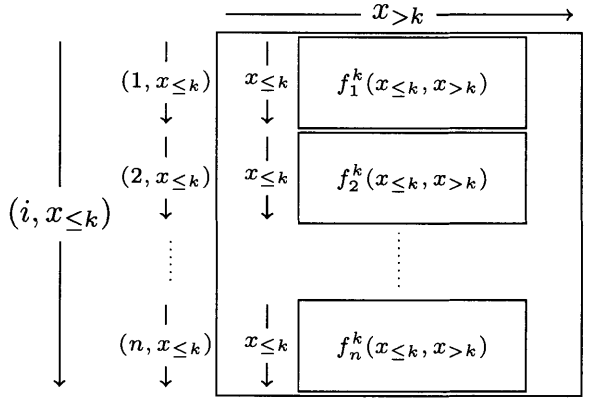


Figure 2-3: Visualization of unfoldings F^k of a vector-valued function F

and the columns are indexed by $x_{>k}$. This interpretation will be used heavily when considering the skeleton decomposition of a vector-valued function in Section 2.3.1.

Formally, a matrix-valued function $\mathcal{F} : \mathscr{X} \to \mathbb{R}^{n \times m}$, where $n, m \in \mathbb{Z}_+$, is a map from $\mathscr{X}$ to a $n \times m$ matrix, and is denoted by an upper-case, calligraphic, non-bold letter. The matrix-valued function can be visualized as an array of vector-valued

[1]Called a quasimatrix in, e.g., [4, 118] because it corresponds to a matrix of infinite rows and n columns

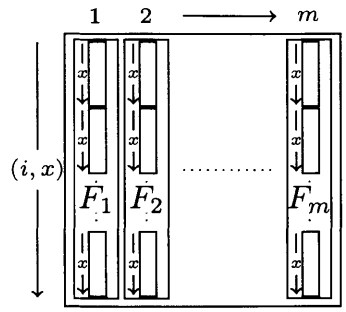


Figure 2-4: Visualization of a matrix-valued function $\mathcal{F}$.

functions $F_j : \mathcal{X} \to \mathbb{R}^n$ for $j = 1 \ldots m$ according to

$$\mathcal{F} = [F_1 \; F_2 \ldots \; F_m] \text{ such that } \mathcal{F}[:, j](x) = F_j(x), \quad j = 1, \ldots, m.$$

If we consider that each F_j is itself an array of scalar-valued functions $f_{i,j} : \mathcal{X} \to \mathbb{R}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$ then the matrix-valued function can also be visualized as a two-dimensional array of scalar-valued functions given by

$$\mathcal{F} = \begin{bmatrix} f_{1,1} & \cdots & f_{1,m} \\ \vdots & & \vdots \\ f_{n,1} & \cdots & f_{n,m} \end{bmatrix}.$$

In Chapter 3, we show that the cores of a function represented in the function-train format are matrix-valued functions, and this interpretation of a matrix-valued function becomes important.

We can interpret the matrix-valued function as a matrix with an infinite number of rows and m columns, as shown in Figure 2-4. Each "row" of this matrix is indexed by the pair (i, x) where $i \in \{1, \ldots, n\}$ and $x \in \mathcal{X}$. Each column refers to the vector-valued function F_j and is indexed by a discrete variable $j = 1, \ldots, m$.

The inner product between $\mathcal{F}$ and another matrix-valued function $\mathcal{G} : \mathcal{X} \to \mathbb{R}^{n \times m}$ is defined as

$$\langle \mathcal{F}, \mathcal{G} \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} \langle \mathcal{F}[i, j](x), \mathcal{G}[i, j](x) \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} \langle f_{i,j}(x), g_{i,j}(x) \rangle. \tag{2.3}$$

This inner product can be interpreted as the inner product between two flattened vector-valued functions, and it is analogous to the square of the Frobenius norm of a matrix.

2.1.2 Multiplication

In addition, we define products between arrays of functions (vector- and matrix-valued functions) and arrays of scalars (vectors and matrices), as shown in Table 2.1.

	$F : \mathscr{X} \to \mathbb{R}^n$	$\mathcal{F} : \mathscr{X} \to \mathbb{R}^{m \times n}$
$\boldsymbol{v} \in \mathbb{R}^n$	$g = F\boldsymbol{x} \iff$ $g(x) = \sum_{i=1}^{n} \boldsymbol{v}[i]F[i](x)$	$G = \mathcal{F}\boldsymbol{x} \iff$ $G[i](x) = \mathcal{F}[i,:](x)\boldsymbol{x}$
$\mathbf{A} \in \mathbb{R}^{n \times l}$	$G = F\mathbf{A} \iff$ $G[i](x) = F\mathbf{A}[:,i]$	$\mathcal{G} = \mathcal{F}\mathbf{A} \iff$ $\mathcal{G}[i,j](x) = \mathcal{F}[i,:](x)\mathbf{A}[:,j]$

Table 2.1: Product of a vector-valued function and a vector (top left); a vector-valued function and a matrix (bottom left); a matrix-valued function and a vector (top right); a matrix-valued function and a matrix (bottom right).

We also define products between functional elements. Let $\mathscr{X} \subset \mathbb{R}^d$ and $\mathcal{Y} \subset \mathbb{R}^{d_2}$, where $d_2 \in \mathbb{Z}^+$. Then products between vector-valued or matrix-valued functions on these domains yield vector-valued or matrix-valued functions on the space $\mathscr{X} \times \mathcal{Y}$. Notation and a summary of these operations are provided in Table 2.2. We now

	$F : \mathscr{X} \to \mathbb{R}^n$	$\mathcal{F} : \mathscr{X} \to \mathbb{R}^{m \times n}$
$H : \mathcal{Y} \to \mathbb{R}^n$	$g = FH \iff$ $g(x,y) =$ $\sum_{i=1}^{n} F[i](x)H[i](y)$	$G = \mathcal{F}H \iff$ $G[i](x,y) =$ $\sum_{j=1}^{n} \mathcal{F}[i,j](x)H[j](y)$
$\mathcal{H} \in \mathcal{Y} \to \mathbb{R}^{n \times k}$	$G = F\mathcal{H} \iff$ $G[i](x,y) =$ $\sum_{j=1}^{n} F[j](x)\mathcal{H}[j,i](y)$	$\mathcal{G} = \mathcal{F}\mathcal{H} \iff$ $\mathcal{G}(x,y) = \mathcal{F}(x)\mathcal{H}(y)$

Table 2.2: Product of two vector-valued functions (top left); a vector-valued function and a matrix-valued function (bottom left); a matrix-valued function and a vector-valued function (top right); two matrix-valued functions (bottom right).

turn to factorizations of vector- and matrix-valued functions.

2.2 Continuous matrix factorizations

Approximating a black box tensor in tensor-train format requires performing a sequence of standard matrix factorizations. Our *continuous* framework requires continuous equivalents of these factorization for the elements of continuous linear algebra, scalar-valued functions, vector-valued functions, and matrix-valued functions, described above.

The LU and QR factorizations and the singular value decomposition (SVD) of a vector-valued function are primary ingredients of any continuous numerical linear algebra package such as Chebfun [101] or ApproxFun [92]. For vector-valued functions of one variable, these decompositions are discussed in [4]. Computing these decompositions often requires continuous analogues of standard discrete algorithms. For example, Householder triangularization may be used to compute the QR decomposition of a vector-valued function [119]. Extensions of these algorithms to functions of two variables are described in [118]. We will use these bivariate extensions as building blocks for our higher dimensional constructions, and we discuss the relevant background.

LU decomposition

To extend the cross approximation and `maxvol` algorithms to the continuous setting, we will require the LU decomposition of a vector-valued function. The key components of this decomposition are a set of pivots $\{z_1, z_2, \ldots, z_n\}$, $z_i \in \mathscr{X}$, and a vector-valued function $L = [\ell_1 \ \ell_2 \ \cdots \ \ell_n]$ (here written as a quasimatrix, with scalar-valued functions ℓ_i) that is "psychologically" lower triangular [118].

A psychologically lower triangular vector-valued function is defined such that column k has zeros at all z_i for $i = 1, \ldots, k-1$. Furthermore, if $\ell_i(z_i) = 1$, then L is *unit lower triangular*. Finally, if $|\ell_k(x)| \leq |\ell_k(z_k)|$ for all $x \in \mathscr{X}$, then L is *diagonally maximal*. Using these definitions we recall the definition of an LU decomposition of a vector-valued function [4, 118]:

Definition 1 (LU factorization of a vector-valued function [4]). *Let* $F : \mathscr{X} \to \mathbb{R}^n$ *be*

a vector-valued function. An LU factorization of F is a decomposition of the form $F = L\mathbf{U}$, where $\mathbf{U} \in \mathbb{R}^{n\times n}$ is upper triangular and $L : \mathscr{X} \to \mathbb{R}^n$ is unit lower triangular and diagonally maximal.

The LU factorization may be computed using Gaussian elimination with row pivoting according to the algorithm in [118].

We can extend this definition of an LU factorization to matrix-valued functions. In particular, the decomposition will result in a set of pivots $\{(i_1, z_1), (i_2, z_2), \ldots, (i_n, z_n)\}$; a psychologically lower triangular matrix-valued function $\mathcal{L}$; an upper triangular matrix $\mathbf{U}$. The definition of the pivots is motivated by viewing $\mathcal{L} : \mathscr{X} \to \mathbb{R}^{m\times n}$ as collection of n columns $[\mathcal{L}[:,1] \ldots \mathcal{L}[:,n]]$, where each column is a vector-valued function to be interpreted as an "$\infty \times 1$" vector, as described in Section 2.1. Each pivot is then specified by a *(row, value)* tuple. A lower triangular matrix-valued function is defined such that $\mathcal{L}[:,k]$ has zeros at all $\{(i_1, z_1), (i_2, z_2), \ldots (i_{k-1}, z_{k-1})]$; that is, $\mathcal{L}[:,1]$ has no enforced zeros, $\mathcal{L}[:,2]$ has a zero in row i_1 at the value z_1, $\mathcal{L}[:,3]$ has zeros at (i_1, z_1) and (i_2, z_2), etc. Furthermore, if $\mathcal{L}[i_k,k](z_k) = 1$ then $\mathcal{L}$ is called *unit lower triangular*, and if $|\mathcal{L}[i,k](x)| \leq |\mathcal{L}[i_k,k](z_k)|$ for all $x \in \mathscr{X}$ and for all $i \in \{1,\ldots,m\}$, then $\mathcal{L}$ is *diagonally maximal.* Using these notions we define an LU decomposition of a matrix-valued function.

Definition 2 (LU factorization of a matrix-valued function)**.** *Let $\mathcal{F} : \mathscr{X} \to \mathbb{R}^{m\times n}$ be a matrix-valued function. An LU factorization of $\mathcal{F}$ is a decomposition of the form $\mathcal{F} = \mathcal{L}\mathbf{U}$ where $\mathbf{U} \in \mathbb{R}^{n\times n}$ is upper triangular, and $\mathcal{L} : \mathscr{X} \to \mathbb{R}^{m\times n}$ is unit lower triangular and diagonally maximal.*

We also implement the LU decomposition of a matrix-valued function using Gaussian elimination with row-pivoting.

QR decomposition

Another decomposition that will be necessary for function-train rounding and for the cross approximation of multivariate functions is the QR factorization of a quasimatrix.

Definition 3 (QR factorization of a vector-valued function [4]). *Let $F : \mathscr{X} \to \mathbb{R}^n$ be a vector-valued function. A QR factorization of F is a decomposition of the form $F = Q\mathbf{R}$, where the vector-valued function $Q : \mathscr{X} \to \mathbb{R}^n$ consists of n orthonormal scalar-valued functions and $\mathbf{R} \in \mathbb{R}^{n\times n}$ is an upper triangular matrix.*

This QR factorization can be computed in a stable manner using a continuous extension of Householder triangularization [119]. In this dissertation, we also require the QR decomposition of a matrix-valued function.

Definition 4 (QR factorization of a matrix-valued function). *Let $\mathcal{F} : \mathscr{X} \to \mathbb{R}^{m\times n}$ be a matrix-valued function. A QR factorization of $\mathcal{F}$ is a decomposition of the form $\mathcal{F} = \mathcal{Q}\mathbf{R}$, where the columns of $\mathcal{Q} : \mathscr{X} \to \mathbb{R}^{m\times n}$ are orthonormal vector-valued functions and $\mathbf{R} \in \mathbb{R}^{n\times n}$ is an upper triangular matrix.*

Since we have defined the inner product of vector-valued functions in (2.2) and therefore are able to take inner products of the columns of $\mathcal{F}$, we can also compute this factorization in a stable manner using Householder triangularization. We can consider the ranks of both vector-valued and matrix-valued functions as the number of nonzero elements of the diagonal of $\mathbf{R}$.

Singular value decomposition

Many of our theoretical results will employ the functional SVD.

Definition 5 (Functional SVD). *Let $\mathcal{Y} \times \mathcal{Z} \subset \mathbb{R}^d$ and let $g : \mathcal{Y} \times \mathcal{Z} \to \mathbb{R}$ be in $L^2(\mathcal{Y} \times \mathcal{Z})$. A singular value decomposition of g is*

$$g(y, z) = \sum_{j=1}^{\infty} \sigma_j u_j(y) v_j(z), \tag{2.4}$$

where the left singular functions $u_j : \mathcal{Y} \to \mathbb{R}$ are orthonormal in $L^2(\mathcal{Y})$, the right singular functions $v_j : \mathcal{Z} \to \mathbb{R}$ are orthonormal in $L^2(\mathcal{Z})$, and $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$ are the singular values.

In practice, the summation in (2.4) is truncated to some finite number of terms r and we group the first r left singular functions $u_1, \ldots, u_r$ into the vector-valued

function $U : \mathcal{Y} \to \mathbb{R}^r$ such that $U[i] = u_i$. Similarly, we group the right singular functions into a vector-valued function $V : \mathcal{Z} \to \mathbb{R}^r$, with $V[i] = v_i$. If we also define $\mathbf{S} = \text{diag}(\sigma_1, \ldots, \sigma_r)$, then we can write the functional SVD as $g = U\mathbf{S}V$. In this form, we say that g is a function that has an *SVD rank* of r. This notion of the SVD of a function has existed for a while [108, 112] and is also called the Schmidt decomposition [108]. In general, convergence can be assumed to be in $L^2(\mathcal{Y} \times \mathcal{Z})$. As described in [118], when $g : [a, b] \times [c, d] \to \mathbb{R}$ is also Lipschitz continuous, then the series in (2.4) converges absolutely and uniformly.

The functional SVD is useful for analyzing certain separated representations f^k of multivariate functions f. For example, in Section 3.2.1 we show that the ranks of our function-train representation are bounded by the SVD ranks of the separated functions $f^k : \mathcal{Y} \times \mathcal{Z} \to \mathbb{R}$, where we put $\mathcal{Y} = \mathcal{X}_{\leq x}$ and $\mathcal{Z} = \mathcal{X}_{>x}$ in the above definition.

Next, we present a decomposition similar to the functional SVD, but for vector-valued rather than scalar-valued functions. We call the decomposition an *extended SVD*, because it shares some properties with the functional SVD, such as a separation rank, and because it decomposes the vector-valued function into a sum of products of orthonormal functions. This decomposition appears in the proofs of Theorems 3 and 4, as well as in the skeleton decomposition of a multivariate function described in Section 3.3.1.

Definition 6 (Extended SVD). *Let $\mathscr{Y} \times \mathscr{Z} \subset \mathbb{R}^d$ and let $G : \mathscr{Y} \times \mathscr{Z} \to \mathbb{R}^n$ be a vector-valued function such that $G[i] \in L^2(\mathscr{Y} \times \mathscr{Z})$ for $i = 1, \ldots, n$. A rank r extended SVD of $G(y, z)$ is a factorization*

$$G[i](y, z) = \sum_{j=1}^{r} \sigma_j U_j[i](y) v_j(z), \tag{2.5}$$

where the left singular functions $U_j : \mathscr{Y} \to \mathbb{R}^n$ are orthonormal[2] and vector-valued, the right singular functions $v_j : \mathscr{Z} \to \mathbb{R}$ are orthonormal and scalar-valued, and $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$ are the singular values.

[2]Orthonormality here is defined with respect to the inner product in (2.2).

We can combine the left singular functions to form the matrix-valued function $\mathcal{U} : \mathcal{Y} \to \mathbb{R}^{n \times r}$ where $\mathcal{U}[:, j] = U_j$, and as in the functional SVD, we can group the right singular functions to form the vector-valued function $V : \mathcal{Z} \to \mathbb{R}^r$ such that $V[j] = v_j$. If we also gather the singular values in a diagonal matrix $\mathbf{S}$, then the extended SVD can be written as $G = \mathcal{U}\mathbf{S}V$.

The main interpretation of this decomposition is that G contains n functions that have the same right singular vectors and different left singular vectors. We will exploit two properties of this decomposition for the proof of Theorem 1. The first is the fact that for any fixed $\hat{z}$, the vector-valued function $\hat{G}(y) \equiv G(y, \hat{z})$ can be represented as a linear combination of the columns of $\mathcal{U}(y)$. Second, for any fixed $\hat{y}$ and column $\hat{\imath}$, the function $\hat{g}(z) \equiv G[\hat{\imath}](\hat{y}, z)$ can be represented as a linear combination of the columns of V.

Again, in the case of functions of more than two variables, the extended SVD can be applied to the function's k-separated representation. In particular, if $\mathcal{X} \subset \mathbb{R}^d$ and we have a vector-valued function $F : \mathcal{X} \to \mathbb{R}^n$, then we can consider the extended SVD of the separated form $F^k : \mathcal{Y} \times \mathcal{Z} \to \mathbb{R}$, where we put $G = F^k$, $\mathcal{Y} = \mathcal{X}_{\leq k}$, and $\mathcal{Z} = \mathcal{X}_{>k}$ in the definition above.

2.3 Skeleton decomposition and cross approximation

In Section 3.2.1 we show that the FT representation of a function f can be computed by performing a sequence of SVDs of various separated versions of f. Such an algorithm would require a prohibitive number of function evaluations, however. In this section, we develop an alternative low-rank decomposition that requires evaluations of the function only at relatively few points, lying along certain "fibers" of the input domain. This decomposition is a continuous version of the skeleton/CUR decomposition of a matrix [18, 52, 87], and is critical to the practical black box approximation algorithm described in Section 3.3.1. In particular, we now develop this continuous CUR decomposition for the specific case of vector-valued functions.

This section is split into three parts. First we establish conditions for the exis-

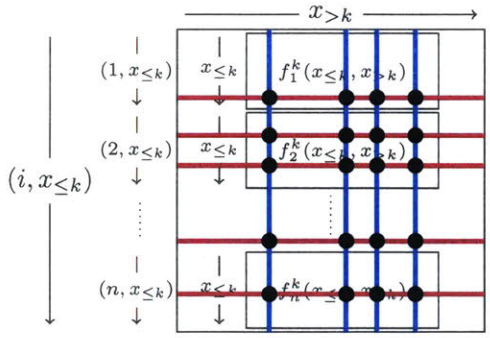


Figure 2-5: Column fibers (blue) and row fibers (red) of a vector-valued function for a particular k-separated form. The black circles form a sub-matrix of the vector-valued function.

tence of a continuous CUR decomposition. Then we motivate a construction of the decomposition based on the maximum volume concept. Finally, we describe a cross approximation algorithm for computing the decomposition.

Before introducing the continuous CUR decomposition, we formally describe its components. In particular, we need to formulate the notion of a *fiber* of a vector-valued function, which is analogous to a row or column of a matrix. In particular, we consider the k-separated form of a vector-valued function $F^k : \mathscr{X}_{\leq k} \times \mathscr{X}_{>k} \to \mathbb{R}^n$, and view it as an "$\infty \times \infty$" matrix, shown in Figure 2-3, where the rows are indexed by $(i, x_{\leq k})$ and the columns are indexed by $(x_{>k})$. According to this interpretation, row fibers are scalar-valued functions and column fibers are vector-valued functions. The intersection of the row and column fibers forms a sub-matrix of the vector-valued function. These choices are graphically highlighted in Figure 2-5.

Definition 7 (Row fiber). *A row fiber of a vector-valued function F^k is the scalar-valued function $r_\alpha : \mathscr{X}_{>k} \to \mathbb{R}$ obtained by fixing an (index, value) pair $\alpha = (i, z) \in \{1, \ldots, n\} \times \mathscr{X}_{\leq k}$ so that*

$$r_\alpha(x_{>k}) = F^k[i](z, x_{>k}).$$

Definition 8 (Set of row fibers). *A set of row fibers of a vector-valued function F^k is the vector-valued function $R_{\boldsymbol{\alpha}} : \mathscr{X}_{>k} \to \mathbb{R}^\ell$ obtained by fixing a set of tuples $\boldsymbol{\alpha} = \{(i_1, z_1), \ldots, (i_\ell, z_\ell)\}$, where $(i_j, z_j) \in \{1, \ldots, n\} \times \mathscr{X}_{\leq k}$, so that*

$$R_{\boldsymbol{\alpha}} = [r_{\alpha_1}\ r_{\alpha_2}\ \cdots\ r_{\alpha_\ell}],$$

where $\alpha_j \equiv (i_j, z_j)$, $j = 1, \ldots, \ell$.

We have corresponding definitions for column fibers and a set of column fibers.

Definition 9 (Column fiber). *A column fiber of a vector-valued function* F^k *is the vector-valued function* $C_y : \mathscr{X}_{\leq k} \to \mathbb{R}^n$ *obtained by fixing an element* $y \in \mathscr{X}_{>k}$ *such that*

$$C_y(x_{\leq k}) = F^k(x_{\leq k}, y).$$

We can group a set of ℓ column fibers together to obtain a matrix-valued function $\mathcal{C}_{\boldsymbol{y}}$ as follows.

Definition 10 (Set of column fibers). *A set of column fibers of a vector-valued function* F^k *is the matrix-valued function* $\mathcal{C}_{\boldsymbol{y}} : \mathscr{X}_{\leq k} \to \mathbb{R}^{n \times \ell}$ *obtained by fixing a set* $\boldsymbol{y} = \{y_1, \ldots, y_\ell\}$, *where* $y_i \in \mathscr{X}_{>k}$ *for* $i \in \{1, \ldots, \ell\}$, *such that*

$$\mathcal{C}_{\boldsymbol{y}} = [C_{y_1}\ C_{y_2}\ \cdots\ C_{y_\ell}].$$

The intersection of a set of row fibers and a set of column fibers forms a submatrix.

Definition 11 (Submatrix of a vector-valued function). *A submatrix of a vector valued function* F^k *is the matrix* $\bar{\mathbf{F}}^k \in \mathbb{R}^{\ell \times \ell}$ *obtained by fixing a set of columns* $\boldsymbol{y} = \{y_1, \ldots, y_\ell\}$ *and a set of rows* $\boldsymbol{\alpha} = \{(i_1, z_1), \ldots, (i_\ell, z_\ell)\}$, *as in Definitions 8 and 10 respectively, such that*

$$\bar{\mathbf{F}}^k[a, b] = F[i_a](z_a, y_b) \quad \text{for} \quad 1 \leq a, b \leq \ell, \tag{2.6}$$

or equivalently,

$$\bar{\mathbf{F}}^k[a, b] = \mathcal{C}_{\boldsymbol{y}}[i_a, b](z_a) = R_{\boldsymbol{\alpha}}[a](y_b). \tag{2.7}$$

Next, we discuss the notion of a skeleton or CUR decomposition of vector-valued functions.

2.3.1 Existence of the skeleton decomposition of vector-valued functions

In this subsection, we define the skeleton decomposition of a vector-valued function and establish conditions for its existence based on the SVD of the k-separated form of the function. The existence of a skeleton decomposition for a scalar-valued function can then be obtained by choosing $n = 1$. Note that the skeleton decomposition of a function has already been used, for example, in [6, 7]. These analyses make smoothness assumptions on the function. Our results, in contrast, formulate the skeleton decomposition using only low-rank properties of the function, without introducing any explicit smoothness assumptions.

Definition 12 (Skeleton/CUR decomposition of a vector-valued function)**.** *Let $F : \mathcal{X} \to \mathbb{R}^n$ be a vector-valued function. The rank ℓ skeleton decomposition of separated form F^k is a factorization of the type*

$$F^k[i](x_{\leq k}, x_{>k}) = \mathcal{C}_{\boldsymbol{y}}[i, :](x_{\leq k})\mathbf{G}R_{\boldsymbol{\alpha}}(x_{>k}), \tag{2.8}$$

for all $i \in \{1, \ldots, n\}$, $x_{\leq k} \in \mathcal{X}_{\leq k}$, and $x_{>k} \in \mathcal{X}_{>k}$. The matrix valued function $\mathcal{C}_{\boldsymbol{y}} : \mathcal{X}_{\leq k} \to \mathbb{R}^{n \times \ell}$ represents a set of ℓ column-fibers (Definition 10), $\mathbf{G}$ is an $\ell \times \ell$ matrix, and the vector-valued function $R_{\boldsymbol{\alpha}} : \mathcal{X}_{>k} \to \mathbb{R}^{\ell}$ represents a set of ℓ row-fibers (Definition 8).

Theorem 1. *Let $F : \mathcal{X} \to \mathbb{R}^n$ be a vector-valued function and let $F^k : \mathcal{X}_{\leq k} \times \mathcal{X}_{>k} \to \mathbb{R}^n$ be its k-separated form for any $1 \leq k < d$. If F^k has a rank-r extended SVD, then a rank r skeleton decomposition* (2.8) *of F^k exists.*

Proof. The proof is constructive and requires one to choose r linearly independent[3] column- and row-fibers of F^k. The intersection of these fibers will form the submatrix $\bar{\mathbf{F}}^k$ such that choosing $\mathbf{G} = [\bar{\mathbf{F}}^k]^\dagger$, where $\mathbf{A}^\dagger$ refers to the Moore-Penrose pseudoinverse of matrix $\mathbf{A}$, will yield a correct construction. The proof strategy then involves

[3] A set of r linearly independent scalar-valued functions $(f_i)_{i=1}^r$, $f_i : \mathcal{X} \to \mathbb{R}$ is one in which the only solution to the equation $\sum_{i=1}^r a_i f_i(x) = 0$ is $a_i = 0$ for $i = 1, \ldots, r$. An analogous definition holds for vector-valued functions.

partitioning the input space into four sets and showing pointwise equality on each partition.

Choose a set of ℓ row indices $\boldsymbol{\alpha}$ (see Definition 8) such that the vector-valued function $R_{\boldsymbol{\alpha}}$ contains r linearly independent scalar-valued functions r_{α_i}, $i = 1, \ldots, \ell$. Next, choose a set of ℓ column indices $\boldsymbol{y}$ (see Definition 10) such that the matrix-valued function $\mathcal{C}_{\boldsymbol{y}}$ contains r linearly independent vector-valued functions C_{y_i}, $i = 1, \ldots, \ell$. Note that these choices also define a submatrix $\bar{\mathbf{F}}^k$. Furthermore, let $\tilde{F}$ be the proposed skeleton decomposition

$$\tilde{F}(x_{\leq k}, x_{>k}) = \mathcal{C}_{\boldsymbol{y}}(x_{\leq k}) \, [\bar{\mathbf{F}}^k]^\dagger \, R_{\boldsymbol{\alpha}}(x_{>k}). \tag{2.9}$$

We now seek to show that $\tilde{F}$ and F^k are pointwise equal.

We show $F^k = \tilde{F}$ by decomposing the space $(\{1, \ldots, n\} \times \mathcal{X}_{\leq k}) \times \mathcal{X}_{>k}$ into four partitions: $(\boldsymbol{\alpha} \times \boldsymbol{y})$, $(\boldsymbol{\alpha}^\perp \times \boldsymbol{y})$, $(\boldsymbol{\alpha} \times \boldsymbol{y}^\perp)$, and $(\boldsymbol{\alpha}^\perp \times \boldsymbol{y}^\perp)$, where $\boldsymbol{\alpha}^\perp = (\{1, \ldots, n\} \times \mathcal{X}_{\leq k}) \setminus \boldsymbol{\alpha}$ and $\boldsymbol{y}^\perp = \mathcal{X}_{>k} \setminus \boldsymbol{y}$. We also rely on the following property of the Moore-Penrose pseudoinverse $\mathbf{A}^\dagger$ of a matrix $\mathbf{A}$:

$$\mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}. \tag{2.10}$$

First, we define matrices $\mathbf{C}, \mathbf{R} \in \mathbb{R}^{\ell \times \ell}$ such that

$$\mathbf{C}[a, b] = \mathcal{C}_{\boldsymbol{y}}[i_a, b](z_a) = C_{y_b}[i_a](z_a), \quad \text{and} \tag{2.11}$$

$$\mathbf{R}[a, b] = R_{\boldsymbol{\alpha}}[a](y_b) = r_{\alpha_a}(y_b), \tag{2.12}$$

where for $1 \leq a, b \leq \ell$, $(i_a, z_a) \in \boldsymbol{\alpha}$ and $y_b \in \boldsymbol{y}$. One can verify that

$$\mathbf{C} = \mathbf{R} = \bar{\mathbf{F}}^k \tag{2.13}$$

using (2.7).

Now we consider the first partition $\boldsymbol{\alpha} \times \boldsymbol{y}$. Showing equality over this partition is equivalent to showing the equality of $\bar{\mathbf{F}}^k$ and the corresponding submatrix $\bar{\tilde{F}}^k$ of $\tilde{F}$.

We proceed using (2.9), (2.11), and (2.12) to represent the elements of $\bar{\tilde{F}}^k$ as

$$\begin{aligned}\bar{\tilde{F}}^k[a,b] &= \tilde{F}[i_a](z_a, y_b) = \mathcal{C}_{\boldsymbol{y}}[i_a,:](z_a)[\bar{\mathbf{F}}^k]^\dagger R_{\boldsymbol{\alpha}}(y_b), \\ &= \mathbf{C}[a,:][\bar{\mathbf{F}}^k]^\dagger \mathbf{R}[:,b],\end{aligned}$$

where for $1 \leq a, b \leq \ell$, $(i_a, z_a) \in \boldsymbol{\alpha}$ and $y_b \in \boldsymbol{y}$. According to (2.13) and (2.10), we obtain

$$\bar{\tilde{F}}^k = \mathbf{C}[\bar{\mathbf{F}}^k]^\dagger \mathbf{R} = \bar{\mathbf{F}}^k [\bar{\mathbf{F}}^k]^\dagger \bar{\mathbf{F}}^k = \bar{\mathbf{F}}^k.$$

Thus, $\tilde{F}$ and F are equivalent over this partition.

Next we consider the partition $\boldsymbol{\alpha}^\perp \times \boldsymbol{y}$. By definition of linear independence and the SVD rank of a function, $\forall (i, x_{\leq k}) \in \{1, \ldots, n\} \times \mathcal{X}_{\leq k}$ there exists a $\boldsymbol{q}_{i,x_{\leq k}} \in \mathbb{R}^\ell$ such that

$$F^k[i](x_{\leq k}, x_{>k}) = \sum_{j=1}^{\ell} \boldsymbol{q}_{i,x_{\leq k}}[j] r_{\alpha_j}(y) \quad \forall x_{>k} \in \mathcal{X}_{>k}. \tag{2.14}$$

In other words, every scalar-valued function $F[i](x_{\leq k}, \cdot) : \mathcal{X}_{>k} \to \mathbb{R}$ can be written as a linear combination of the scalar-valued functions comprising $R_{\boldsymbol{\alpha}}$. This property follows directly from the fact that $R_{\boldsymbol{\alpha}}$ contains r linearly independent functions and that F^k has an extended SVD of rank r. In particular, for all $y_b \in \boldsymbol{y}$ we have

$$F^k[i](x_{\leq k}, y_b) = \mathcal{C}_{\boldsymbol{y}}[i,b](x_{\leq k}) = \boldsymbol{q}_{i,x_{\leq k}} \mathbf{R}[:,b], \quad \forall (i, x_{\leq k}) \in \{1, \ldots, n\} \times \mathcal{X}_{\leq k} \tag{2.15}$$

We now show that (2.8) holds for $(i, x_{\leq k}) \in \boldsymbol{\alpha}^{\perp}$ and $y_b \in \boldsymbol{y}$:

$$\begin{aligned}
\tilde{F}[i](x_{\leq k}, y_b) &= \mathcal{C}_{\boldsymbol{y}}[i,:](x_{\leq k})\,[\bar{\mathbf{F}}^k]^{\dagger}\,\mathbf{R}[:,b] \\
&= \boldsymbol{q}_{i,x_{\leq k}}\mathbf{R}\,[\bar{\mathbf{F}}^k]^{\dagger}\,\mathbf{R}[:,b] \\
&= \boldsymbol{q}_{i,x_{\leq k}}\bar{\mathbf{F}}^k\,[\bar{\mathbf{F}}^k]^{\dagger}\,\bar{\mathbf{F}}^k[:,b] \\
&= \boldsymbol{q}_{i,x_{\leq k}}\bar{\mathbf{F}}^k[:,b] \\
&= \boldsymbol{q}_{i,x_{\leq k}}\mathbf{R}[:,b] \\
&= F^k[i](x_{\leq k}, y_b)
\end{aligned}$$

where the first equality comes from the definition of $\tilde{F}$, the second comes from (2.15), the third comes from (2.13), the fourth equality comes from using (2.10), the fifth also comes from (2.13), and the sixth comes from another application of (2.15).

We can proceed with a symmetric argument for the partition $\boldsymbol{\alpha} \times \boldsymbol{y}^{\perp}$. Linear independence and finite rank require that for all $x_{>k} \in \mathcal{X}_{>k}$, there exists a $\boldsymbol{q}_{x_{>k}} \in \mathbb{R}^{\ell}$ such that

$$F^k[i](x_{\leq k}, y) = \sum_{j=1}^{\ell} \mathcal{C}_{\boldsymbol{y}}[i,j](x_{\leq k})\boldsymbol{q}_y[j] \quad \forall (i, x_{\leq k}) \in \{1,\ldots,n\} \times \mathcal{X}_{\leq k}.$$

We can obtain an analogue to (2.15) by noticing that for all $(i_a, z_a) \in \boldsymbol{\alpha}$ we have

$$F^k[i_a](z_a, x_{>k}) = R_{\boldsymbol{\alpha}}[a](x_{>k}) = \mathbf{C}[a,:]\boldsymbol{q}_{x_{>k}}, \quad \forall x_{>k} \in \mathcal{X}_{>k}. \tag{2.16}$$

Using (2.16), a symmetric argument yields the desired equivalence for elements of the partition $\boldsymbol{\alpha} \times \boldsymbol{y}^{\perp}$.

Finally, we show that (2.8) holds pointwise for all $(x_{\leq k}, x_{>k}) \in (\boldsymbol{\alpha}^{\perp} \times \boldsymbol{y}^{\perp})$. First,

by (2.14) and (2.16) we have

$$
\begin{aligned}
F^k[i](x_{\leq k}, x_{>k}) &= \sum_{j=1}^{\ell} \boldsymbol{q}_{i,x_{\leq k}}[j] R_{\boldsymbol{\alpha}}[j](x_{>k}) \\
&= \sum_{j=1}^{\ell} \boldsymbol{q}_{i,x_{\leq k}}[j] F^k[i_j](z_j, x_{>k}), \text{ where } (i_j, z_j) = \boldsymbol{\alpha}[j], \\
&= \sum_{j=1}^{\ell} \boldsymbol{q}_{i,x_{\leq k}}[j] \mathbf{C}[j,:] \boldsymbol{q}_{x_{>k}} \\
&= \boldsymbol{q}_{i,x_{\leq k}} \mathbf{C} \boldsymbol{q}_{x_{>k}}
\end{aligned}
\tag{2.17}
$$

Now we show that the application of (2.15) and (2.16) to (2.8) yields the desired result:

$$
\begin{aligned}
\tilde{F}[i](x_{\leq k}, x_{>k}) &= \mathcal{C}_{\boldsymbol{y}}[i,:](x_{\leq k}) \left[\bar{\mathbf{F}}^k\right]^{\dagger} R_{\boldsymbol{\alpha}}(x_{>k}) \\
&= \boldsymbol{q}_{i,x_{\leq k}} \mathbf{R} \left[\bar{\mathbf{F}}^k\right]^{\dagger} \mathbf{C} \boldsymbol{q}_{x_{>k}} \\
&= \boldsymbol{q}_{i,x_{\leq k}} \bar{\mathbf{F}}^k \left[\bar{\mathbf{F}}^k\right]^{\dagger} \bar{\mathbf{F}}^k \boldsymbol{q}_{x_{>k}} \\
&= \boldsymbol{q}_{i,x_{\leq k}} \bar{\mathbf{F}}^k \, \boldsymbol{q}_{x_{>k}} \\
&= \boldsymbol{q}_{i,x_{\leq k}} \mathbf{C} \, \boldsymbol{q}_{x_{>k}} \\
&= F^k[i](x_{\leq k}, x_{>k}),
\end{aligned}
$$

where the first equality follows from the definition of $\tilde{F}$, the second follows from an application of (2.15) and (2.16), the third follows from (2.13), the fourth follows from the Moore-Penrose pseudoinverse (2.10), the fifth is another application of (2.13), and the final equality follows from (2.17). □

This proof shows the general strategy needed to construct a CUR decomposition of a vector-valued function when one can afford to evaluate $\ell \geq r$ row and column fibers. In this case, one must seek at r linearly independent row and column fibers to obtain an exact reconstruction. A natural question to ask, however, is what happens when one wants to obtain an approximation with $\ell < r$ for a particular separated representation F^k of F? In particular, how do we choose $\boldsymbol{y}$ and $\boldsymbol{\alpha}$ to obtain a well-

behaved approximation? The answer is to choose them so that submatrix $\bar{\mathbf{F}}^k$ lying at the intersection of the row and column fibers has *maximum volume* among all possible combinations of ℓ row and column fibers.

2.3.2 Optimality of maxvol

In this section we consider the case $\ell < r$, where we would like to use fewer terms in a skeleton approximation than the actual rank of the function. This situation arises because many functions of interest are *approximately* low-rank. Consider the SVD of a function with a possibly infinite number r of nonzero singular values, and explicitly separate the first ℓ terms from the remainder:

$$f(x,y) = \sum_{i=1}^{\ell} \sigma_i u_i(x) v_i(y) + g(x,y)\,. \tag{2.18}$$

If $\|g\|_{L^2} = \epsilon$, then f will have a relatively accurate rank ℓ approximation whenever $\epsilon/\|f\|_{L^2} \ll 1$—for instance if the singular values σ_i quickly decay to zero. Now consider the extended SVD of a vector-valued function $F^k(x_{\leq k}, x_{>k})$ of rank r. Choose an approximation rank $\ell < r$ and form a CUR decomposition defined by a set of column fibers $\boldsymbol{y}$ and a set of row fibers $\boldsymbol{\alpha}$:

$$\tilde{F}^k[i](x_{\leq k}, x_{>k}) = C_{\boldsymbol{y}}[i,:](x_{\leq k})\mathbf{G}R_{\boldsymbol{\alpha}}(x_{>k}). \tag{2.19}$$

The goal of this section is to bound $\|F - \tilde{F}\|$ and to show that this bound holds when the matrix $\mathbf{G}^\dagger$ in the skeleton decomposition is chosen to be a maximum volume submatrix of F, where the "volume" of a matrix refers to the modulus of its determinant.

Definition 13 (Maximum volume submatrix). *A submatrix $\bar{\mathbf{F}}^k$ of the k-separated representation F^k of a vector-valued function F has maximum volume if its determinant has maximum modulus among all possible sub-matrices of F^k. The maximum volume submatrix is denoted as $\bar{\mathbf{F}}^{*,k}$.*

A result analogous to the following was proven for matrices by Goreinov [51];

here, we extend it to multivariate vector-valued functions. Let the norm $\|F\|_C = \max_{i,x_{\leq k},x_{>k}} F[i](x_{\leq k}, x_{>k})$, denote the maximum-in-modulus value of the vector-valued function F. Similarly, $\|\mathbf{F}\|_C$ denotes the maximum modulus element of the matrix $\mathbf{F}$.

Theorem 2. *Let $F : \mathcal{X} \to \mathbb{R}^n$ be a continuous vector-valued function with k-separated representation F^k, and let F^k have an extended SVD with singular functions uniformly bounded by ρ, i.e., $\max_{i,j,x_{\leq k}} |U_j[i](x_{\leq k})| \leq \rho$ and $\max_{j,x_{>k}} |v_j(x_{>k})| \leq \rho$. Furthermore, let $\tilde{F}$ be a skeleton approximation of F^k comprising ℓ row and column fibers, and let the matrix $\bar{\mathbf{F}}^{*,k} \in \mathbb{R}^{\ell \times \ell}$ formed by the intersection of these fibers be a nonsingular maximum volume submatrix of F^k. Then it holds that*

$$\|F - \tilde{F}\|_C \leq M\rho^2(\ell+1)^{1+2\varepsilon}\sigma_{\ell+1} \tag{2.20}$$

where $\varepsilon > \frac{1}{2}$, M is independent of ℓ, and $\sigma_{\ell+1}$ is the $(\ell+1)$th singular value of F^k.

The proof of this theorem is provided in Appendix A.1.

Theorem 2 implies that using a maximum volume submatrix $\bar{\mathbf{F}}^{*,k}$ allows us to bound the error of the skeleton decomposition of a function by a constant factor greater than the error of the SVD, the optimal low-rank decomposition. This result is different from the result of [6], in which the error of a skeleton decomposition was related to the error of a polynomial approximation of the function. The present result refocuses the problem directly on the *rank* of the function instead of the accuracy of polynomial approximation.

2.3.3 Cross approximation and maxvol

Next, we describe an algorithm for finding a good skeleton decomposition of a vector-valued function. Computing the skeleton decomposition of a matrix is the subject of much current research. Some approaches employ random sampling of the rows and columns of the matrix [18, 42, 87]. Another class of methods attempt to explicitly find the rows and columns that maximize the volume of the submatrix. Goreinov [51, 53] describes how a skeleton decomposition with a maximum-volume submatrix is

quasioptimal. We follow this second route because it has been successfully extended to the tensor-train format [98], and subsequently shown to be quasioptimal [107]. This section focuses on extending maximum-volume cross approximation algorithms to the continuous/functional setting.

Cross approximation of a vector-valued function

Cross approximation of a vector-valued function involves a fairly straightforward row-column alternating algorithm. Pseudocode for this approach is given in Algorithm 1. This algorithm is a continuous analogue of Algorithm 2 by Oseledets and Tyrtyshnikov [98]. The algorithm works by first fixing a set of column fibers $\boldsymbol{y}$ and computing a set of rows $\boldsymbol{\alpha}$ to maximize the volume of a submatrix of the associated matrix-valued function $\mathcal{C}_{\boldsymbol{y}}$. Next, the set of row-fibers $\boldsymbol{\alpha}$ are fixed and a new set of indices for the column fibers $\boldsymbol{y}$ are identified such that they maximize the volume of a submatrix of $R_{\boldsymbol{\alpha}}$. These indices $\boldsymbol{y}$ and $\boldsymbol{\alpha}$ are found by solving *continuous* optimization problems. In other words, no discretization is required to choose the fibers, and the choice of optimization algorithm is flexible. In Section 2.3.3 we provide more detail on the how these optimization problems are solved.

The algorithm continually alternates between rows and columns until the difference between successive approximations falls below a particular tolerance. Note that this algorithm requires the prescription of an upper bound on the rank. Later, in Section 3.3.2 of Chapter 3, we describe a rank estimation scheme in the context of a multivariate cross approximation algorithm.

Algorithm 1 requires several subroutines: `qr-mvf` refers to the QR decomposition of a matrix-valued function and is computed using Householder triangularization; `qr-vvf` refers to a QR decomposition of a vector-valued function and is also computed using Householder triangularization; `maxvol-mvf` refers to Algorithm 2, discussed in Section 2.3.3.

We would like to point out the distinction between this row-column alternating algorithm, which can be viewed as an LU decomposition with *partial* pivoting, and the *adaptive* cross approximation algorithm [6]. Adaptive cross approximation algo-

Algorithm 1 Cross approximation of a vector-valued function

Require: Separated form $F^k : \mathscr{X}_{\leq k} \times \mathscr{X}_{>k} \to \mathbb{R}^r$ of a vector-valued function F; Upper bound on rank r; Initial column fibers $\boldsymbol{y} = [y_1, y_2, \ldots, y_r]$; Stopping tolerance $\delta_{\text{cross}} > 0$

Ensure: $(\boldsymbol{\alpha}, \boldsymbol{y})$ such that sub-matrix $\mathbf{F}$ has " large " volume

$F_{(0)} = 0$
$k = 1$
$\mathcal{C}(x_{\leq k}) = [F^k(x_{\leq k}, y_1)\ F^k(x_{\leq k}, y_2)\ \ldots\ F^k(x_{\leq k}, y_r)]$
repeat
  $\mathcal{Q}\mathbf{T} = \texttt{qr-mvf}(\mathcal{C})$
  $\boldsymbol{\alpha} = \texttt{maxvol-mvf}(\mathcal{Q})$
  $R(x_{>k}) = [F^k[i_1](z_1, x_{>k})\ \ldots\ F^k[i_r](z_r, x_{>k})]$ where $(i_j, z_j) = \boldsymbol{\alpha}[j]$
  $U\mathbf{T} = \texttt{qr-vvf}(R)$
  $\boldsymbol{y} = \texttt{maxvol-mvf}(U)$ # Interpret U as a $1 \times r$ matrix-valued function.
  $\mathcal{C}(x_{\leq k}) = [F^k(x_{\leq k}, y_1)\ F^k(x_{\leq k}, y_2)\ \ldots\ F^k(x_{\leq k}, y_r)]$
  $\hat{\mathbf{Q}} = [U(y_1)\ U(y_2)\ \ldots\ U(y_r)]$
  $F_{(k)}(x_{\leq k}, x_{>k}) = \mathcal{C}(x_{\leq k})\hat{\mathbf{Q}}^{\dagger}U(x_{>k})$
  $\delta = ||F_{(k)} - F_{(k-1)}||/||F_{(k)}||$
  $k = k + 1$
until $\delta \leq \delta_{\text{cross}}$

Algorithm 2 `maxvol-mvf`: Maximum volume of matrix-valued function

Require: $\mathcal{A} : \mathscr{X} \to \mathbb{R}^{n \times r}$, a matrix-valued function.

Ensure: $\boldsymbol{\alpha} = [(i_1, x_1,) \ldots (i_r, x_r)]$ such that $\bar{\mathbf{A}}_{\text{mat}}$ is dominant

$L, \mathbf{U}, \boldsymbol{\alpha} = \texttt{lu-mvf}(A)$ # LU decomposition of the matrix-valued function
$\delta = 2$
while $\delta > 1$ **do**
  $\bar{\mathbf{A}}_{\text{mat}} \leftarrow$ sub-matrix defined by by $\boldsymbol{\alpha}$
  $x^*, i^*, j^* = \underset{(x,i,j)}{\arg\max}\ \mathcal{A}[i, :](x)\bar{\mathbf{A}}^{\dagger}_{\text{mat}}[:, j]$
  $\delta = \mathcal{A}[i^*, :](x^*)\bar{\mathbf{A}}^{\dagger}_{\text{mat}}[:, j^*]$
  if $\delta > 1$ **then**
    $x_{j^*} = x^*$, $i_{j^*} = i^*$
  end if
end while

rithms for the approximation of bivariate functions build up a set of row and column indices by sequentially choosing points which maximize the volume. In other words, they do not require a prescribed rank, nor do they require evaluating entire function fibers. These algorithms, which are equivalent to an LU decomposition with *complete* pivoting, can use two-dimensional optimization methods to seek the optimal locations

for function evaluation (which become the pivots). This methodology is attractive for bivariate problems but difficult to extend to the multivariate case, because it would require us to optimize over locations in d dimensions. To retain good scaling properties, we use the row-column alternating algorithm that is standard for computing with tensors.

Maxvol computation

Algorithm 1 contains calls to `maxvol-mvf` with two types of arguments, matrix-valued functions and vector-valued functions. In both cases, the goal of `maxvol-mvf` is to find indices that describe a maximum volume submatrix. Pseudocode for computing the maximum volume submatrix of a matrix-valued function is given in Algorithm 2; it mirrors the algorithm provided by Goreinov [50] for "tall and skinny" matrices. The algorithm attempts to find a submatrix which is *dominant*, because a dominant submatrix has volume which is "close" to that of the maximum volume submatrix, as described below. In this section we will always assume that a matrix-valued function $\mathcal{A} : \mathscr{X} \to \mathbb{R}^{n\times r}$ has rank r according to the notion of rank defined by the QR decomposition in Section 2.2. All of the results below can be specialized to vector-valued functions by assuming that $n = 1$.

Definition 14 (Submatrix of a matrix-valued function). *A submatrix of a matrix-valued function $\mathcal{A} : \mathscr{X} \to \mathbb{R}^{n\times r}$ is the matrix $\bar{\mathbf{A}}_{\text{mat}} \in \mathbb{R}^{r\times r}$ obtained by fixing a set of r tuples $\{(i_1, x_1), (i_2, x_2), \ldots, (i_r, x_r)\}$, where $(i_k, x_k) \in \{1, \ldots, n\} \times \mathscr{X}$ for $k, l = 1 \ldots r$ and $(i_k, x_k) \neq (i_l, x_l)$ for $k \neq l$, such that*

$$\bar{\mathbf{A}}_{\text{mat}}[k, j] = \mathcal{A}[i_k, j](x_k). \tag{2.21}$$

Note the distinction between Definitions 11 and 14: the former defines submatrices of vector-valued functions in separated form F^k, whereas Definition 14 defines submatrices of matrix-valued functions.

While the results below consider the general case of matrix-valued functions, it is useful to keep in mind how they arise in the context of cross approximation. In that

specific context, the matrix-valued function of interest will be a set of column fibers. For example, after finding a set of r column fibers $C_{\boldsymbol{y}} : \mathscr{X}_{\leq k} \to \mathbb{R}^{n \times r}$, we seek new values of $(i, x_{\leq k}) \in \{1, \ldots, n\} \times \mathscr{X}_{\leq k}$ that form the row fibers. These new indices also define a submatrix of $C_{\boldsymbol{y}}$, and we seek to choose those indices that maximize the volume of that submatrix.

Next, we need the notion of a *dominant submatrix* of a matrix valued function.

Definition 15 (Dominant submatrix). *A dominant submatrix of a matrix-valued function $\mathcal{A} : \mathscr{X} \to \mathbb{R}^{n \times r}$ is any submatrix $\bar{\mathbf{A}}_{\text{mat}}$ such that for all values $(i, x, k) \in \{1, \ldots, n\} \times \mathscr{X} \times \{1, \ldots, r\}$ the matrix-valued function $\mathcal{B} = \mathcal{A}\bar{\mathbf{A}}_{\text{mat}}^{-1}$ is bounded as $|\mathcal{B}[i, k](x)| \leq 1$.*

Lemma 1. *A maximum-volume submatrix $\bar{\mathbf{A}}^*_{\text{mat}} \in \mathbb{R}^{r \times r}$ of a matrix-valued function $\mathcal{A} : \mathscr{X} \to \mathbb{R}^{n \times r}$ is dominant.*

Proof. The proof follows exactly the case for matrices [50]. Here we interpret a matrix-valued function as a matrix with r columns and an infinite number of rows, where each row is indexed by $(i, x) \in \{1, \ldots, n\} \times \mathscr{X}$.

Suppose, without loss of generality, that we first rearrange the "rows" of $\mathcal{A}$ such that

$$\mathcal{A}\left[\bar{\mathbf{A}}^*_{\text{mat}}\right]^{-1} = \begin{bmatrix} \mathbf{I}_{r \times r} \\ \mathcal{Z} \end{bmatrix} = \mathcal{B},$$

where $\bar{\mathbf{A}}^*_{\text{mat}}$ is a maximum-volume submatrix of $\mathcal{A}$. In other words, if we view $\mathcal{A}$ as an "$\infty \times r$" matrix, we have moved the rows corresponding to $\bar{\mathbf{A}}^*_{\text{mat}}$, with indices $\{(i_1, x_1), \ldots (i_r, x_r)\}$, to the "top" of $\mathcal{A}$. As a result, the first r rows of $\mathcal{B}$ correspond to the identity matrix, and the remaining rows are indexed by (i, x) pairs (but with each (i_j, x_j), for $j = 1 \ldots r$, missing).

Now recall that $\det(\mathbf{CD}) = \det(\mathbf{C}) \det(\mathbf{D})$ for square matrices $\mathbf{C}, \mathbf{D}$ of equal size. This property implies that multiplying $\mathcal{A}$ by a nonsingular $r \times r$ matrix does not change the ratio between the determinants of any pair of $r \times r$ submatrices of $\mathcal{A}$. Therefore the upper submatrix $\mathbf{I}_{r \times r}$ is a maximum-volume submatrix of $\mathcal{B}$.

Suppose $\bar{\mathbf{A}}^*_{\text{mat}}$ is not dominant so that there is an (i, j, x) where $|\mathcal{Z}[i, j](x)| > 1$. In such a situation, we would be able to increase the volume of the upper submatrix by swapping row j of the upper $r \times r$ submatrix of $\mathcal{B}$ with $\mathcal{Z}[i, :](x)$. The new upper submatrix of $\mathcal{B}$, called $\mathbf{B}_{upper}$, would no longer be the identity $\mathbf{I}_{r\times r}$; we would have $\det(\mathbf{B}_{upper}) = \mathcal{Z}[i, j](x) > 1$ instead of $\det(\mathbf{I}_{r\times r}) = 1$. Then $\mathbf{I}_{r\times r}$ would not be a maximum volume submatrix of $\mathcal{B}$ and hence $\bar{\mathbf{A}}^*_{\text{mat}}$ would not be a maximum volume submatrix of $\mathcal{A}$. □

Furthermore, analogous to Lemma 2 by Goreinov [50] we can show that any dominant submatrix cannot have a volume that is too much smaller than that of the maximum-volume submatrix.

Lemma 2. *For any full rank matrix-valued function $\mathcal{A} : \mathscr{X} \to \mathbb{R}^{n\times r}$,*

$$\left|\det(\bar{\mathbf{A}}_{\text{mat}})\right| \geq \frac{\left|\det(\bar{\mathbf{A}}^*_{\text{mat}})\right|}{r^{r/2}},$$

where $\bar{\mathbf{A}}_{\text{mat}}$ is any dominant submatrix and $\bar{\mathbf{A}}^*_{\text{mat}}$ is the submatrix of $\mathcal{A}$ with maximum volume.

The proof is almost equivalent to that in [50] and is omitted here for brevity. Of course, we also have the upper bound, $|\det(\bar{\mathbf{A}}^*_{\text{mat}})| \geq |\det(\bar{\mathbf{A}}_{\text{mat}})|$.

Together, Lemmas 1 and 2 imply that searching for a maximum volume submatrix of a matrix-valued function $\mathcal{A}$ via the row switching scheme described in Algorithm 2 is a good idea. The algorithm swaps "rows" of $\mathcal{A}$ (recall that these are specified by (index, x-value) combinations) until all the elements of $\mathcal{B}$ are less than 1. This is exactly what the operations in Lines 5 and 8 of Algorithm 2 are doing. To find an initial set of linearly independent "rows" of $\mathcal{A}$, the algorithm first performs a continuous pivoted LU decomposition, denoted by `lu-mvf`, yielding pivots $\boldsymbol{\alpha} = \{(i_1, x_1), \ldots, (i_r, x_r)\}$.

The optimization problem specified in Line 5 of Algorithm 2 is where we obtain tremendous benefits over the discretized tensor-train approach. First, it is a *continuous* optimization problem in x, allowing us to search over the entire space $\mathscr{X}$; in the discretized version, this maximization can only occur over the discretized points.

Secondly, when $\mathcal{A} : [a, b] \to \mathbb{R}^{n \times r}$, the continuous optimization problem involves a one-dimensional decision variable and can exploit the structure of the scalar-valued functions that comprise $\mathcal{A}$. For example, if these scalar-valued functions are represented as orthonormal polynomials, then the maximization reduces to an eigenvalue problem [35].

2.4 Operations with functions expressed in an orthonormal basis

The algorithms described in this chapter were all provided in a general form. However, for practical purposes we need to be able to add, multiply, and take inner products of functions in a parameterized format. In this section, we provide the details for a sample implementation of a framework in which all functions are represented as an expansion in an orthonormal basis. Suppose that we define a complete basis $\{\phi_1(x), \phi_2(x), \ldots\}$ such that $\langle \phi_i(x), \phi_j(x) \rangle = \delta_{i,j}$ for all i, j.

We describe the complexity of the algorithms using asymptotic notation represented by Landau's symbol $\mathcal{O}$.

Definition 16 (Asymptotic notation with Landau's symbol). *We write* $f(x) = \mathcal{O}(g(x))$ *as* $x \to \infty$ *if and only if for every* $c > 0$ *there exists a real number* N *such that for all* $x > N$ *we have* $|f(x)| \leq c|g(x)|$*; if* $g(x) \neq 0$*, this is equivalent to* $\lim_{x \to \infty} f(x)/g(x) = 0$.

Next, we detail the main operations, and their complexity, for scalar-valued, vector-valued, and matrix-valued functions.

2.4.1 Scalar-valued functions

Let $f : \mathcal{X} \to \mathbb{R}$ and $g : \mathcal{X} \to \mathbb{R}$ refer to scalar-valued functions, and suppose that both of these functions are represented using the P basis functions with coefficients

$\boldsymbol{f} \in \mathbb{R}^p$ and $\boldsymbol{g} \in \mathbb{R}^P$ respectively

$$f(x) = \sum_{i=1}^{P} \boldsymbol{f}[i]\phi_i(x)$$
$$g(x) = \sum_{i=1}^{P} \boldsymbol{g}[i]\phi_i(x)$$

Addition of functions is of primary importance for computing with functions.

Proposition 1 (Addition of scalar-valued functions). *Let $f : \mathscr{X} \to \mathbb{R}$ and $g : \mathscr{X} \to \mathbb{R}$ be represented in an expansion of P orthonormal basis functions. Then, addition $f(x) + g(x)$ requires $\mathcal{O}(P)$ operations.*

Proof.

$$f(x) + g(x) = \sum_{i=1}^{P} \boldsymbol{f}[i]\phi_i(x) + \sum_{j=1}^{P} \boldsymbol{g}[j]\phi_j(x) = \sum_{i=1}^{P} \left(\boldsymbol{f}[i] + \boldsymbol{g}[i]\right)\phi_i(x)$$

Addition of P coefficients costs $\mathcal{O}(P)$. □

Proposition 2 (Multiplication of scalar-valued functions). *Let $f : \mathscr{X} \to \mathbb{R}$ and $g : \mathscr{X} \to \mathbb{R}$ be represented in an expansion of P orthonormal basis functions. Assume that the third order tensor with entries $\int \phi_i(x)\phi_j(x)\phi_k(x)dx$ for $i, j, k = 1, \ldots P$ is pre-computed and that its values can be accessed in $\mathcal{O}(1)$ operations. Then, multiplication $f(x)g(x)$ requires $\mathcal{O}(P^3)$ operations.*

Proof. Let $h(x) = f(x)g(x)$ and notice that $h(x)$ has an exact expansion in terms of $2P$ polynomials with coefficients $\boldsymbol{h} \in \mathbb{R}^{2P}$. Then we have

$$h(x) = \sum_{k=1}^{2P} \boldsymbol{h}[k]\phi_k(x) = \sum_{i,j=1}^{P} \boldsymbol{f}[i]\boldsymbol{g}[j]\phi_i(x)\phi_j(x)$$

Each coefficient $\boldsymbol{h}[k]$ can be obtained by projecting the right side of the equation above onto ϕ_k

$$\boldsymbol{h}[k] = \sum_{i,j=1}^{P} \boldsymbol{f}[i]\boldsymbol{g}[j] \int \phi_i(x)\phi_j(x)\phi_k(x)dx$$

The computation of this coefficient first requires the multiplication of all combinations of the pair $(\boldsymbol{f}[i], \boldsymbol{g}[j])$, an $\mathcal{O}(P^2)$ operation. This computation can be reused for all the coefficients $\boldsymbol{h}[k]$, and is therefore not the dominant cost. Next, for each $k = 1, \ldots, 2P$, the summation of the P^2 terms can be performed in $\mathcal{O}(P^2)$. Thus, the total cost is $\mathcal{O}(2P^3) = \mathcal{O}(P^3)$ operations. □

The inner product is important for defining the QR decomposition.

Proposition 3 (Inner product between scalar-valued functions). *Let $f : \mathscr{X} \to \mathbb{R}$ and $g : \mathscr{X} \to \mathbb{R}$ be represented in an expansion of P orthonormal basis functions. Then the inner product $\langle f(x), g(x) \rangle$ requires $\mathcal{O}(P)$ operations.*

Proof.

$$\begin{aligned}
\langle f(x), g(x) \rangle &= \int \sum_{\ell=1}^{P} \boldsymbol{f}[\ell]\phi_\ell(x) \sum_{j=1}^{P} \boldsymbol{g}[j]\phi_j(x)dx \\
&= \sum_{\ell,j=1}^{P} \boldsymbol{f}[\ell]\boldsymbol{g}[j] \int \phi_\ell(x)\phi_j(x)dx \\
&= \sum_{\ell=1}^{P} \boldsymbol{f}[\ell]\boldsymbol{g}[\ell]
\end{aligned}$$

Multiplication of each of these coefficients costs $\mathcal{O}(P)$ operations, and adding these coefficients costs $\mathcal{O}(P)$ operations yielding a total of $\mathcal{O}(2P) = \mathcal{O}(P)$ operations. □

2.4.2 Vector-valued functions

Next, we consider vector-valued functions. Let $F : \mathscr{X} \to \mathbb{R}^n$ and $G : \mathscr{X} \to \mathbb{R}^n$ refer to vector-valued functions consisting of scalar-valued functions $f_i : \mathscr{X} \to \mathbb{R}$ and $g_i : \mathscr{X} \to \mathbb{R}$, respectively, such that

$$F[i] = f_i = \sum_{\ell=1}^{P} \boldsymbol{f}_i[\ell]\phi_\ell, \tag{2.22}$$

$$G[i] = g_i = \sum_{\ell=1}^{P} \boldsymbol{g}_i[\ell]\phi_\ell. \tag{2.23}$$

The two most widely used operations needed with vector-valued functions will be addition and taking the inner product.

Proposition 4 (Addition of vector-valued functions). *Let $F : \mathscr{X} \to \mathbb{R}^n$ and $G : \mathscr{X} \to \mathbb{R}^n$ be defined according to* (2.22) *and* (2.23)*, respectively. Then addition $F(x)+G(x)$ requires $\mathcal{O}(Pn)$ operations.*

Proof.

$$F(x) + G(x) = [(f_1(x) + g_1(x)) \ \ldots \ (f_n(x) + f_n(x))]$$

Addition of n scalar-valued functions costs $\mathcal{O}(Pn)$. □

Proposition 5 (Inner product between vector-valued functions). *Let $F : \mathscr{X} \to \mathbb{R}^n$ and $G : \mathscr{X} \to \mathbb{R}^n$ be defined according to* (2.22) *and* (2.23)*, respectively. Then the inner product $\langle F(x), G(x) \rangle$ requires $\mathcal{O}(Pn)$ operations.*

Proof.

$$\langle F(x), G(x) \rangle = \sum_{\ell=1}^{n} \langle f_\ell(x), g_\ell(x) \rangle$$

Each scalar-valued function inner product costs $\mathcal{O}(P)$ operations (Proposition 3), n inner products have to be computed, and n summations must be performed leading to a complexity of $\mathcal{O}(Pn + n) = \mathcal{O}(Pn)$. □

2.4.3 Matrix-valued functions

Next, we consider matrix-valued functions. Let $\mathcal{F} : \mathscr{X} \to \mathbb{R}^{n \times m}$ and $G : \mathscr{X} \to \mathbb{R}^{n \times m}$ refer to matrix-valued functions consisting of scalar-valued functions $f_{i,j} : \mathscr{X} \to \mathbb{R}$ and $g_{i,j} : \mathscr{X} \to \mathbb{R}$, respectively, such that

$$\mathcal{F}[i, j] = f_{i,j} = \sum_{\ell=1}^{P} \boldsymbol{f}_{i,j}[\ell] \phi_\ell, \tag{2.24}$$

$$\mathcal{G}[i, j] = g_{i,j} = \sum_{\ell=1}^{P} \boldsymbol{g}_{i,j}[\ell] \phi_\ell. \tag{2.25}$$

Throughout this dissertation, considerably more operations are performed using matrix-valued functions than using vector-valued functions. Thus, in addition to addition, multiplication, and inner product, we also discuss integration of products of matrix-valued functions, Kronecker products, and vector-times-Kronecker products.

Proposition 6 (Addition of matrix-valued functions). *Let $\mathcal{F} : \mathscr{X} \to \mathbb{R}^{m \times n}$ and $\mathcal{G} : \mathscr{X} \to \mathbb{R}^{m \times n}$ be defined according to* (2.24) *and* (2.25)*, respectively. Then, addition $\mathcal{F}(x) + \mathcal{G}(x)$ requires $\mathcal{O}(Pnm)$ operations.*

Proof.

$$\mathcal{F}(x) + \mathcal{G}(x) = \begin{bmatrix} f_{1,1}(x) + g_{1,1}(x) & \dots & f_{1,m}(x) + g_{1,m}(x) \\ \vdots & & \vdots \\ f_{n,1}(x) + g_{n,1}(x) & \dots & f_{n,m}(x) + g_{n,m}(x) \end{bmatrix}$$

Addition of nm scalar-valued functions costs $\mathcal{O}(Pnm)$. □

Proposition 7 (Multiplication of matrix-valued functions). *Let $\mathcal{F} : \mathscr{X} \to \mathbb{R}^{m \times r}$ and $\mathcal{G} : \mathscr{X} \to \mathbb{R}^{r \times n}$ be defined according to* (2.24) *and* (2.25)*, respectively. Assume that the third order tensor with entries $\int \phi_i(x)\phi_j(x)\phi_k(x)dx$ for $i, j, k = 1, \dots P$ is precomputed and that its values can be accessed in $\mathcal{O}(1)$ operations. Then, multiplication $\mathcal{F}(x)\mathcal{G}(x)$ requires $\mathcal{O}(P^3rmn)$ operations.*

Proof. Let $\mathcal{H} : \mathscr{X} \to \mathbb{R}^{m \times n}$ such that for $i = 1, \dots, m$ and $j = 1, \dots n$

$$\begin{aligned} \mathcal{H}[i,j](x) &= \sum_{k=1}^{r} \mathcal{F}[i,k](x)\mathcal{G}[k,j](x) \\ &= \sum_{k=1}^{r} \sum_{\ell=1}^{P} \boldsymbol{f}_{i,k}[\ell]\phi_\ell(x) \sum_{\ell'=1}^{P} \boldsymbol{g}_{k,j}[\ell']\phi_{\ell'}(x) \\ &= \sum_{\ell=1}^{P} \sum_{\ell'=1}^{P} \phi_\ell(x)\phi_{\ell'}(x) \sum_{k=1}^{r} \boldsymbol{f}_{i,k}[\ell]\boldsymbol{g}_{k,j}[\ell'] \\ &= \sum_{\ell=1}^{P} \sum_{\ell'=1}^{P} \phi_\ell(x)\phi_{\ell'}(x)\ \hat{\mathbf{D}}_{i,j}[\ell,\ell'] \end{aligned}$$

where the matrix $\hat{\mathbf{D}}_{i,j} \in \mathbb{R}^{P \times P}$ represents the coefficients of $\mathcal{D}[i,j]$ in a "double" basis consisting of products of ϕ_ℓ and $\phi_{\ell'}$. Each element of this matrix can be computed through the sum of r coefficients $\boldsymbol{f}_{i,k}$ and $\boldsymbol{g}_{k,j}$ in $\mathcal{O}(r)$ operations.

We project this expansion back onto a single basis. The single basis now has maximum order $2P$ and be represented as

$$\mathcal{D}[i,j](x) = \sum_{l=1}^{2P} \boldsymbol{d}_{i,j}[l]\phi_l(x)$$

This computation is performed by projecting the double expansion onto ϕ_l for $l = 1, \ldots, P$

$$\boldsymbol{d}_{i,j}[l] = \sum_{\ell=1}^{P}\sum_{\ell'=1}^{P} \int \phi_\ell(x)\phi_{\ell'}(x)\phi_l(x)dx \; \hat{\mathbf{D}}_{i,j}[\ell,\ell']$$

If the triple product is precomputed and stored in memory, then each coefficient $\boldsymbol{d}_{i,j}[l]$ requires $\mathcal{O}(P^2 r)$ operations. Since there are $2P$ coefficients, the total cost for computing $\mathcal{D}[i,j](x)$ is $\mathcal{O}(P^3 r)$ since there are mn such functions the final complexity of the computation is $\mathcal{O}(P^3 rmn)$. □

Proposition 8 (Inner product between matrix-valued functions). *Let $\mathcal{F} : \mathscr{X} \to \mathbb{R}^{m \times n}$ and $\mathcal{G} : \mathscr{X} \to \mathbb{R}^{m \times n}$ be defined according to* (2.24) *and* (2.25)*, respectively. Then, the inner product $\langle \mathcal{F}(x), \mathcal{G}(x) \rangle$ requires $\mathcal{O}(Pnm)$ operations.*

Proof.

$$\langle \mathcal{F}(x), \mathcal{G}(x) \rangle = \sum_{\ell=1}^{n}\sum_{\ell'}^{m} \langle f_{\ell,\ell'}(x), g_{\ell,\ell'}(x) \rangle$$

Each scalar-valued function inner product costs $\mathcal{O}(P)$ operations (Proposition 3), nm inner products and a summation of nm elements must be computed, leading to a complexity of $\mathcal{O}(Pnm + nm) = \mathcal{O}(Pnm)$. □

Next we consider additional operations between matrix-valued functions and matrices that will, especially, be useful for designing efficient alternating least squares

algorithms in Section 3.5.

Proposition 9 (Multiplication of a matrix-valued function and a matrix). *Let* $\mathcal{F} : \mathscr{X} \to \mathbb{R}^{n\times m}$ *be defined according to* (2.24) *and let* $\mathbf{A} \in \mathbb{R}^{m\times r}$. *Multiplying* $\mathcal{F}\mathbf{A}$ *requires* $\mathcal{O}(Pnmr)$ *operations.*

Proof. Let $\mathcal{G}(x) = \mathcal{F}(x)\mathbf{A}$ be a matrix-valued function such that $\mathcal{G} : \mathscr{X} \to \mathbb{R}^{n\times r}$.

$$\begin{aligned}
\mathcal{G}[i,j](x) &= \sum_{k=1}^{n} \mathcal{F}[i,k](x)\mathbf{A}[k,j] \\
&= \sum_{k=1}^{n}\sum_{\ell=1}^{P} \boldsymbol{f}_{i,k}[\ell]\phi_\ell(x)\mathbf{A}[k,j] \\
&= \sum_{\ell=1}^{P}\left(\sum_{k=1}^{n} \boldsymbol{f}_{i,k}[\ell]\mathbf{A}[k,j]\right)\phi_\ell(x) \\
&= \sum_{\ell=1}^{P} \boldsymbol{g}_{i,j}[\ell]\phi_\ell(x)
\end{aligned}$$

where as set of P coefficients $\boldsymbol{g}_{i,j}[\ell] = \sum_{k=1}^{n} \boldsymbol{i},\boldsymbol{k}[\ell]\mathbf{A}[k,j]$ for $\ell = 1 \ldots P$ has to be computed at a cost of $\mathcal{O}(Pn)$ operations for each element i,j. Thus, the total cost is $\mathcal{O}(Pnmr)$ operations. □

Proposition 10 (Integration of matrix-valued function products). *Let* $\mathcal{F} : \mathscr{X} \to \mathbb{R}^{m\times n}$ *and* $\mathcal{G} : \mathscr{X} \to \mathbb{R}^{m\times n}$ *be defined according to* (2.24) *and* (2.25), *respectively. Then, integration of the product* $\int \mathcal{F}(x)\mathcal{G}(x)dx$ *requires* $\mathcal{O}(Pmnr)$ *operations.*

Proof. Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ be such that $\mathbf{A} = \int \mathcal{F}(x)\mathcal{G}(x)dx$. Then for $i = 1,\ldots,n$ and $j = 1,\ldots,m$ we have

$$\begin{aligned}
\mathbf{A}[i,j] &= \int \sum_{k=1}^{r} \mathcal{F}[i,k](x)\mathcal{G}[k,j]dx \\
&= \sum_{k=1}^{r} \int \sum_{\ell=1}^{P}\sum_{\ell'=1}^{P} \boldsymbol{f}_{i,k}[\ell]\boldsymbol{g}_{k,j}[\ell']\phi_\ell(x)\phi_{\ell'}(x)dx \\
&= \sum_{k=1}^{r}\sum_{\ell=1}^{p} \boldsymbol{f}_{i,k}[\ell]\boldsymbol{g}_{k,j}[\ell']
\end{aligned}$$

Thus the cost of computing each element of $\mathbf{A}$ is $\mathcal{O}(Pr)$ yielding a total cost of $\mathcal{O}(Pmnr)$. □

Finally, we describe fast operations for multiply a Kronecker product of matrices by a vector and a matrix. These operations will be critical for taking the inner product between continuous low-rank decompositions and for alternating least squares in Chapter 3.

Proposition 11. *Let* $\boldsymbol{c} \in \mathbb{R}^{1\times nk}$, $\mathcal{F} : \mathscr{X} \to \mathbb{R}^{n\times m}$, *and* $\mathcal{G} : \mathscr{X} \to \mathbb{R}^{k\times l}$. *Then computing*

$$\boldsymbol{c}(\mathcal{F} \otimes \mathcal{G}) \tag{2.26}$$

requires

$$\mathcal{O}(Pknm + P^3nml)$$

operations.

Proof. The computation can be broken into two steps. First, $\boldsymbol{c}$ is reshaped into an $n \times k$ matrix $\mathbf{C}$, then product $\mathcal{D}(x) = \mathbf{C}\mathcal{G}(x)$, where $\mathcal{D} : \mathscr{X} \to \mathbb{R}^{n\times l}$, is computed in $\mathcal{O}(Pknm)$ operations according to Proposition 9. Then after an appropriate reordering of the scalar-valued functions making up $\mathcal{D}$, we can obtain a new matrix-valued function $\hat{\mathcal{D}} : \mathscr{X} \to \mathbb{R}^{l\times n}$, and we compute $\hat{\mathcal{D}}\mathcal{F}$ in $\mathcal{O}(P^3nml)$ according to Proposition 7. The final result is then obtained by flattening this product into vector-valued that maps $\mathscr{X} \to \mathbb{R}^{ml}$. □

Proposition 12. *Let* $\mathbf{C} \in \mathbb{R}^{r\times nk}$, $\mathcal{F} : \mathscr{X} \to \mathbb{R}^{n\times m}$, *and* $\mathcal{G} : \mathscr{X} \to \mathbb{R}^{k\times l}$. *Then computing*

$$\mathbf{C}(\mathcal{F} \otimes \mathcal{G}) \tag{2.27}$$

requires

$$\mathcal{O}(r\left(Pknm + P^3nml\right))$$

operations.

Proof. The proof simply applies Proposition 11 to each row of $\mathbf{C}$. □

Chapter 3

Function-train decomposition

To compute with multivariate functions it is important to represent them in a scalable manner that tackles the curse of dimensionality. Specifically, we seek to represent multivariate functions in a format whose storage cost grows polynomially, rather than exponentially, with dimension. Furthermore, we pursue representations and associated algorithms that scale with the *intrinsic* complexity associated with a function's "rank," rather than with the number of inputs. Consider the example of representing the function $f(x_1, \ldots, x_d) = x_1$ as d increases. Since this function is only a function of the first variable, expressing this function on a computer should not take an exponentially growing amount computational resources.

The notion of the SVD rank of a function of two variables, discussed in Chapter 2, does not readily extend to the multivariate case. Instead, we will borrow the notion of rank from the literature on tensor decompositions. In essence, we view low-rank decompositions of functions as *continuous* extensions of low-rank decompositions of tensor.

In this chapter, we present algorithms for converting and computing with functions in a low-rank format called the *function-train* (FT) decomposition. In particular, we describe an algorithm that, under certain conditions, provably converges to an FT representation of the function with a prescribed accuracy after a finite number of function evaluations. We specifically distinguish this notion from approximation of a function in a data fit context. In data fit problems, one is given a set of data and is

tasked to build a functional representation for it. In those cases, one cannot create an algorithm that can even attempt to guarantee convergence of the approximation to the "true" function that generated the data without the ability to obtain more data. Our goal, on the other hand, is to develop a framework for building algorithms that can accurately approximate multivariate functions. For low-rank functional approximation in the context of data-fit problems we refer to the recent work by Doostan [40], Chevreuil et al. [26], and the dissertation by Rai [103].

In Section 3.1 we provide background on tensor decompositions. In Section 3.2 we provide conditions under which an FT decomposition achieves a certain accuracy. In Section 3.3, we describe both how to perform cross approximation to accurately decompose a function into its FT representation and how to use rounding for reapproximating a function in FT format to a certain accuracy with smaller FT ranks. In Section 3.4 we describe how to perform multilinear algebra in FT format, and in Section 3.5 we describe examples when these operations can be performed faster with a continuous and rank-revealing alternating least squares. Finally, in Section 3.6 we provide examples involving function approximation and integration.

3.1 Discrete tensor decompositions

A tensor is a multidimensional d-way array, i.e., a first order tensor is a vector, a second order tensor is a matrix, etc. Tensor *decompositions* extend ideas of SVD based reduction to multiple dimensions and allow the complexity and structure of the problem to guide computational cost rather than the dimension. Generalizations of the SVD to multiway arrays are not straightforward, and as a result, many types of tensor formats have been created. In this section, we review the basics of tensor decompositions to put into context the continuous analogue that we develop. For a broad overview of tensor computation and discussion of various tensor decompositions we refer to [29, 60, 61, 75].

Some of the first tensor decompositions [65, 66] involved approximating a tensor through finite sum of r outer products of vectors. The method is now called the

canonical decomposition [23] or the parallel factors (PARAFAC) [63] decomposition. Consider a tensor $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ with mode sizes $n_k \in \mathbb{Z}^+$. The canonical decomposition of the tensor represents each element of the tensor as

$$\boldsymbol{\mathcal{F}}[i_1, i_2, \ldots, i_d] = \sum_{\alpha=1}^{r} \mathbf{U}_1[i_1, \alpha]\mathbf{U}_2[i_2, \alpha] \ldots \mathbf{U}_d[i_d, \alpha] \tag{3.1}$$

where r is called the canonical rank and the matrices $\mathbf{U}_k \in \mathbb{R}^{n_k \times r}$ are called the canonical factors. This representation has a storage capacity that is linear in dimension d, mode size n, and rank r. For simplicity we have assumed equal mode sizes $n_k = n$ for all k. One issue with this decomposition is that there is no finite algorithm for determining the rank of a tensor, as it is NP complete [64, 76]. Another problem is that, for fixed rank, the problem of finding the best approximation in the Frobenius norm can be ill-posed [36].

Another tensor decomposition that has better properties is the Tucker decomposition [82, 122]. This representation decomposes a tensor into a small core tensor multiplied by a matrix along each mode. It is written as

$$\boldsymbol{\mathcal{F}}[i_1, i_2, \ldots, i_d] = \sum_{\alpha_1=1}^{r_1} \sum_{\alpha_2=1}^{r_2} \ldots \sum_{\alpha_d=1}^{r_d} \boldsymbol{\mathcal{G}}[\alpha_1, \ldots, \alpha_d]\mathbf{U}_1[i_1, \alpha_1]\mathbf{U}_2[i_2, \alpha_2] \ldots \mathbf{U}_d[i_d, \alpha_d], \tag{3.2}$$

where r_k are called the Tucker ranks, the small tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{r_1 \times \ldots \times r_d}$ is called the Tucker core, and $\mathbf{U}_k \in \mathbb{R}^{n_k \times \alpha_k}$ are called the Tucker factors. Techniques for computation of this format are stable; however, because the Tucker core is an order d tensor, it has exponential storage cost with dimension, $\mathcal{O}\left(dnr + r^d\right)$, where we again assume $n_k = n$ and $r_k = r$.

An instructive interpretation of the Tucker decomposition is as a subspace approximation [62]; the Tucker core $\boldsymbol{\mathcal{G}}$ describes the coefficients of $\boldsymbol{\mathcal{F}}$ with respect to basis vectors $\mathbf{U}_k[:, i]$ spanning certain subspaces. The notion of subspace approximations is an important way in which different kinds of tensor decompositions can be understood. Consider a tensor product space $\mathscr{X} = \mathscr{X}_1 \times \ldots \times \mathscr{X}_d$ where $\mathscr{X}_k \subset \mathbb{R}^{n_k}$ such that a tensor $\boldsymbol{\mathcal{F}}$ is an element of this space $\boldsymbol{\mathcal{F}} \in \mathscr{X}$. Now suppose that we take

subspaces of each set $\mathscr{X}_1$, denoted as $\mathscr{Y}_k \subset \mathscr{X}_k$, where each subspace $\mathscr{Y}_k$ is spanned by an orthonormal basis $\{\mathbf{U}_k[:,\alpha] : 1 \leq \alpha \leq \dim \mathscr{Y}_k\}$. Then, a Tucker representation of the tensor $\boldsymbol{\mathcal{F}} \in \mathscr{X}$ is defined by the subspaces $\mathscr{Y}_k$ so that $\boldsymbol{\mathcal{F}} \in \mathscr{Y} = \mathscr{Y}_1 \times \ldots \times \mathscr{Y}_d$, and the Tucker factors $\mathbf{U}_k$ are matrices whose columns are the collection of basis vectors spanning the corresponding dimension's subspace. In this situation, it is clear that the $\boldsymbol{\mathcal{G}}$ represents the coefficients of all the possible combination of basis functions. The $\mathcal{O}(r^d)$ storage cost arises because *all combinations* of the basis vectors are considered.

To reduce this storage expense, Hackbusch and Kühn [62] and Grasedyck [59] developed the notion of a hierarchy of subspaces that describe the interaction between *groups* of variables, instead of considering interactions between all of them. In this way, the exponential cost with respect to tensor rank can be reduced. The resulting Hierarchical Tucker (HT) format is a flexible generalization of tensor formats that allows for any groupings of variables. Such groupings are essentially a way to take advantage of the *separability* of the problem.

Consider the example by Hackbusch and Kühn [62] of a function $f(x_1, x_2, x_3, x_4) = \alpha(x_1, x_2)\beta(x_3, x_4)$. Clearly, this function is separable between dimensions x_2 and x_3. In this case, one does not need to consider the interaction of all basis functions $\mathbf{U}_k$. Instead, one can build a low rank representation of α using only the basis $\mathbf{U}_1$ and $\mathbf{U}_2$, and a low-rank representation of β using basis $\mathbf{U}_3$ and $\mathbf{U}_4$. The HT generalizes this example by introducing *levels* of subspaces, where each level is a recursive splitting of the previous level. One special splitting, a binary splitting, first splits the variables in half, e.g., $f(x_1, x_2, x_3, x_4) = \alpha(x_1, x_2)\beta(x_3, x_4)$. Then at the next level, each of the halves are again split in half. In the example above, since α and β are only functions of two variables the process would stop.

We now provide a more abstract example of this binary splitting, let l denote a subspace level. For $l = 0$, the tensor approximation subspace $\mathscr{X}^0$ is the tensor product of each each dimension of the corresponding level $\mathscr{Y}_i^0$ for $i = 1, \ldots, d$. At level $l = 1$, subspaces are grouped together two at a time. These grouped subspaces $\mathscr{Y}_{i,i+1}^1 = \mathscr{Y}_i^0 \times \mathscr{Y}_{i+1}^0$ form a new a vector space for the combined variables x_i and

x_{i+1} termed $\mathscr{X}^1_{i,i+1}$. The $l = 1$ level approximation subspace for these variables is correspondingly defined by all of the $\mathscr{Y}^1_{i,i+1}$. The splitting then proceeds recursively until a final level L, where all of the variables are grouped together. This process is visualized in the adaptation of a figure from [62] below

$$
\begin{aligned}
\mathscr{X}^0 &= \underbrace{\mathscr{Y}^0_1 \times \mathscr{Y}^0_2}_{\mathscr{X}^1_{1,2}} \times \underbrace{\mathscr{Y}^0_3 \times \mathscr{Y}^0_4}_{\mathscr{X}^1_{3,4}} \times \cdots \times \underbrace{\mathscr{Y}^0_{d-1} \times \mathscr{Y}^0_d}_{\mathscr{X}^1_{d-1,d}} \\
\mathscr{X}^1 &= \underbrace{\mathscr{Y}^1_{1,2} \times \mathscr{Y}^1_{3,4}}_{\mathscr{X}^2_{1,2,3,4}} \times \cdots \times \underbrace{\mathscr{Y}^1_{d-1,d}}_{\mathscr{X}^2_{d-3,d-2,d-1,d}} \\
\mathscr{X}^2 &= \mathscr{Y}^2_{1,2,3,4} \times \cdots \mathscr{Y}^2_{d-3,d-2,d-1,d} \\
&\vdots \\
\mathscr{X}^L &= \mathscr{Y}^L_{1,\ldots,d}
\end{aligned}
$$

Note the hierarchy of tensor spaces formed by these levels $\mathscr{X}^L \subset \mathscr{X}^{L-1} \subset \cdots \subset \mathscr{X}^0 \subset \mathscr{X}$. Intuitively, at level l there are half as many variables as in level $l-1$, because the variables of level $l-1$ were grouped together to form the variables of level l. Therefore, at level l one only has to consider a set of interactions of basis functions from two subspaces, interactions between all of the subspaces are never required. Such a hierarchy allows one to reduce the computational storage cost to one that grows polynomial with rank because interactions between all the variables are never considered.

A special case of the HT format that has good numerical properties is the *tensor-train* (TT) decomposition [95, 98]. The TT is obtained by a particular splitting of variables corresponding to pruning away one dimension at a time. This procedure can be visualized as

$$
\begin{aligned}
\mathscr{X}^0 &= \underbrace{\mathscr{Y}_1^0 \times \mathscr{Y}_2^0}_{\mathscr{X}_{1,2}^1} \times \underbrace{\mathscr{Y}_3^0}_{\mathscr{X}_3^1} \times \cdots \times \underbrace{\mathscr{Y}_{d-1}^0}_{\mathscr{X}_{d-1}^1} \times \underbrace{\mathscr{Y}_d^0}_{\mathscr{X}_d^1} \\
\mathscr{X}^1 &= \underbrace{\mathscr{Y}_{1,2}^1 \times \mathscr{Y}_3^1}_{\mathscr{X}_{1,2,3}^2} \times \cdots \times \times \underbrace{\mathscr{Y}_{d-1}^0}_{\mathscr{X}_{d-1}^2} \times \underbrace{\mathscr{Y}_d^0}_{\mathscr{X}_d^2} \\
&\vdots \\
\mathscr{X}^{L-2} &= \underbrace{\mathscr{Y}_{1,\ldots,d-2}^{L-2} \times \mathscr{Y}_{d-1}^{L-2}}_{\mathscr{X}_{1,\ldots,d-1}^{L-1}} \times \underbrace{\mathscr{Y}_d^{L-2}}_{\mathscr{X}_d^{L-1}} \\
\mathscr{X}^{L-1} &= \underbrace{\mathscr{Y}_{1,\ldots,d-1}^{L-1} \times \mathscr{Y}_d^{L-1}}_{\mathscr{X}_{1,\ldots,d}^{L}} \\
\mathscr{X}^{L} &= \mathscr{Y}_{1,\ldots,d}^{L}
\end{aligned}
$$

The TT decomposition of tensor $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ represents each element as a product of matrices

$$
\begin{aligned}
\boldsymbol{\mathcal{F}}[i_1, i_2, \ldots, i_d] &= \sum_{\alpha_0=1}^{r_0} \sum_{\alpha_1=1}^{r_k} \cdots \sum_{\alpha_d=1}^{r_d} \boldsymbol{\mathcal{F}}_1[\alpha_0, i_1, \alpha_1] \boldsymbol{\mathcal{F}}_2[\alpha_1, i_2, \alpha_2] \ldots \boldsymbol{\mathcal{F}}_2[\alpha_{d-1}, i_d, \alpha_d], \\
&= \boldsymbol{\mathcal{F}}_1[:, i_1, :] \boldsymbol{\mathcal{F}}_2[:, i_2, :] \ldots \boldsymbol{\mathcal{F}}_d[:, i_d, :], \qquad (3.3)
\end{aligned}
$$

where r_k are called TT ranks with $r_0 = r_d = 1$ and $\boldsymbol{\mathcal{F}}_k \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$ are third-order tensors called the TT cores. The storage complexity of this representation is linear with dimension and quadratic with the TT ranks, i.e., storage complexity is $\mathcal{O}(dnr^2)$ where we let $n_k = n$ and $r_k = r$. Because of the simplicity associated with decomposing a tensor into its TT format and of performing multilinear algebra operations with tensors in TT format, we consider this format most heavily in this thesis. In fact, it is this format for which we develop a continuous analogues of discrete algorithms.

The subspace interpretation of the TT allows us to link the TT ranks to the SVD ranks of various unfolding matrices. An unfolding matrix is the discrete equivalent of

the separated forms discussed in Section 2.1. It refers to the reshaping of a tensor into a matrix, i.e., converting a multidimensional array into a two dimensional array. For example the kth reshaping $\mathbf{F}^k$ of $\boldsymbol{\mathcal{F}}$ refers to grouping the first k variables together and the last $(d-k)$ variables together,

$$\mathbf{F}^k[i_1,\ldots,i_k;i_{k+1},\ldots,i_d] = \boldsymbol{\mathcal{F}}[i_1,\ldots,i_d], \quad \mathbf{F}^k \in \mathbb{R}^{\prod_{i=1}^{k} n_i \times \prod_{i=k+1}^{d} n_i}.$$

Oseledets [95] showed that the ranks of the reshapings provide an upper bound for the TT ranks, i.e., $r_k \leq \operatorname{rank} \mathbf{F}^k$. This result underpins the notion that tensor decompositions exploit the *separability* of the function. In other words, they take advantage of the fact that functions can often be written as the product of a small number of functions of a subset of the dimensions.

In the rest of this chapter, we describe a continuous analogue for many techniques that are typically performed with discrete tensor decompositions. Continuous version of all the major discrete algorithms are provided, and they include cross approximation of black box tensors, tensor rounding, multilinear algebra, and alternating least squares.

3.2 Continuous low-rank decompositions

In this section we describe the continuous analogue of the tensor-train decomposition. In Sections 3.2.1 and 3.2.2, we extend the existence results for the discrete tensor-train developed in [95, 98] to the continuous case. These proofs formalize the construction used in [96] to obtain FT representations of particular example functions.[1]

The algorithms that result from the framework, and that will be discussed in later sections, are completely decoupled from—and hence agnostic to—the representation of the univariate functions that underly the separated representation we discuss in this section. This decoupling is a critical distinction between our work and previous

[1]Note that [96] called these representations functional tensor-train (FTT) decompositions. We prefer the name "function-train" because tensor typically refers to a discrete multidimensional array, while function-train better reflects the continuous nature of the decomposition.

work on low-rank functional decompositions [17, 26, 40, 103]. These previous efforts essentially translate the problem of approximating a function in low-rank format into the problem of approximating the *tensor of parameters* describing the underlying univariate functions. This translation is possible if the univariate approximations are linear in the parameters—for example, when the parameters are coefficients of a prescribed basis. The present framework and algorithms are generalizations of these previous methods: because of our "computing with functions" approach, we can seamlessly incorporate adaptive and nonlinear representations of univariate functions within the overall low-rank format. As such, we are better able to adapt to local structure, for instance, by identifying knot locations for piecewise approximation. Some illustrative implementations of the framework, on discontinuous functions and other problems requiring adaptive or heterogeneous univariate approximations, are given in Section 3.6.1.

We refer to a hypercubic input space $\mathscr{X} = [a_1, b_1] \times \ldots \times [a_d, b_d]$. A function-train is defined by a set of d matrix-valued functions $\{\mathcal{F}_1(x_1), \mathcal{F}_2(x_2), \ldots, \mathcal{F}_d(x_d)\}$ and a set of FT-ranks $\boldsymbol{r} = [r_0, r_1, \ldots, r_d]$ such that $r_0 = r_d = 1$ and $\mathcal{F}_i : [a_i, b_i] \to \mathbb{R}^{r_{i-1} \times r_i}$ for $i = 1, \ldots, d$. Furthermore, an evaluation of a function $f : \mathscr{X} \to \mathbb{R}$ in FT format is obtained through a sequence of vector-matrix products

$$f(x) = \mathcal{F}_1(x_1)\mathcal{F}_2(x_2)\ldots\mathcal{F}_d(x_d). \tag{3.4}$$

It will also be helpful to think of these cores $\mathcal{F}_i$ as matrices of univariate functions, e.g.,

$$f(x) = \begin{bmatrix} f_{1,1}^{(1)} & \ldots & f_{1,r_1}^{(1)} \end{bmatrix} \begin{bmatrix} f_{1,1}^{(2)} & \ldots & f_{1,r_2}^{(2)} \\ \vdots & & \vdots \\ f_{r_1,1}^{(2)} & \ldots & f_{r_1,r_2}^{(2)} \end{bmatrix} \ldots \begin{bmatrix} f_{1,1}^{(d)} \\ \vdots \\ f_{r_{d-1},1}^{(d)} \end{bmatrix}, \tag{3.5}$$

where $f_{i,j}^{(k)} : [a_k, b_k] \to \mathbb{R}$. Note the equivalence between the cores in the FT and the cores of the TT (3.3). If the third-order tensors representing the cores of the TT are

discretized into an infinite number of grid points we arrive at matrix-valued functions that represent the FT cores. Analogously to the discrete case, the FT of a function f inherits many of the properties of the separated forms f^k. For example, if the functional SVD of each of these f^k exists, then we can prove the FT exists. Also, the ranks of the truncated SVD upper bound the FT rank r_k, just as in the discrete case.

3.2.1 FT for functions with finite-rank unfoldings

We start with a result bounding the FT ranks of a function whose unfoldings are all finite rank.

Theorem 3. *If each separated form f^k (2.1) of a d-dimensional function f has a finite rank SVD (2.4) with ranks*

$$\operatorname{rank} f^k = r_k, \tag{3.6}$$

then there exists an FT decomposition $\hat{f}$ of form (3.4) with FT-ranks not higher than r_k such that $\|\hat{f} - f\|_{L^2} = 0$.

The proof of this theorem is the continuous analogue of the corresponding theorem by Oseledets [95] and is provided in Appendix A.3. The proof is constructive and it describes an SVD based algorithm for decomposing the function. In particular, beginning with the right singular functions of the first unfolding of f, a sequence of extended SVD are performed for a sequence of right singular functions.

3.2.2 FT for functions with approximately low-rank unfoldings

We continue by showing that if the unfoldings functions f^k of f are *approximately* low-rank (see (2.18)), then we can obtain a FT whose error is bounded.

Theorem 4. *Suppose that the unfoldings f^k of the function f satisfy*

$$f^k = g^k + e^k, \quad \operatorname{rank} g^k = r_k, \quad \left|\left|e^k\right|\right|_{L^2} = \varepsilon_k, \quad \text{for } k = 1, \ldots, d-1 \tag{3.7}$$

Then a rank $\boldsymbol{r} = [r_0, r_1, \ldots, r_d]$ *approximation* $\hat{f}$ *of* f *in the the FT-format may obtained with bounded error*

$$\sqrt{\int \left(f - \hat{f}\right)^2 dx} \leq \sqrt{\sum_{k=1}^{d-1} \varepsilon_k^2}. \tag{3.8}$$

The proof is provided in Appendix A.4. As with Theorem 3, the proof of this theorem is constructive, and it follows the corresponding proof in the discrete TT case [98].

Similar results for the function-train are developed in [17]; in particular, see [17, Prop. 9]. There, the FT approximation error is expressed in terms of the singular values of the extended SVD of each collection of right singular functions. These singular values are further related to the Sobolev regularity of f, thus yielding results on the approximation rate of the FT format (e.g., error bounds that depend on rank, regularity, and dimension). In the present paper, however, we sidestep this functional analytic perspective and obtain error bounds directly from an assumption about the properties of unfoldings of f. (These bounds do not directly involve regularity; for instance, even a discontinuous unfolding might be low rank.) Our approach thus closely follows the discrete TT results in order to provide a general link to a variety of subspace-based [61] tensor decompositions. The properties of an FT decomposition can then be seen directly from the properties of the unfoldings. For example, to find an ordering of the input variables that yields the lowest storage cost, one can search for orderings that yield low-rank unfoldings.

3.3 Continuous cross approximation and rounding

The proofs of the two theorems described in sections above are constructive in the sense that they describe an algorithm that may be used to obtain the FT approximation. In particular, the algorithm requires taking the SVDs of a sequence of vector-valued functions V_i to obtain the next core. However, taking an SVD of high dimensional functions is infeasible, and Oseledets and Tyrtyshnikov [98] propose re-

placing the SVD of each V_i with the skeleton decomposition to obtain the discrete TT. In this section, we extend their cross approximation algorithm to the continuous case for obtaining a FT approximation of a black box function. Then, we describe tensor rounding and show how it can be used for rank adaptation in situations for which the rank is not known.

3.3.1 Cross approximation of multivariate functions

The cross approximation of a multivariate function is a straightforward extension of the cross approximation of the separated representation of a vector-valued function (described in Section 2.3.3). The idea is that whenever the extended SVD of a vector-valued function is required, Algorithm 1 is used to obtain a skeleton decomposition of the function instead.

Furthermore, since we consider sequences of unfoldings corresponding to *different* partitions of the input variables, the skeleton decomposition is defined by more than just a single pair of fiber sets $\boldsymbol{\alpha}$ and $\boldsymbol{y}$ as in Section 2.3. Instead, we have $d-1$ pairs of fiber sets $\{(\boldsymbol{z}^1, \boldsymbol{y}^1), \ldots, (\boldsymbol{z}^{d-1}, \boldsymbol{y}^{d-1})\}$, where each pair $(\boldsymbol{z}^k, \boldsymbol{y}^k)$ corresponds to the fibers used in the skeleton decomposition of the unfolding f^k. For example, if we consider an unfolding $f^2(x_{\leq 2}, x_{>2})$ of rank r_2, then we can use the fiber sets

$$\boldsymbol{z}^2 = \left\{z_1^2, \ldots, z_{r_2}^2\right\}, \quad z_i^2 \in \mathscr{X}_{\leq 2}, \quad i = 1, \ldots, r_2,$$

and

$$\boldsymbol{y}^2 = \left\{y_1^2, \ldots, y_{r_2}^2\right\}, \quad y_i^2 \in \mathscr{X}_{>2}, \quad i = 1, \ldots, r_2,$$

to write the skeleton decomposition of this unfolding as

$$\begin{aligned} f^2(x_{\leq 2}, x_{>2}) &= f(\{x_1, x_2\}, \boldsymbol{y}^2)\mathbf{G}_2 f(\boldsymbol{z}^2, \{x_3, x_4, \ldots x_d\}) \\ &= U_2(x_1, x_2)\mathbf{G}_2 V_2(x_3, x_4, \ldots x_d), \end{aligned}$$

where U_2 is a vector-valued function composed of r_2 scalar-valued functions, each from $\mathcal{X}_1 \times \mathcal{X}_2 \to \mathbb{R}$; $\mathbf{G}_2 = [f(\boldsymbol{z}^2, \boldsymbol{y}^2)]^\dagger$; and V_2 is a vector-valued function composed of

r_2 functions, each from $\mathscr{X}_3 \times \cdots \times \mathscr{X}_d \to \mathbb{R}$. Here we use the notational convention that if f accepts a vector argument, it returns a vector-valued function, e.g.,

$$f\left(x_{\leq 2}, \boldsymbol{y}^2\right) = \left[f\left(x_{\leq 2}, y_1^2\right) \ \cdots \ f\left(x_{\leq 2}, y_{r_2}^2\right)\right] = U_2(x_{\leq 2}).$$

As with the cross approximation algorithm in [98] we enforce a nestedness condition on the sequence $(\boldsymbol{z}^k)_{k=1}^d$: if $z_i^{k+1} = (x_1, \ldots, x_k, x_{k+1}) \in \boldsymbol{z}^{k+1}$ for some $i \in \{1, \ldots, r_{k+1}\}$ then there exists a $j \in \{1, \ldots, r_k\}$ such that $z_j^k = (x_1, \ldots x_k)$. Our algorithm also enforces an analogous nestedness condition for $(\boldsymbol{y}^k)_{k=1}^d$.

We now walk through the cross approximation procedure for a function f whose unfoldings have ranks $(r_k)_{k=1}^{d-1}$. Cross approximation starts with the skeleton decomposition of the first unfolding

$$\begin{aligned} f^1(x_1, x_{>1}) &= f(x_1, \boldsymbol{y}^1)\mathbf{G}_1 f(\boldsymbol{z}^1, x_{>1}) \\ &= \mathcal{F}_1(x_1) V_1(x_{>1}), \end{aligned}$$

where $\mathcal{F}_1(x_1) := f(x_1, \boldsymbol{y}^1)\mathbf{G}_1$ and V_1 is a vector-valued function such that $V_1[i](x_{>1}) = f(z_i^1, x_{>1})$ where z_i^1 is the i-th element of $\boldsymbol{z}^1$. Now, V_1 is a vector-valued function with a rank-r_2 extended SVD, and thus, by Theorem 1 its skeleton decomposition exists and can be written as

$$\begin{aligned} V_1(x_{>1}) &= V_1(x_2, x_{>2}) \\ &= f(\boldsymbol{z}^1, x_2, x_{>2}) \\ &= f(\boldsymbol{z}^1, x_2, \boldsymbol{y}^2)\mathbf{G}_2 f(\boldsymbol{z}^2, x_{>2}), \end{aligned}$$

where the second equality is simply the definition of V_1, and the third equality follows from the definition of the skeleton decomposition and the fact that $\boldsymbol{z}^1$ and $\boldsymbol{y}^2$ are nested in the corresponding sequences of $\boldsymbol{z}^k$ and $\boldsymbol{y}^k$, respectively.

From the last line above, we can write $\mathcal{F}_2(x_2) := f(\boldsymbol{z}^1, x_2, \boldsymbol{y}^2)\mathbf{G}_2$ and $V_2 := f(\boldsymbol{z}^2, x_{>2})$. We now have have the first two cores $\mathcal{F}_1$ and $\mathcal{F}_2$, and can proceed recur-

sively to decompose V_2, and so on. Once we finish this forward sweep, we can perform a similar backwards sweep. Further sweeps may be performed until convergence.

3.3.2 Rounding and rank adaptation

The cross approximation procedure discussed above relies on specifying the ranks r_k *a priori*. In practice, one can find the ranks adaptively through a procedure called *rounding*. The idea is that if an FT with certain ranks can be well approximated by an FT of smaller ranks, then we have overestimated the ranks used in the cross approximation algorithm and can be confident about its results. Rounding begins with a given FT decomposition and aims to generate an approximation of it with relative error ϵ. Rounding is useful not only as a way of verifying the ranks used in cross approximation, but also because the ranks of an FT representation may be higher than necessary depending on how it was created. We now describe a continuous rounding procedure, which is similar to the procedure for discrete TT representations in [95], with the primary distinction being the notions of left and right orthogonality.

The rounding procedure follows directly from the definition of the ranks of the separated form of the function f. Suppose we start with the first unfolding of a function in FT format given by

$$\begin{aligned} f^1(x_1, x_{>1}) &= \mathcal{F}_1(x_1)\left[\mathcal{F}_2(x_2)\mathcal{F}_3(x_3)\ldots\mathcal{F}_d(x_d)\right] \\ &= \mathcal{F}_1\mathcal{V}_1, \end{aligned}$$

where $\mathcal{V}_1 : \mathscr{X}_{>1} \to \mathbb{R}^{r_1\times 1}$. This equation expresses f^1 as a rank r_1 singular value decomposition. Next we seek to compress f^1 through a truncated SVD. The truncated SVD is obtained by first performing two QR decompositions of matrix-valued functions according to Definition 4:

$$\mathcal{F}_1(x_1) = \mathcal{Q}_1(x_1)\mathbf{R}_1 \text{ and } \mathcal{V}_1^T(x_{>1}) = \widetilde{\mathcal{Q}}_1(x_{>1})\widetilde{\mathbf{R}}_1, \tag{3.9}$$

such that

$$f^1(x_1, x_{>1}) = \mathcal{Q}_1(x_1)\mathbf{R}_1\widetilde{\mathbf{R}}_1^T\widetilde{\mathcal{Q}}_1(x_{>1})^T.$$

Then, one obtains the truncated SVD of $\mathbf{R}_1\widetilde{\mathbf{R}}_1^T \simeq \mathbf{U_1}\mathbf{D_1}\mathbf{V_1^T}$, where $\mathbf{U}_1 \in \mathbb{R}^{r_1 \times \hat{r}_1}$, $\mathbf{V}_1^T \in \mathbb{R}^{\hat{r}_1 \times r_1}$, and $\mathbf{D_1}$ is a diagonal matrix whose values are all truncated so that we obtain an approximation of g_1 that has an error δ:

$$g_1 = \underbrace{\mathcal{Q}_1\mathbf{U}_1}_{\hat{\mathcal{F}}_1}\underbrace{\left[\mathbf{D}_1{\mathbf{V}_1}^T\widetilde{\mathcal{Q}}_1^T\right]}_{\hat{\mathcal{F}}_2\hat{\mathcal{F}}_3\ldots\hat{\mathcal{F}}_d},$$

where $\|f - g_1\| \leq \delta$ and $\hat{\mathcal{F}}_1$ now has a reduced size $1 \times \hat{r}_1$ where $\hat{r}_1 \leq r_1$. Now that we have reduced the number of columns of the first core and the number of rows of the second core from r_1 to $\hat{r}_1$, we move on to the second unfolding of the approximate g_1 using the updated cores. Again, a truncated SVD will be performed on this unfolding to reduce r_2 to $\hat{r}_2$. In this case we have

$$\begin{aligned} g_1^2(x_{\leq 2}, x_{>2}) &= \left[\hat{\mathcal{F}}_1(x_1)\hat{\mathcal{F}}_2(x_2)\right]\left[\mathcal{F}_3(x_3)\ldots\mathcal{F}_d(x_d)\right], \\ &= \mathcal{U}_2\mathcal{V}_2, \end{aligned}$$

where $\mathcal{U}_2 : \mathscr{X}_2 \to \mathbb{R}^{1\times r_2}$ and $\mathcal{V}_2 : \mathscr{X}_2 \to \mathbb{R}^{r_2\times 1}$. The QR decomposition of matrix-valued functions is again used to obtain a truncated SVD

$$\begin{aligned} \mathcal{U}_2(x_{\leq 2}) &= \mathcal{Q}_2(x_{\leq 2})\mathbf{R}_2, && \text{Matrix-valued QR of the left cores} \\ \mathcal{V}_2^T(x_{>2}) &= \widetilde{\mathcal{Q}}_2(x_{>2})\widetilde{\mathbf{R}}_2, && \text{Matrix-valued QR of the right cores} \\ \mathbf{R}_2\widetilde{\mathbf{R}}_2^T &\simeq \mathbf{U_2}\mathbf{D_2}\mathbf{V_2^T} && \text{Truncated SVD} \\ g_2 &= \underbrace{\mathcal{Q}_2\mathbf{U_2}}_{\hat{\mathcal{F}}_1\hat{\mathcal{F}}_2}\underbrace{\left[\mathbf{D_2}{\mathbf{V_2}}^T\widetilde{\mathcal{Q}}_2^T\right]}_{\hat{\mathcal{F}}_3\ldots\hat{\mathcal{F}}_d}, && \text{New approximation} \end{aligned}$$

such that $\|g_2 - g_1\| \leq \delta$ The truncated SVD in the third equation is responsible for making the second and third cores smaller such that $\hat{r}_2 \leq r_2$. After the truncation

associated with the first and second cores we obtain a total error

$$\begin{aligned}\|g_2 - f\| &= \|g_2 - g_1 + g_1 - f\|, \\ &\leq \delta + \delta = 2\delta.\end{aligned}$$

We can repeat this procedure $(d-1)$ times to obtain a final approximation $\hat{f} = g_{d-1}$ such that

$$\|\hat{f} - f\| \leq (d-1)\delta.$$

Thus we can obtain a relative error of ϵ by setting $\delta = \frac{\epsilon}{d-1}\|f\|$.

The computational difficulty with the algorithm described above is obtaining the QR decomposition of the matrix-valued functions $\mathcal{U}_k(x_{\leq k})$ and $\mathcal{V}_k(x_{>k})$, since they have multivariate, potentially high dimensional inputs. A computationally feasible algorithm must be limited to one in which the QR decompositions are feasible. In the discrete setting, we can obtain an algorithm that only requires the QR decompositions of reshapings of single cores. In the continuous setting, we analogously obtain an algorithm that requires only the QR decompositions of each FT core. This algorithm starts by assuming that all the cores $\mathcal{F}_2, \ldots, \mathcal{F}_d$ have *orthonormal rows*. Then, we can show that $\mathcal{V}_1$ also has orthonormal rows and that we are not required to take its QR decomposition in (3.9). Thus in the first step of the rounding procedure we only need to compute the QR of $\mathcal{F}_1$. The concept of *orthonormal rows* is analogous to left orthogonality [95] and, relatedly, *orthonormal columns* is analogous to right orthogonality. The two definitions are explicitly given below where the inner products refer to the inner products between quasimatrices as given in (2.2).

Definition 17 (Orthonormal rows). *A function train core $\mathcal{F}_k : \mathcal{X}_k \to \mathbb{R}^{r_{k-1} \times r_k}$ has orthonormal rows iff*

$$\langle \mathcal{F}_k[i,:](x), \mathcal{F}_k[j,:](x) \rangle = \delta_{i,j} \quad \text{for } i, j = 1 \ldots r_{k-1} \tag{3.10}$$

Definition 18 (Orthonormal columns). *A function train core $\mathcal{F}_k : \mathcal{X}_k \to \mathbb{R}^{r_{k-1} \times r_k}$*

has orthonormal columns iff

$$\langle \mathcal{F}_k[:,i](x), \mathcal{F}_k[:,j](x) \rangle = \delta_{i,j} \quad \textit{for } i,j = 1 \dots r_k \tag{3.11}$$

Now to show that $\mathcal{V}_1$ has orthonormal rows we consider the inner product between the rows of $\mathcal{V}_1$. Since evaluations of $\mathcal{V}_1$ take values in $\mathbb{R}^{r_1 \times 1}$ we will denote by v_k

$$v_k(x_2, \dots, x_d) = \mathcal{F}_2[k,:](x_2)\mathcal{F}_3(x_3) \dots \mathcal{F}_d(x_d),$$

the scalar-valued function representing the k-th row of $\mathcal{V}_1$. The inner product between row k and l can be obtained as

$$\begin{aligned}
\langle v_k, v_l \rangle &= \int v_k(x_2, \dots, x_d) v_l(x_2, \dots, x_d) dx_2 \dots dx_d \\
&= \int \left(\mathcal{F}_2[k,:](x_2)\mathcal{F}_3(x_2) \dots \mathcal{F}_d(x_d)\right) \left(\mathcal{F}_2[l,:](x_2)\mathcal{F}_3(x_2) \dots \mathcal{F}_d(x_d)\right) dx_2 \dots dx_d \\
&= \int \left(\mathcal{F}_2[k,:](x_2)\mathcal{F}_3(x_2) \dots \mathcal{F}_d(x_d)\right) \left(\mathcal{F}_d(x_d)^T \dots \mathcal{F}_3(x_2)^T \mathcal{F}_2[l,:](x_2)^T\right) dx_2 \dots dx_d \\
&= \int \mathcal{F}_2[k,:](x_2) \Big(\int \mathcal{F}_3(x_3) \Big(\dots \Big(\int \mathcal{F}_d(x_d) \mathcal{F}_d(x_d)^T dx_d \Big) \dots \Big) \mathcal{F}_3(x_3)^T dx_3 \Big) \mathcal{F}_2[l,:](x_2)^T dx_2
\end{aligned} \tag{3.12}$$

Row orthonormality implies that

$$\int \mathcal{F}_k(x_k) \mathcal{F}_k^T(x_k) dx_k = \mathbf{I}_{r_{k-1}}, \quad \text{for all } k = 2, \dots, d,$$

and therefore

$$\langle v_k, v_l \rangle = \int \mathcal{F}_2[k,:](x_2) \mathbf{I}_{r_2} \mathcal{F}_2[l,:](x_2)^T dx_2 = \delta_{k,l}.$$

Since $\mathcal{V}_1$ has orthonormal rows, we have $\widetilde{\mathbf{R}}_1 = \mathbf{I}_{r_1}$, and there is no need to take the QR decompositions of $\mathcal{V}_1$ in the first step. Furthermore, one can apply this reasoning to each $\mathcal{V}_k$ to show that the QR decomposition is never required as long as the cores from which it is formed has orthonormal rows.

We now move onto dealing with taking the QR decomposition of $\mathcal{U}_2$. Recall that $\mathcal{U}_2 = \hat{\mathcal{F}}_1 \hat{\mathcal{F}}_2$ where $\hat{\mathcal{F}}_1$ has orthonormal columns obtained from the SVD at the

previous step. Since one can use the same reasoning as above to show that the product of matrix-valued functions with orthonormal columns has orthonormal columns, we can orthogonalize the columns of $\mathcal{U}_2$ by only orthogonalizing the columns of $\hat{\mathcal{F}}_2$. This procedure only requires the QR decomposition of a univariate matrix-valued function (e.g., $\hat{\mathcal{F}}_2$), a computationally feasible procedure. Therefore, when considering the QR decompositions of the successive $\mathcal{U}_k$, only the QR decomposition of $\hat{\mathcal{F}}_k$ obtained from the previous iteration is required.

The entire rounding algorithm is given in Algorithm 3. It starts with a sweep through cores $k = 2 \ldots d$ from the right to the left to orthogonalize all of their rows. Once this is completed, a left-to-right sweep is performed in which the reduced SVD is performed. Core orthogonalization is performed by taking a sequence of QR decompositions denoted by `qr-mvf`. The reduced SVD algorithm performed by first taking the QR decomposition followed by a reduced SVD of $\mathbf{R}_k$, and it is denoted by `svd-core`.

Algorithm 3 `ft-round`: Function-train rounding

Require: An FT f; accuracy parameter ϵ
Ensure: $\hat{f}$ with reduced ranks such that $\|f - \hat{f}\| \leq \epsilon \|f\|$
1: $\delta = \frac{\epsilon}{d-1}\|f\|$
2: $\hat{\mathcal{F}}_d = \mathcal{F}_d$
3: **for** $k = d$ **to** 2 **do**
4: $\quad \hat{\mathcal{F}}_k(x_k)\mathbf{R} = \texttt{qr-mvf}(\hat{\mathcal{F}}_k{}^T)$ # Orthonormalize all elements of core k
5: $\quad \hat{\mathcal{F}}_{k-1} = \mathcal{F}_{k-1}\mathbf{R}^T$
6: **end for**
7: **for** $k = 1$ **to** $d - 1$ **do**
8: $\quad \hat{\mathcal{F}}_k\mathbf{\Lambda}\mathbf{V} = \texttt{svd-core}(\hat{\mathcal{F}}_k, \delta)$ # Truncated SVD with tolerance δ
9: $\quad \hat{\mathcal{F}}_{k+1} = \mathbf{\Lambda}\mathbf{V}\hat{\mathcal{F}}_k$
10: **end for**

The overall approximation algorithm with rank adaptation can then be implemented by successively increasing, or "kicking," the FT ranks higher until rounding leads to a reduction of all ranks. This type of algorithm ensures that the FT ranks are *overestimated* for the cross approximation procedure. The pseudocode for the rank-adaptation procedure is provided by Algorithm 4.

Algorithm 4 `ft-rankadapt`: Function-train approximation with rank adaptation

Require: A d-dimensional black box function $f : \mathscr{X} \to \mathbb{R}$; cross approximation tolerance δ_{cross}; Number of ranks to increase with each adaptation `kickrank`; Rounding accuracy ϵ_{round}; Initial ranks $\boldsymbol{r}$

Ensure: Approximation $\hat{f}$ such that a rank increase and rounding does not change ranks.

$\hat{f} = \texttt{cross-approx}(f, \boldsymbol{r}, \delta_{\text{cross}})$
$\hat{f}_r = \texttt{ft-round}(\hat{f}, \epsilon_{\text{round}})$
$\hat{\boldsymbol{r}} = \text{rank}(\hat{f}_r)$
while $\exists i$ s.t. $\hat{r}_i = r_i$ **do**
  for $k = 1 \to d-1$ **do**
    $\boldsymbol{r}_k = \hat{\boldsymbol{r}}_k + \texttt{kickrank}$
    $\hat{f} = \texttt{cross-approx}(f, \boldsymbol{r}, \delta_{\text{cross}})$
  end for
  $\hat{f}_r = \texttt{ft-round}(\hat{f}, \epsilon_{\text{round}})$
  $\hat{\boldsymbol{r}} = \text{rank}(\hat{f}_r)$
end while
$\hat{f} = \hat{f}_r$

3.4 Continuous multilinear algebra

Performing continuous multilinear algebra is one of the main advantages of the continuous framework discussed in this dissertation. The operations of addition, multiplication, differentiation, integration, and inner products are easily performed for functions in FT format.

Addition and multiplication of two functions are performed similarly to addition and multiplication of tensors in TT format. For *addition*, the cores of $g(x) = f(x) + h(x)$ are

$$\mathcal{G}_1(x) = [\mathcal{F}_1(x) \quad \mathcal{H}_1(x)], \quad \mathcal{G}_k(x) = \begin{bmatrix} \mathcal{F}_k(x) & \mathbf{0} \\ \mathbf{0} & \mathcal{H}_k(x) \end{bmatrix}, \quad \mathcal{G}_d(x) = \begin{bmatrix} \mathcal{F}_d(x) \\ \mathcal{H}_d(x) \end{bmatrix},$$

for $k = 2 \ldots d$. For *multiplication*, $g(x) = f(x)h(x)$, we have

$$\mathcal{G}_k(x) = \mathcal{F}_k(x) \otimes \mathcal{H}_k(x) \quad \text{for } k = 1 \ldots d.$$

For both of these operations, the continuous functional decomposition has an impor-

tant advantage compared with operations based on discretized representation of f and h. Primarily, this advantage comes from the ability to add functions of differing discretization levels, e.g., functions represented with bases of different orders. In the discrete case, one can only add functions with identical discretizations.

The continuous nature of the FT also allows us to perform *differentiation*. To perform this operation in the discrete case, one would be required to choose some finite difference rule. From the purely discrete perspective of multidimensional arrays, the concept of differentiation is not well defined or unique, e.g., one can choose different finite difference rules. In the continuous case, performing differentiation requires differentiating scalar-valued functions that make up the corresponding core. For example, consider the partial derivative of a d dimensional function f

$$\frac{\partial f}{\partial x_k} = \mathcal{F}_1 \dots \mathcal{F}_{k-1} \begin{bmatrix} \frac{df_{1,1}^{(k)}}{dx_k} & \cdots & \frac{df_{1,r_k}^{(k)}}{dx_k} \\ \vdots & & \vdots \\ \frac{df_{r_{k-1},1}^{(k)}}{dx_k} & \cdots & \frac{df_{r_{k-1},r_k}^{(k)}}{dx_k} \end{bmatrix} \mathcal{F}_{k+1} \dots \mathcal{F}_d.$$

If each of the corresponding univariate functions is expressed in, for example, a basis of orthonormal polynomials, then this operation is unique, well defined, and computationally inexpensive to compute.

Integration is another area of difference between the discrete and continuous cases. In the discrete case, integration involves a sequence of tensor-vector contractions, where the vectors denote quadrature weights. Again, this requires both a non-unique interpretation of the values of the tensor as quadrature nodes and an explicit decision as to what integration rule to use. In the continuous case, integration involves integrating over all the one dimensional functions in each core and then performing

matrix-vector multiplication $d-1$ times

$$\begin{aligned}\int f(x)dx &= \int \mathcal{F}_1(x_1)\mathcal{F}_2(x_2)\dots\mathcal{F}_d(x_d)dx_1\dots dx_d \\ &= \left(\int \mathcal{F}_1(x_1)dx_1\right)\left(\int \mathcal{F}_2(x_2)dx_2\right)\dots\left(\int \mathcal{F}_d(x_d)dx_d\right) \\ &= \mathbf{\Gamma_1\Gamma_2}\dots\mathbf{\Gamma_d} \end{aligned} \tag{3.13}$$

where $\mathbf{\Gamma_k} = \int \mathcal{F}_k(x_k)dx_k$ contains entries $\mathbf{\Gamma_k}[i,j] = \int f_{i,j}^{(k)}(x_k)dx_k$. Furthermore, since each of the univariate functions is typically represented in a known basis, this integral is well defined, unique, and computationally inexpensive to obtain.

The *inner product* between two functions is another important operation that is ubiquitous. Naïvely, the inner product can be implemented by first computing the product $g(x) = f(x)h(x)$ and then integrating $g(x)$, requiring $\mathcal{O}(dr^4)$ operations. However, this operation can be made more efficient by combining the operations needed for integration and multiplication. For example, Algorithm 5 uses an efficient computation of $v^T(\mathbf{A}\otimes\mathbf{B})$ to perform the inner product in $\mathcal{O}(dr^3)$.

Algorithm 5 `ft-inner`: Inner product between two functions in FT format

Require: Functions f with ranks $r_k^{(f)}$ and g with ranks $r_k^{(g)}$ in FT format
Ensure: $y = \int f(x)g(x)dx$
1: $\boldsymbol{y} = \int \mathcal{G}_1(x_1)\otimes\mathcal{F}_1(x_1)dx_1$
2: **for** $k = 2$ **to** d **do**
3: $\quad\mathbf{Y} = \texttt{reshape}(\boldsymbol{y}, r_{k-1}^{(f)}, r_{k-1}^{(g)})$
4: $\quad\mathcal{T} = \mathcal{F}_k(x)^T\mathbf{Y}$
5: $\quad\mathcal{A} = \mathcal{G}_k(x)^T\mathcal{T}^T(x)$
6: $\quad\mathbf{Y} = \int \mathcal{A}(x_k)dx_k$
7: $\quad\boldsymbol{y} = \texttt{reshape}(\mathbf{Y}, 1, r_k^{(f)}r_k^{(g)})$
8: **end for**
9: $y = \boldsymbol{y}[1]$

Furthermore, once in FT format, many other familiar operators may be applied to a function with relative ease. Consider the Laplacian $\Delta f(x) = g(x) = \sum_{k=1}^{d}\frac{\partial^2 f(x)}{\partial x_k^2}$. Written in this form, one can consider the Laplacian as the summation of d functions

$g_k(x)$ in function-train format where

$$g_k(x) = \frac{\partial^2 f(x)}{\partial x_k^2}.$$

One can imagine using the ability to apply this operator directly on the functional format to construct iterative linear system solvers. In the context of low-rank tensor decompositions, these types of solvers are the subject of much current research [37, 74, 116]

3.5 Continuous rank-revealing alternating least squares

Alternating least squares (ALS) is a ubiquitous optimization algorithm for tensor-based computation [38, 40, 75, 94, 97, 123]. The main idea of ALS is to sequentially solve a multivariate optimization problem by solving a sequence of least squares subproblems that result from fixing all but one or two variables. In the context of low-rank tensor decompositions, the separated form of the decompositions means that the notion of dealing with one or two variables at a time is conceptually attractive.

Many operations, for example multiplication of low-rank functions, can potentially be performed more efficiently by solving a corresponding optimization problem using ALS. Recall from Section 3.4 that multiplication of two functions results in a function with squared ranks, and in practice, rounding is then used to decrease the rank once more. Alternatively, posing such operations as solutions of corresponding optimization problems can potentially bypass intermediate results that have large rank. In other words, when the rounded result of a multilinear algebraic operation may indeed be low-rank, solving a corresponding optimization problem with ALS can yield computational advantages. We discuss specific optimization problems corresponding to multiplying functions and applying high-dimensional diffusion operators in Sections 3.5.2 and 3.5.3, respectively.

In this section, we first show how to use continuous linear algebra within ALS to obtain an adaptive and rank-revealing solution of an optimization problem. Loosely

speaking, we want to find the "best" approximation $f : \mathscr{X} \to \mathbb{R}$ in FT format to another function $g : \mathscr{X} \to \mathbb{R}$. Recall (3.4) where f in FT format is written as

$$f(x) = \mathcal{F}_1(x_1) \dots \mathcal{F}_d(x_d),$$

the FT cores are denoted $\mathcal{F}_k(x_k) : \mathscr{X}_k \to \mathbb{R}^{r_{k-1} \times r_k}$, and the core sizes $r_0, \dots r_d$ are the TT/FT ranks.

Mathematically, this optimization problem can be formulated in two related ways. The first way is

$$\underset{f}{\text{minimize}} \sum_{k=1}^{d} r_{k-1} r_k \quad \text{subject to} \quad \|f - g\|^2 \leq \epsilon, \tag{3.14}$$

where the norm is the L^2 norm. This formulation seeks to find the function-train approximation to $g(x)$ that has the smallest FT cores. In spirit, this formulation is related to

$$\underset{f}{\text{minimize}} \ \|f - g\|^2 \quad \text{subject to} \quad r_k \leq r, \quad k = 1, \dots, d. \tag{3.15}$$

The difference between these two formulations is that in (3.14) we specify a maximum error bound ϵ, and seek to find an approximation f with the smallest ranks that satisfies this error. Formulation (3.15) reverses the constraint and seeks an approximation within a set of functions whose FT ranks are bounded by r. Formulation (3.14) can be more attractive if a tight error tolerance is desired, while (3.15) can be more attractive if the memory or computational time is constrained.

Algorithms that solve these optimization problems require indirect access to the cores of the FT decomposition of g, i.e.,

$$g(x) = \mathcal{G}_1(x_1) \dots \mathcal{G}_d(x_d),$$

that has ranks $\hat{r}_0, \dots, \hat{r}_d$. Typically, the ranks of g will be much larger than the ranks of a good approximation f, i.e., $\hat{r}_k \gg r_k$, and the purposes of the optimization is to obtain a good approximation that has smaller ranks. Furthermore, the need

for performing ALS typically arises because each core $\mathcal{G}_k$ is itself too expensive to form; instead, it has some structure that can be exploited by the ALS algorithm. For example, if g is the result of multiplying two functions $g(x) = h(x)z(x)$ with cores $\mathcal{H}_k$ and $\mathcal{Z}_k$, respectively, then in Section 3.4 it is shown that each core of $\mathcal{G}_k$ is given as

$$\mathcal{G}_k(x_k) = \mathcal{H}_k(x_k) \otimes \mathcal{Z}_k(x_k).$$

The algorithm described in this section never explicitly forms $\mathcal{G}_k$ because this operation is too expensive; it requires $\mathcal{O}(r^4)$ operations and a corresponding increase in storage. Together with rounding, such an algorithm requires $\mathcal{O}(r^6)$ operations with respect to the rank of h and g. Instead, our algorithm uses these cores indirectly through multiplication by matrices on the left and right. In other words, we only compute $\mathcal{G}_k(x_k)\mathbf{\Psi}_i$ or $\mathbf{\Phi}_i\mathcal{G}_k(x_k)$, and this computation can be performed in $\mathcal{O}(r^3)$ operations according to Proposition 12. Furthermore, we never compute a SVD of the large matrix-valued function $\mathcal{G}_k$; instead, we perform QR decompositions of the smaller matrices $\mathcal{G}_k(x_k)\mathbf{\Psi}_i$ and $\mathbf{\Phi_i}\mathcal{G}_k$ to achieve a total complexity of $\mathcal{O}(r^5)$ operations, an order of magnitude better than the rounding based approach.

The rank-revealing algorithm developed in this section utilizes the framework of the ALS-DMRG algorithms used in the tensor-train literature for performing multilinear algebra [94] and solving linear systems [97]. The ALS-DMRG methods slightly modifies the classical ALS algorithm by first fixing all but *two* FT cores and then splitting the cores using a truncated SVD to obtain a rank estimate. Next, we develop continuous analogues of the ALS-DMRG algorithm.

3.5.1 Optimization algorithm

The continuous rank-revealing ALS algorithm requires solving a sequence of least squares problems obtained by fixing all but two neighboring cores. Once the two neighboring cores are optimized, the left-right or right-left sweep proceeds by optimizing for the next pair of cores. This pseudocode for this algorithm is provided in Algorithm 6. There are two main tasks in this algorithm, and these tasks are shown

in Lines 7 and 8 in the context of the left-right sweep. Note that they also reappear for the right-left sweep.

The first task is solving an optimization problem (3.16) over a two-dimensional core $\mathcal{W}_i : \mathscr{X}_i \times \mathscr{X}_{i+1} \to \mathbb{R}^{r_i \times r_{i+2}}$ that represents the product of two cores $\mathcal{W}_i = \mathcal{F}_i \mathcal{F}_{i+1}$:

$$\underset{\mathcal{W}_i}{\text{minimize}} \ \|\mathcal{F}_1 \dots \mathcal{F}_{i-1} \mathcal{W}_i \mathcal{F}_{i+2} \dots \mathcal{F}_d - g\|^2. \tag{3.16}$$

The solution to this problem is described in Section 3.5.1. Once $\mathcal{W}_i$ is obtained, it needs to be split into new components $\mathcal{F}_i$ and $\mathcal{F}_{i+1}$.

The second task, splitting $\mathcal{W}$ is referred to as `core-decimation` in Line 8 of Algorithm 6, and its solution is described in Section 3.5.1. Briefly, core decimation uses a truncated SVD to separate the bivariate core W_i into two univariate cores $\mathcal{F}_i$ and $\mathcal{F}_{i+1}$. The ranks of f are learned during this step.

Solution of ALS subproblem

An efficient solution to this optimization relies on row orthonormality (Definition 17) and column orthonormality (Definition 18) of the FT cores $\mathcal{F}_k$. Additionally, we use the following properties, derived in Section 3.3.2, of products of orthonormal matrix-valued functions.

Proposition 13. *Let $f : \mathscr{X}_1 \times \cdots \times \mathscr{X}_d \to \mathbb{R}$ have the FT decomposition $f = \mathcal{F}_1(x_1) \dots \mathcal{F}_d(x_d)$. Suppose that the matrix-valued functions $\mathcal{F}_i : \mathscr{X}_i \to \mathbb{R}^{r_{i-1} \times r_i}$ for $i = \ell, \dots, d$ have orthonormal rows according to Definition 17. Then $\mathcal{R} : \mathscr{X}_\ell \times \cdots \times \mathscr{X}_d \to \mathbb{R}^{r_{\ell-1} \times r_d}$ defined by*

$$\mathcal{R}(x_\ell, x_{\ell+1}, \dots, x_d) = \mathcal{F}_\ell(x_\ell) \mathcal{F}_{\ell+1}(x_{\ell+1}) \dots \mathcal{F}_d(x_d)$$

also has orthonormal rows.

A similar proposition holds for matrix-valued functions with orthonormal columns.

Proposition 14. *Let $f : \mathscr{X}_1 \times \cdots \times \mathscr{X}_d \to \mathbb{R}$ have the FT decomposition $f = \mathcal{F}_1(x_1) \dots \mathcal{F}_d(x_d)$. Suppose that the matrix-valued functions $\mathcal{F}_i : \mathscr{X}_i \to \mathbb{R}^{r_{i-1} \times r_i}$ for*

Algorithm 6 Alternating least squares with DMRG for RHS in FT format

Require: FT cores of g, $\mathcal{G}_k$ for $k = 1, \ldots, d$;
Initial cores approximate $\mathcal{F}_k^{(0)}$ for $k = 1, \ldots, d$;
SVD truncation tolerance ϵ;
Convergence tolerance δ;

Ensure: $\|\mathcal{F}_1^{(\ell)} \ldots \mathcal{F}_d^{(\ell)} - \mathcal{F}_1^{(\ell-1)} \ldots \mathcal{F}_d^{(\ell-1)}\| \leq 1$ where ℓ is the iteration number

$\ell = 1$
while not converged **do**
  {Left-Right Sweep}
  $\mathcal{W}_1 = \arg\min_{\mathcal{W}} \|\mathcal{W}_1 \mathcal{F}_3^{(\ell-1)} \ldots \mathcal{F}_d^{(\ell-1)} - \mathcal{G}_1 \ldots \mathcal{G}_d\|^2$
  $[\mathcal{F}_1^{\ell}, \mathcal{F}_2^{\ell}] =$ `core-decimate`$(\mathcal{W}_1, \epsilon)$ # SVD truncation tolerance ϵ
  for $i = 2 \ldots d-1$ **do**
    $\mathcal{W}_i = \arg\min_{\mathcal{W}} \|\mathcal{F}_1^{(\ell)} \ldots \mathcal{F}_{i-1}^{(\ell)} \mathcal{W} \mathcal{F}_{i+2}^{(\ell-1)} \ldots \mathcal{F}_d^{(\ell-1)} - \mathcal{G}_1 \ldots \mathcal{G}_d\|^2$
    $[\mathcal{F}_i^{\ell}, \mathcal{F}_{i+1}^{\ell}] =$ `core-decimate`$(\mathcal{W}_i, \epsilon)$
  end for
  $f^{(\ell)} = \mathcal{F}_1^{(\ell)} \ldots \mathcal{F}_d^{(\ell)}$
  if $\hat{\delta} \leq \delta$ **then**
    return
  end if
  $\ell = \ell + 1$
  {Right-Left Sweep}
  $\mathcal{W}_{d-1} = \arg\min_{\mathcal{W}} \|\mathcal{F}_1^{(\ell-1)} \ldots \mathcal{F}_{d-2}^{(\ell-1)} \mathcal{W} - \mathcal{G}_1 \ldots \mathcal{G}_d\|^2$
  $[\mathcal{F}_{d-1}^{(\ell)}, \mathcal{F}_d^{(\ell)}] =$ `core-decimate`$(\mathcal{W}_{d-1}, \epsilon)$
  for $i = d-2 \ldots 1$ **do**
    $\mathcal{W}_i = \arg\min_{\mathcal{W}} \|\mathcal{F}_1^{(\ell-1)} \ldots \mathcal{F}_{i-1}^{(\ell-1)} \mathcal{W} \mathcal{F}_{i+2}^{(\ell)} \ldots \mathcal{F}_d^{(\ell)} - \mathcal{G}_1 \ldots \mathcal{G}_d\|^2$
    $[\mathcal{F}_i^{(\ell)}, \mathcal{F}_{i+1}^{(\ell)}] =$ `core-decimate`$(\mathcal{W}_i)$
  end for
  $f^{(\ell)} = \mathcal{F}_1^{(1)} \ldots \mathcal{F}_d^{(\ell)}$
  $\hat{\delta} = ||f^{(\ell)} - f^{(\ell-1)}||$
  if $\hat{\delta} \leq \delta$ **then**
    return
  end if
  $k = k + 1$
end while

$i = 1, \ldots, \ell$ *have orthonormal columns according to Definition 18. Then* $\mathcal{L} : \mathscr{X}_1 \times \cdots \times \mathscr{X}_\ell \to \mathbb{R}^{r_0 \times r_\ell}$ *defined by*

$$\mathcal{L}(x_1, x_2, \ldots, x_\ell) = \mathcal{F}_1(x_\ell)\mathcal{F}_2(x_\ell) \ldots \mathcal{F}_\ell(x_\ell)$$

also has orthonormal columns.

These notions provide the solution for (3.16) that is shown Theorem 5.

Theorem 5. *Let $\mathscr{Y} \subset \mathbb{R}^d$ consist of the tensor product of three subspaces $\mathscr{Y} = \mathscr{Y}_1 \times \mathscr{Y}_2 \times \mathscr{Y}_3$. Furthermore, let $\mathcal{L} : \mathscr{Y}_1 \to \mathbb{R}^{1 \times n_1}$ have orthonormal columns, let $\mathcal{W} : \mathscr{Y}_2 \to \mathbb{R}^{n_1 \times m_1}$ and let $\mathcal{R} : \mathscr{Y}_3 \to \mathbb{R}^{m_1 \times 1}$ have orthonormal rows. Let $\mathcal{G}_1 : \mathscr{Y}_1 \to \mathbb{R}^{1 \times n_2}$, $\mathcal{G}_2 : \mathscr{Y}_2 \to \mathbb{R}^{n_2 \times m_2}$, and $\mathcal{G}_3 : \mathscr{Y}_3 \to \mathbb{R}^{m_2 \times 1}$. Finally, let the cost function be defined as*

$$J(\mathcal{W} = \int_{\mathscr{Y}_1 \times \mathscr{Y}_2 \times \mathscr{Y}_3} \left(\mathcal{L}(y_1)\mathcal{W}(y_2)\mathcal{R}(y_3) - \mathcal{G}_1(y_1)\mathcal{G}_2(y_2)\mathcal{G}_3(y_3)\right)^2 dy_1 dy_2 dy_3.$$

Then the local extremum of the minimization problem

$$\mathcal{W}(y_2)^* = \arg\min_{\mathcal{W}} J(\mathcal{W}(y_2))$$

is

$$\mathcal{W}^*(y_2) = \left[\int_{\mathscr{Y}_1} \mathcal{L}^T(y_1)\mathcal{G}_1(y_1)dy_1\right] \mathcal{G}_2(y_2) \left[\int_{\mathscr{Y}_3} \mathcal{G}_3(y_3)\mathcal{R}^T(y_3)dy_3\right]. \tag{3.17}$$

The proof is provided in Appendix A.5. Theorem 5 can be used for solving (3.16) by associating $\mathscr{X}_{\leq i-1}$ with $\mathscr{Y}_1$, $\mathscr{X}_i \times \mathscr{X}_{i+1}$ with $\mathscr{Y}_2$ and $\mathscr{X}_{>i+1}$ with $\mathscr{Y}_3$. Furthermore, if we equate $\mathcal{F}_{<i} = \mathcal{F}_1 \ldots \mathcal{F}_{i-1}$ with $\mathcal{L}$ and $\mathcal{F}_{>i+2} = \mathcal{F}_{i+2} \ldots \mathcal{F}_d$ with $\mathcal{R}$ we achieve the following solution:

$$\mathcal{W}^*(x_i, x_{i+1}) = \left[\int_{\mathscr{X}_{<i}} \mathcal{F}_{<i}^T \mathcal{G}_{<i}(x_{<i})dx_{<i}\right] \mathcal{G}_i(x_i)\mathcal{G}_{i+1}(x_{i+1}) \left[\int_{\mathscr{X}_{>i+1}} \mathcal{G}_{>i+1}(x_{>i+1})\mathcal{F}_{>i+1}^T(x_{>i+1})dx_{>i+1}\right], \tag{3.18}$$

for $i = 2, \ldots, d-2$, where $\mathcal{G}_{<i} = \mathcal{G}_1 \ldots \mathcal{G}_{i-1}$ and $\mathcal{G}_{>i+1} = \mathcal{G}_{i+2} \ldots \mathcal{G}_d$. A straightforward extension to Theorem 5 can be made for the cases where all but the first two cores are fixed (corresponding to searching for $W(x_1, x_2)$) and when all but the last two cores are fixed (corresponding to searching for $W(x_{d-1}, x_d)$. The solution for the first

core is

$$\mathcal{W}^*(x_1, x_2) = \mathcal{G}_1(x_1)\mathcal{G}_2(x_2)\left[\int_{\mathscr{X}_{>2}} \mathcal{G}_{>2}(x_{>2})\mathcal{F}_{>2}^T(x_{>2})dx_{>2}\right], \tag{3.19}$$

and the solution for the last core is

$$\mathcal{W}^*(x_{d-1}, x_d) = \left[\int_{\mathscr{X}_{<d-1}} \mathcal{F}_{<d-1}^T\mathcal{G}_{<d-1}(x_{<d-1})dx_{<d-1}\right]\mathcal{G}_{d-1}(x_{d-1})\mathcal{G}_d(x_d). \tag{3.20}$$

Core decimation

After the solution of each least squares problem, the bivariate core $\mathcal{W}^*$ must be separated into two univariate cores. This operation is called *decimation* and is the step where rank adaptation occurs by using a truncated SVD. Consider (3.18) re-written as

$$\mathcal{W}^*(x_i, x_{i+1}) = \mathbf{\Phi}_i\mathcal{G}_i(x_i)\mathcal{G}_{i+1}(x_{i+1})\mathbf{\Psi}_i, \tag{3.21}$$

where the matrix $\mathbf{\Phi}_i \in \mathbb{R}^{r_i \times \hat{r}_i}$ corresponds to $\mathbf{\Phi}_i = \int_{\mathscr{X}_{<i}} \mathcal{F}_{<i}^T(x_{<i})\mathcal{G}_{<i}(x_{<i})dx_{<i}$ and the matrix $\mathbf{\Psi}_i \in \mathbb{R}^{\hat{r}_{i+2} \times r_{i+1}}$ corresponds to $\mathbf{\Psi}_i = \int_{\mathscr{X}_{>i+1}} \mathcal{G}_{>i+1}(x_{i+1})\mathcal{F}_{>i+1}^T(x_{>i+1})dx_{>i+1}$. Note the sizes of these matrices, $\mathbf{\Phi}_i \in \mathbb{R}^{r_{i-1} \times \hat{r}_{i-1}}$ is a "short and fat matrix" and $\mathbf{\Psi}_i \in \mathbb{R}^{\hat{r}_{i+1} \times r_{i+1}}$ is "tall and skinny", correspond to the fact that the number of rows and columns of $\mathcal{G}_k$ is greater than the number of rows and columns of $\mathcal{F}_k$.

We decompose the bivariate core with a truncated SVD of a matrix-valued function, i.e.,

$$\mathcal{W}^*(x_i, x_{i+1}) = \mathcal{U}(x_i)\mathbf{S}\mathcal{V}(x_{i+1}),$$

where the matrix-valued function $\mathcal{U}$ has orthonormal columns according to Definition 18 and the matrix-valued function $\mathcal{V}$ has orthonormal rows according to Definition 17. When this optimization problem is a part of a left-right sweep, $\mathcal{U}$ becomes the updated core $\mathcal{F}_i^{(\ell)}$ in Line 8 of Algorithm 6; and when the optimization problem is a part of the right-left sweep, the new core $\mathcal{V}$ becomes the updated core $\mathcal{F}_{i+1}^{(\ell)}$. The diagonal matrix $\mathcal{S}$ can be absorbed into either the left or right eigenfunctions.

This SVD can be obtained in four steps that are similar to the process used for rounding in Section 3.3.2:

1. QR of a matrix-valued function: $\mathcal{Q}(x_i)\mathbf{R} = \mathbf{\Phi}_i \mathcal{G}_i(x)$
2. QR decomposition: $\widetilde{\mathcal{Q}}(x_{i+1})\widetilde{\mathbf{R}} = \left(\mathcal{G}_{i+1}(x_{i+1})\mathbf{\Psi}_i\right)^T$
3. (Truncated) SVD decomposition: $\mathbf{USV} = \mathbf{R}\widetilde{\mathbf{R}}^T$
4. Compute $\mathcal{U}(x_i) = \mathcal{Q}(x_i)\mathbf{U}$ or $\mathcal{V}(x_{i+1}) = \mathbf{V}\widetilde{\mathcal{Q}}^T(x_{i+1})$ depending on sweep direction

While these steps seem similar to those taken for rounding, the primary factor that contributes to higher efficiency is that in step 1, we are taking the QR decomposition of a matrix-valued function that takes values in $r_i \times \hat{r}_{i+1}$ instead of $\hat{r}_i \times \hat{r}_{i+1}$. The latter case would arise if we first formed g and then proceeded to perform rounding. Similarly, we are taking the QR decomposition of a smaller matrix-valued function in step 2. In practice, further benefit will be achieved through taking advantage of the special structure of each core $\mathcal{G}_i$, for example by using a fast matrix times Kronecker of matrix-valued functions operation.

Recursive computation of matrices $\mathbf{\Phi}_i$ and $\mathbf{\Psi}_i$

As we sweep through each core it may be expensive to compute each new $\mathbf{\Phi}_i$ and $\mathbf{\Psi}_i$ by evaluating the integrals in (3.18). Instead, we use a recursive relation for each update. From (3.19) it is obvious that $\mathbf{\Phi}_1 = 1$, and, analogously from (3.20) the last bivariate core, we have $\mathbf{\Psi}_{d-1} = 1$. A recursive relationship can then be created for the subsequent matrices $\mathbf{\Phi}_2, \mathbf{\Phi}_3, \ldots$ by considering that

$$
\begin{aligned}
\mathbf{\Phi}_2 &= \int_{\mathscr{X}_1} \mathcal{F}_1^T(x_1)\mathcal{G}_1(x_1)dx_1 = \int_{\mathscr{X}_1} \mathcal{F}_1^T(x_1)\mathbf{\Phi}_1\mathcal{G}_1(x_1)dx_1, \qquad \text{and} \\
\mathbf{\Phi}_3 &= \int_{\mathscr{X}_1\times\mathscr{X}_2} \mathcal{F}_2^T(x_2)\mathcal{F}_1^T(x_1)\mathcal{G}_1(x_1)\mathcal{G}_2(x_2)dx_1dx_2 \\
&= \int_{\mathscr{X}_2} \mathcal{F}_2^T(x_2)\left[\int_{\mathscr{X}_1} \mathcal{F}_1^T(x_1)\mathcal{G}_1(x_1)dx_1\right]\mathcal{G}_2(x_2)dx_2 \\
&= \int_{\mathscr{X}_2} \mathcal{F}_2^T(x_2)\mathbf{\Phi}_2\mathcal{G}_2(x_2)dx_2.
\end{aligned}
$$

Proceeding recursively we have

$$\mathbf{\Phi}_{k+1} = \int_{\mathscr{X}_k} \mathcal{F}_k^T(x_k)\mathbf{\Phi}_k\mathcal{G}_k(x_k)dx_k, \tag{3.22}$$

for $k = 1 \ldots d-2$. A similar relationship holds for $\mathbf{\Psi}_k$, i.e.,

$$\mathbf{\Psi}_k = \int_{\mathscr{X}_{k+2}} \mathcal{G}_{k+2}(x_{k+2})\mathbf{\Psi}_{k+1}\mathcal{F}_{k+2}^T(x_{k+2})dx_{k+2}, \tag{3.23}$$

for $k = 1 \ldots d-2$.

3.5.2 Example: multiplication

We now illustrate the advantage of rank-revealing ALS for function multiplication. Recall that the direct method for multiplying two functions with FT ranks equal to r involves first taking the Kronecker product of all the cores, an $\mathcal{O}(r^4)$ operation. After this initial multiplication the resulting ranks are r^2. The second step is to round the resulting function, an operation that scales cubically with rank. Thus, the total computational cost of the direct method is $\mathcal{O}(r^6)$. Posing this operation as the minimum of an optimization problem, and using rank-revealing ALS, allows us to lower this cost to $\mathcal{O}(r^5)$. The advantage is gained by exploiting the Kronecker structure of the cores by using the fast matrix times the matrix-valued function Kronecker product algorithm provided in Proposition 12.

Our experiment compares the computational cost of computing $g(x) = f(x)f(x)$ for increasing rank r of f using rank-revealing ALS and the direct method. In the experiment, the univariate scalar-valued functions making up each of the cores of the function f are represented in a basis of 10 Legendre polynomials. The coefficients of each basis function are sampled uniformly on $[-1, 1]$ and independently for each univariate function. The truncated SVD tolerance is chosen to be $\epsilon = 10^{-10}$ for both the rank-revealing ALS algorithm and for the FT rounding algorithm. The convergence tolerance of the ALS algorithm is set to $\delta = 10^{-5}$, and the dimension of f is fixed to $d = 5$. Next for ranks ranging from 2 to 22, we average the computational

time for computing g over five realizations of f. The results are shown in Figure 3-1.

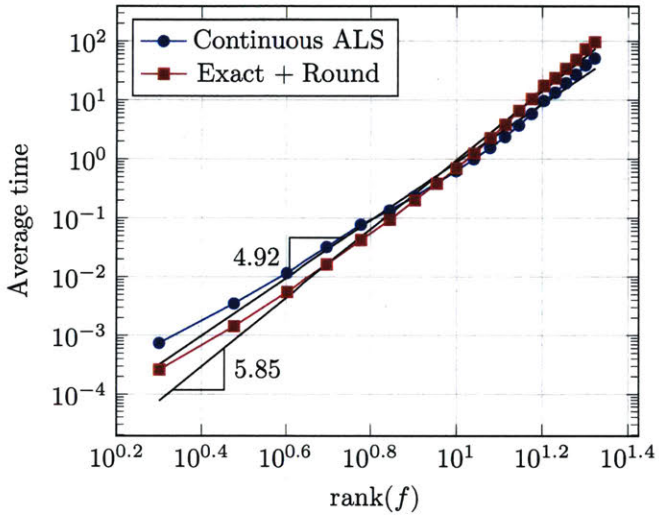


Figure 3-1: Comparison of the computational time required for continuous rank-revealing ALS and the direct method for function multiplication.

3.5.3 Example: diffusion operator

Consider the ubiquitous diffusion operator that arises, for example, in an elliptic PDE. Iterative linear solvers for PDEs typically require an application of the linear operator, and in this section we show how to leverage continuous alternating least squares for this task. Consider a function in FT format: $f(x) = \mathcal{F}_1(x_1)\ldots\mathcal{F}_d(x_d)$. Let the application of the diffusion operator result in the new function $g(x) = \mathcal{G}_1(x_1)\ldots\mathcal{G}_d(x_d)$ through the operation

$$g(x) = \nabla \cdot [a(x)\nabla f(x)], \tag{3.24}$$

where $a(x) = \mathcal{A}_1(x_1)\ldots\mathcal{A}_d(x_d)$ represents a conductivity field. The structure of the resulting cores of g are given in the following proposition.

Proposition 15. *Let $f(x)$ be a function in FT format $f(x) = \mathcal{F}_1(x_1)\ldots\mathcal{F}_d(x_d)$. Let $a(x)$ be another function in FT format $a(x) = \mathcal{A}_1(x_1)\ldots\mathcal{A}_d(x_d)$. Then applying the*

diffusion operator $g(x) := \nabla \cdot (a(x)\nabla f(x))$ *to* $f(x)$ *results in a function* $g(x)$ *whose FT representation* $g(x) = \mathcal{G}_1(x_1)\dots\mathcal{G}_d(x_d)$ *has the following cores*

$$\begin{aligned}
\mathcal{G}_1(x_1) &= \begin{bmatrix} \frac{d}{dx_1}\left(\mathcal{A}_1(x_1)\otimes\frac{d\mathcal{F}_1(x_1)}{dx_1}\right) & \mathcal{A}_1(x_1)\otimes\mathcal{F}_1(x_1)\end{bmatrix},\\
\mathcal{G}_k(x_k) &= \begin{bmatrix} \mathcal{A}_k(x_k)\otimes\mathcal{F}_k(x_k) & 0\\ \frac{d}{dx_k}\left(\mathcal{A}_k(x_k)\otimes\frac{d\mathcal{F}_k(x_k)}{dx_k}\right) & \mathcal{A}_k(x_k)\otimes\mathcal{F}_k(x_k)\end{bmatrix},\\
\mathcal{G}_d(x_d) &= \begin{bmatrix} \mathcal{A}_d(x_k)\otimes\mathcal{F}_d(x_d)\\ \frac{d}{dx_d}\left(\mathcal{A}_d(x_d)\otimes\frac{d\mathcal{F}_d(x_d)}{dx_d}\right)\end{bmatrix}.
\end{aligned} \tag{3.25}$$

Proof. The diffusion operator is given by

$$\begin{aligned}
g(x) &= \sum_{i=1}^{d} g^{(i)}(x)\\
g^{(i)}(x) &= \frac{\partial}{\partial x_i}\left(a(x)\frac{\partial f(x)}{\partial x_i}\right).
\end{aligned}$$

Recall from Section 3.4 that multiplying functions together requires taking the Kronecker product of their cores, and taking the derivative with respect i-th variable only requires modifications to the ith core. Therefore, each $g^{(i)}$ has cores given by

$$\mathcal{G}_k^{(i)} = \begin{cases} \mathcal{A}_k(x_k)\otimes\mathcal{F}_k(x_k) & \text{for } k\neq i,\\ \frac{d}{dx_k}\left(\mathcal{A}_k(x_k)\otimes\frac{d\mathcal{F}_k(x_k)}{dx_k}\right) & \text{otherwise.}\end{cases}$$

Further recall that adding functions in FT format requires concatenation of the cores. However, since the contribution to core $\mathcal{G}_k$ is identical for all $\mathcal{G}_k^{(i)}$ for $k \neq i$, a triangular structure emerges. The first core, for example, is the concatenation of the two contributions given by

$$\mathcal{G}_1(x_1) = \begin{bmatrix} \frac{d}{dx_1}\left(\mathcal{A}_1(x_1)\otimes\frac{d\mathcal{F}_1(x_1)}{dx_1}\right) & \mathcal{A}_1(x_1)\otimes\mathcal{G}_1(x_1)\end{bmatrix}.$$

The middle cores have an upper triangular structure such that the upper left and

lower right block stores all the interactions between derivative and non-derivative terms. The lower left block stores the derivative terms for that dimension

$$\mathcal{G}_k(x_k) = \begin{bmatrix} \mathcal{A}_k(x_k) \otimes \mathcal{F}_k(x_k) & 0 \\ \frac{d}{dx_k}\left(\mathcal{A}_k(x_k) \otimes \frac{d\mathcal{F}_k(x_k)}{dx_k}\right) & \mathcal{A}_k(x_k) \otimes \mathcal{F}_k(x_k) \end{bmatrix}.$$

Note that the derivative term in the kth core never interacts with the derivative terms of the other cores. A similar analysis for the last core yields

$$\mathcal{G}_d(x_d) = \begin{bmatrix} \mathcal{A}_d(x_k) \otimes \mathcal{F}_d(x_d) \\ \frac{d}{dx_d}\left(\mathcal{A}_d(x_d) \otimes \frac{d\mathcal{F}_d(x_d)}{dx_d}\right) \end{bmatrix}$$

□

Recall that fast versions of pre- and post-multiplication of each core by matrices $\mathbf{\Phi}$ and $\mathbf{\Psi}$ are needed for the rank-revealing alternating least squares algorithm to be efficient. Since post-multiplication can be derived in a similar manner, we only describe the specialized pre-multiplication by $\mathbf{\Phi}$ for the diffusion operator. First we consider cores k for $1 < k < d$.

Proposition 16. *Let* $\mathcal{A}_k(x_k) : \mathscr{X}_k \to \mathbb{R}^{r_{k-1} \times r_k}$, $\mathcal{F}_k(x_k) : \mathscr{X}_k \to \mathbb{R}^{r_{k-1} \times r_k}$, *and* $\mathbf{\Phi} \in \mathbb{R}^{\hat{r} \times 2r_{k-1}^2}$. *Furthermore, let the scalar-valued functions making up* $\mathcal{A}_k$ *and* $\mathcal{F}_k$ *consist of an expansion of* P *orthonormal basis functions. Let* $G_k : \mathscr{X}_k \to \mathbb{R}^{2r_{k-1}^2 \times 2r_k^2}$ *be computed according to* (3.25). *Then computing* $\mathbf{\Phi}\mathcal{G}_k$ *costs*

$$\mathcal{O}(P^3 \hat{r} r_{k-1} r_k^2)$$

for cores $1 < k < d$.

Proof. We first split the core $\mathcal{G}_k$ into two components:

$$\begin{aligned}
\boldsymbol{\Phi}\mathcal{G}_k(x_k) &= \left[\mathcal{B}(x_k) \quad \mathcal{C}(x_k)\right], \\
\mathcal{B}(x_k) &= \boldsymbol{\Phi}[:, 1 : r_{k-1}^2]\left(\mathcal{A}_k(x_k) \otimes \mathcal{F}_k(x_k)\right) \\
&\qquad + \boldsymbol{\Phi}[:, r_{k-1}^2 + 1 : 2r_{k-1}]\frac{d}{dx_k}\left(\mathcal{A}_k(x_k) \otimes \frac{d\mathcal{F}_k(x_k)}{dx_k}\right), \quad \text{and} \\
\mathcal{C}(x_k) &= \boldsymbol{\Phi}[:, r_{k-1}^2 + 1 : 2r_{k-1}]\left(\mathcal{A}_k(x_k) \otimes \mathcal{F}_k(x_k)\right).
\end{aligned}$$

Recall that the product between a matrix and the Kronecker product of two matrices can be computed in $\mathcal{O}(P^3\hat{r}r_{k-1}r_k^2)$ operations according to Proposition 12. Thus, computing the first element in $\mathcal{B}(x_k)$ and $\mathcal{C}(x_k)$ requires this many operations.

To compute the second element in the summation forming $\mathcal{B}$, it is convenient to work with derivatives of the individual cores in order to avoid computing the full Kronecker. To this end we use the fact that

$$\frac{d}{dx_k}\left(\mathcal{A}_k(x_k) \otimes \frac{d\mathcal{F}_k(x_k)}{dx_k}\right) = \frac{d\mathcal{A}_k(x_k)}{dx_k} \otimes \frac{d\mathcal{F}_k(x_k)}{dx_k} + \mathcal{A}_k(x_k) \otimes \frac{d^2\mathcal{F}_k(x_k)}{dx_k^2}. \tag{3.26}$$

Now premultiplying each of these elements by $\boldsymbol{\Phi}$ costs $\mathcal{O}(P^3\hat{r}r_{k-1}r_k^2)$. The total complexity of this operation is thus dominated by four matrix times matrix-Kronecker products for a cost of $\mathcal{O}(P^3\hat{r}r_{k-1}r_k^2)$. □

Note, that the algorithm for efficiently calculating $\boldsymbol{\Phi}\mathcal{G}_d$ follows the same pattern. The algorithm used in the proof of Proposition 16 is considerably less expensive than first computing the Kronecker product and then premultiplying by $\boldsymbol{\Phi}$. Computing the Kronecker product first would costs $\mathcal{O}(P^3r_{k-1}^2r_k^2)$ operations, thus the algorithm would be $\mathcal{O}(r_{k-1})$ more expensive.

Numerical verification

We now verify the computational benefit of continuous alternating least squares for the application of the diffusion operator. In particular, we perform three benchmarking tests: a test for scaling with dimension, a test for scaling with polynomial order,

and a test for scaling with rank of the conductivity $a(x)$. For each of these tests we generate random functions f and random conductivities a by uniformly sampling the coefficients of a Legendre polynomial spectral expansions making up their cores on $[-1, 1]$. Ten repetitions are made for each benchmark, and the algorithm parameters that are the same across the tests are the truncated SVD tolerance $\epsilon = 10^{-10}$ and an ALS convergence tolerance $\delta = 10^{-8}$.

In the first benchmark we fix the polynomial order to $P = 5$ and the FT-ranks of a and f to 2. The dimension is then varied from 2 to 152 in steps of 5. The results, shown in Figure 3-2, show the predicted linear growth with dimension.

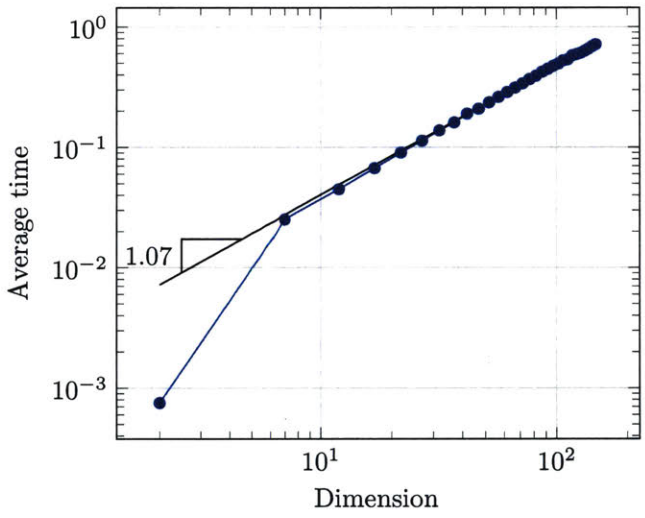


Figure 3-2: Linear growth of computational time with dimension obtained by applying the diffusion operator using continuous rank-revealing alternating least squares. Computational time is averaged over 10 realizations of a and f.

The second benchmark tests the scaling with the number of basis functions. The dimension is fixed to $d = 5$ and the ranks of a and f are again fixed to two. The polynomial order is varied between 2 and 82 using a step size of two. The results, shown in Figure 3-3, agree with the theoretical prediction of cubic growth with polynomial order.

The third benchmark compares the scaling of the continuous rank-revealing ALS method with the direct method of first computing the exact solution and then round-

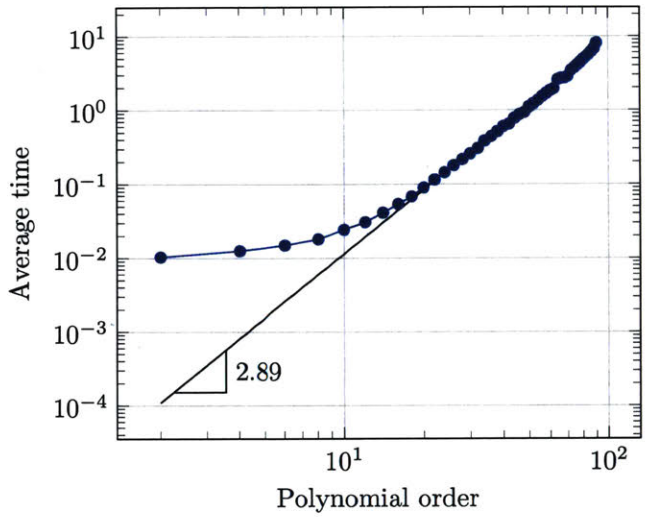


Figure 3-3: Cubic growth of computational time with univariate expansion order P obtained by applying the diffusion operator using continuous rank-revealing alternating least squares. Computational time is averaged over 10 realizations of a and f.

ing. The dimension is fixed to $d = 5$ and the polynomial order is fixed to $P = 3$. The FT-ranks of f are again fixed to 2. The ranks of a are varied between 2 and 62 in steps of two. For this experiment we set the SVD truncation tolerance to $\epsilon = 10^{-5}$. The results, shown in Figure 3-4, display at least an order of magnitude benefit obtained by ALS. This benefit is achieved because the continuous rank-revealing ALS never has to obtain an exact representation of the function, instead it directly finds an approximation with smaller ranks by using a truncated SVD with tolerance 10^{-5}.

3.6 Numerical examples

We now discuss some implementation details and show applications of the FT for a variety of integration and approximation test problems. Showing the benefits of using the FT for approximation and integration highlights its applicability for a wide variety of applications. The benefits will be discussed in further detail for the two

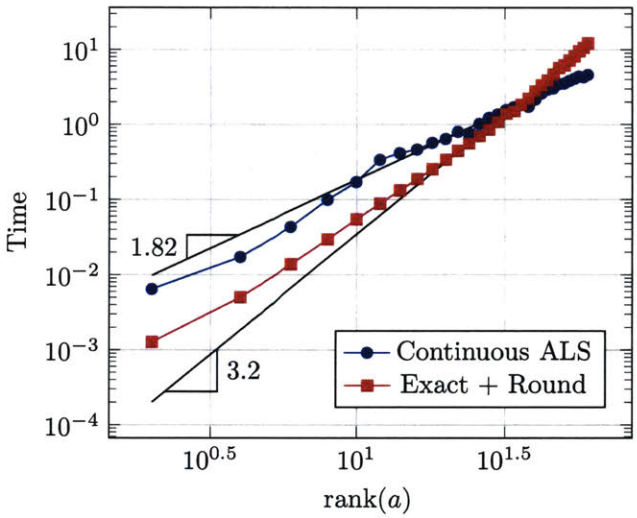


Figure 3-4: Comparison of the computational time required for continuous rank-revealing ALS and the direct method, with respect to rank of the conductivity field rank(a). Computational time is averaged over 20 realizations of a and f.

specific applications of stochastic optimal control and Gaussian filtering in Chapters 4 and 5, respectively.

All of the experiments performed in this section utilized the publicly available Compressed Continuous Computation (C^3) library [54].

3.6.1 Implementation details

We briefly turn to the practical issues surrounding the implementation of some of the algorithms described in this chapter, and we show simple examples highlighting how this implementation allows the FT to differ from the TT. We focus on two specific areas: representing each univariate function in the FT core in a known basis for cross approximation and integrating functions by first representing them in FT format. These implementation details set the stage for the higher-dimensional examples in Sections 3.6.2 and 3.6.3.

One-dimensional fiber adaptation for cross approximation

We start with cross approximation. When performing cross approximation to convert a function to its FT format, one only needs to compute with one dimensional fibers of the function, i.e., we fix all variables except one. These fibers are necessary, for example, to create R and C of Algorithm 1. When such operations are required we first *approximate* the fiber in some *basis*. These bases are adapted *independently* to each fiber that needs to be approximated. The choice of the approximation can vary from fiber to fiber. The algorithms we have described for compressing and computing functions are agnostic to the way that the fibers are approximated.

For instance, if one expects the fibers to be smooth functions, then one can use an orthogonal expansion of Legendre polynomials. Alternatively, if one expects or detects local phenomena such as discontinuities, one can turn to piecewise polynomials. Notice that representing fibers with adaptive piecewise polynomials is a *nonlinear* parameterization of the fiber because one seeks the optimal locations for the knots. This approach highlights the significant generality that our framework presents compared with approaches that convert the problem of low-rank functional approximation to the problem of low-rank approximation of tensors of coefficients [17, 26, 40, 103].

To make this idea concrete, we define a routine `approx-fiber` that takes in a univariate function and produces an approximation in an appropriate basis. Using univariate fiber approximations, we reproduce Algorithm 1 for the special case of a scalar-valued function of two variables for illustration purposes in Algorithm 7.

Note that we can implement different approximation schemes for different dimensions and even for different fibers within each dimension. Furthermore, once R and C are represented in a known basis, the steps required for the `maxvol` algorithm can be adapted to the types of approximations. For example, Line 5 of Algorithm 2 requires finding the maximum element of a vector-valued function. If each column of the quasimatrix is a Legendre polynomial, then the maximum element in each column can be found through a root finding or eigenvalue procedure to arbitrary precision. Such a procedure removes any need for discretization or gridding. The other methods

Algorithm 7 Cross approximation of a two dimensional function using fiber adaptive approximation

Require: A two-dimensional function $f \in [a,b] \times [c,d]$; Rank upper bound r; Initial y index $\boldsymbol{y} = [y_1, y_2, \ldots y_r]$; Stopping tolerance $\delta_{\text{cross}} > 0$; Approximation scheme `approx-fiber`$(f, \epsilon_{\text{approx}})$, Fiber approximation tolerance ϵ_{approx};

Ensure: $\boldsymbol{x}, \boldsymbol{y}$ such that $\mathbf{F} = f(\boldsymbol{x}, \boldsymbol{y})$ has " large " volume

1: $f^{(0)} = 0$
2: $k = 1$
3: **repeat**
4: Initialize vector-valued function $R \in \mathbb{R}^{[a,b] \times r}$
5: $R[k](x) =$ `approx-fiber`$(f(x, y_k), \epsilon_{\text{approx}})$ for $k = 1 \ldots r$
6: $Q\mathbf{T} =$ `qr-vvf`(R)
7: $\boldsymbol{x} =$ `maxvol-mvf`(Q)
8: Initialize vector-valued function $C \in \mathbb{R}^{[c,d] \times r}$
9: $C[k](y) =$ `approx-fiber`$(f(x_k, y), \epsilon_{\text{approx}})$ for $k = 1 \ldots r$
10: $Q\mathbf{T} =$ `qr-vvf`(C)
11: $\boldsymbol{y} =$ `maxvol-mvf`(Q)
12: $\hat{\mathbf{Q}} = [Q(y_1)\ Q(y_2)\ \ldots\ Q(y_r)]$
13: $f^{(k)}(x, y) = f(x, \boldsymbol{y})\hat{\mathbf{Q}}^{\dagger} Q(y)$
14: $\delta = ||f^{(k)} - f^{(k-1)}||/||f^{(k)}||$
15: $k = k + 1$
16: **until** $\delta \leq \delta_{\text{cross}}$

such as `qr-qm` and `lu-qm` can be performed for quasimatrices made up of polynomial expansions just as in [4, 118, 119]. If splines are used, the procedures can be adapted appropriately.

We now demonstrate the advantage of adapting each fiber via two examples where the resulting function evaluations *do not* lie on a tensor product grid. The example further shows one of the primary ways this algorithm is different from the tensor-train representation and from other functional tensor-train approaches [17]. Consider the approximation of two canonical rank-1 functions

$$f_1(x_1, x_2) = \sin(10x_1 + \frac{1}{4})\,(x_2 + 1) \tag{3.27}$$

$$f_2(x_1, x_2) = \begin{cases} 0 & \text{if } x_1 > 0.5 \text{ or } x_2 > 0.5 \\ \exp(5x_1 + 5x_2) & \text{otherwise} \end{cases} \tag{3.28}$$

where (3.28) is a two dimensional Genz function of the discontinuous family [47].

For (3.27) we use a global expansion of Legendre polynomials ϕ_i to represent each fiber

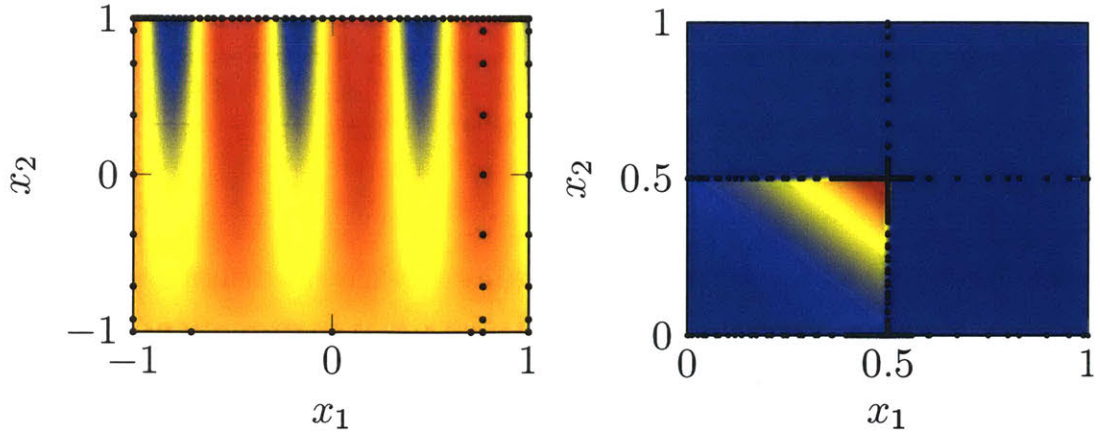
$$f(x) = \sum_{i=1}^{n} a_i \phi_i(x), \quad \int \phi_i(x)\phi_j(x)dx = \delta_{i,j},$$

where the coefficients are determined through projection using Clenshaw-Curtis quadrature. The order of the approximation n is progressively increased until four sequential coefficients are $a_i \leq \epsilon_{\text{approx}} = 10^{-10}$. Furthermore, even though (3.27) is a FT-rank 1 function we use a rank-2 approximation in order to further illuminate the difference in locations of function evaluations between our method and existing tensor-based approximation methods. In particular, we see that fibers at different positions are approximated using different numbers of evaluations. The number of evaluations required in the oscillatory portion of the function when $x_2 = 1$ is greater than in the constant portion when $x_2 = -1$. Such an adaptation of the grid would be difficult to achieve with discrete TT, and it highlights the flexible adaptivity of the continuous approximation framework.

For (3.28) we can no longer use a global polynomial expansion due to the discontinuity, and therefore we turn to a piecewise polynomial approximation. We first use a one-dimensional edge detection method based on polynomial annihilation [2, 3, 58] to locate the discontinuities. We then approximate each polynomial piece using the same polynomial expansion scheme described above. Furthermore, we employ a rank-1 approximation of this function.

Figure 3-5 shows the function and the resulting evaluation locations, where both approximations achieve machine precision accuracy. In the left panel, we see that the function is evaluated densely along the top of the plot and coarsely along the bottom— i.e., the evaluations again do not lie on a tensor product grid. In the right panel, we see that the algorithm clusters points around the discontinuity as desired.

Another point to note is that once we have the function-trains of each of these functions we can perform computation with them directly in compressed form. For example, we can now integrate the discontinuous function f_2. Such integration cannot be performed using array-based tensor-train algorithms without manually splitting



(a) Sin function (3.27) and evaluation points.

(b) Genz function (3.28) and evaluation points.

Figure 3-5: Contour plots and evaluations of f_1 and f_2.

the domain because it would require specialized integration rules to deal with the discontinuity. When representing everything in functional form, we are able to perform the integration and approximation automatically, without specifying any specialized integration or approximation rules. In Section 3.6.2, we will show integration performance on higher dimensional discontinuous Genz functions.

Function-train integration

Suppose that we have a FT representation in which each core consists of Legendre polynomial expansions. We can integrate each core easily if we know how to integrate Legendre polynomials. More concretely, suppose that the fibers of core k are represented using a Legendre polynomial expansion

$$f_{i,j}^{(k)}(x) = \sum_{\ell=1}^{n} a_{i,j,\ell}^{(k)} \phi_\ell(x), \quad \langle \phi_\ell, \phi_m \rangle = \delta_{\ell,m}, \quad \phi_1 = \frac{1}{2}, \quad a_{i,j,\ell}^{(k)} \in \mathbb{R}.$$

Then we can compute its integral

$$\int f_{i,j}^{(k)}(x)dx = \int \sum_{\ell=1}^{n} a_{i,j,\ell}^{(k)} \phi_\ell(x)dx = \sum_{\ell=1}^{k} a_{i,j,\ell}^{(k)} \int \phi_\ell(x)dx = a_{i,j,1}^{(k)}.$$

This means that the integral of each core results in the matrix consisting of the first coefficient of each function

$$\int \mathcal{F}_k(x_k)dx_k = \begin{bmatrix} a^{(k)}_{1,1,1} & \cdots & a^{(k)}_{1,r_k,1} \\ \vdots & & \vdots \\ a^{(k)}_{1,r_{k-1},1} & \cdots & a^{(k)}_{r_{k-1},r_k,1} \end{bmatrix}.$$

From here we use (3.13) to obtain the integral.

3.6.2 Integration

We now show the performance of the FT algorithm on two high dimensional integration examples.

Rank-2 Sin function

Analogously to [16, 98], we consider the FT rank-2 function

$$f(x) = \sin(x_1 + x_2 + \ldots + x_d) \tag{3.29}$$

for which we know the analytic integral

$$\int_{[0,1]^d} f(x)dx = \mathrm{Im}\left[\left(\frac{e^i - 1}{i}\right)^d\right] \tag{3.30}$$

We seek to study the performance associated with computing this integral as a function of dimension and fiber adaptation error. Specifically we choose an adaptive procedure based on approximating each fiber as Legendre polynomial series expansion. Each approximation begins with a fifth order expansion, and each time adaptation occurs the expansion order goes from $k \to 2k-1$. We stop adaptation after the last *two* coefficients of the expansion drop below a tolerance ϵ_{approx}. Figure 3-6 presents results studying the integration error as a function of this ϵ_{approx} and the dimensionality of the problem.

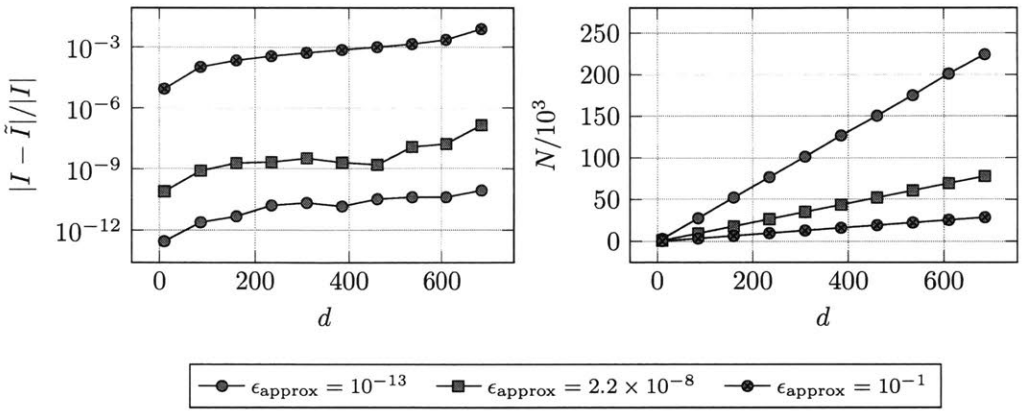


Figure 3-6: Errors (left panel) and linear growth in the number of evaluations (right panel) involved in the integration of (3.29) as a function of dimension d and fiber adaptation parameter ϵ_{approx}.

Figure 3-6 shows that, as expected, the number of function evaluations grows linearly with the number of input dimensions. Furthermore, for the three tightest approximation tolerances, the errors are virtually constant across dimensions. Note that the y-axis is on a log scale and therefore the variation of the errors for the three tightest tolerances would be virtually invisible on the $\epsilon_{\text{approx}} = 10^{-1}$ curve. In contrast to [98], our integration algorithm is adaptive. Furthermore, it is adapted at a local fiber level rather than at a dimension level. Obtaining such adaptivity would be extremely nontrivial without the continuous framework.

Rank-1 discontinuous Genz function

We now demonstrate integration on the Genz discontinuous functions of various dimensions. Specifically, we integrate the function $f : [0,1]^d \to \mathbb{R}$ defined as

$$f(x_1, x_2, \ldots, x_d) = \begin{cases} 0 & \text{if } x_i > \frac{1}{2} \text{ for any } i = 1 \ldots d \\ \exp\left(\sum_{i=1}^{d} 5x_i\right) & \text{otherwise} \end{cases} \tag{3.31}$$

The analytical integral of (3.31) is

$$I[f] = \left(\frac{\exp\left(\frac{5}{2}\right) - 1}{5} \right)^d,$$

where we see that this problem is quite challenging because the integral grows exponentially with dimension. For example, for 10 dimensions this integral is $I \approx 3.131 \times 10^3$ and for 100 dimensions this integral is $I \approx 9.05455 \times 10^{34}$.

We integrate this function by first converting it to its rank-1 function-train representation. The fibers are approximated by sixth order piecewise polynomials. The discontinuities are located automatically using a polynomial annihilation edge detection routine [2, 3, 58]. After conversion to the function-train representation, we integrate the FT using the technique described in Section 3.4. The resulting errors and required function evaluations as a function of dimension are depicted in Figure 3-7.

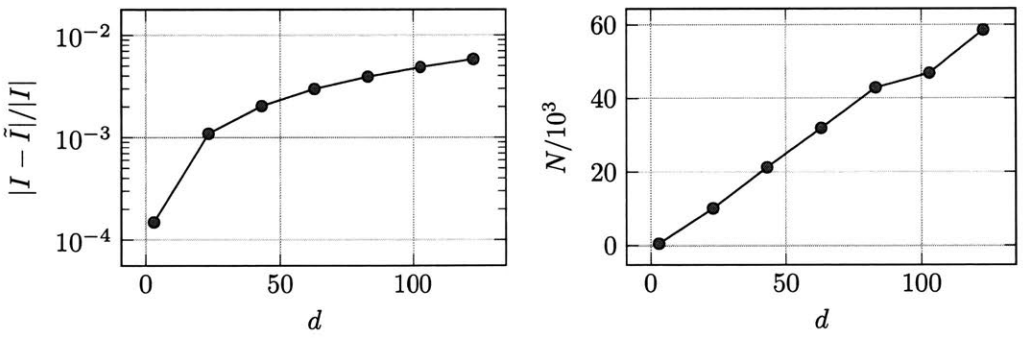


Figure 3-7: Errors (left panel) and number of evaluations (right panel) involved in the integration of (3.31) as a function of dimension d.

The results indicate that we are able to achieve $\mathcal{O}(10^{-3})$ relative accuracy for all of the dimensions and that the number of function evaluations required scales linearly with dimension. Furthermore, we are able to approximate large integral values, indicating a general robustness of the algorithm. We also note that such an integration would be extremely difficult to perform using either the discrete tensor-train or the spectral tensor-train [17] techniques because the discontinuities pose

Fiber approximation	Cross approximation	Rank adaptation
Initialize to 7th degree Increase degree $k \to 2k+1$ $\epsilon_{\text{approx}} = 10^{-7}$	$\delta_{\text{cross}} = 10^{-3}$ ≤ 5 sweeps	$\epsilon_{\text{round}} = 10^{-5}$ `kickrank` $= 5$ ≤ 5 adaptations

Table 3.1: Algorithmic parameters used for approximation of the simulation library benchmark problems shown in Table 3.2.

problems for most integration rules. Individual fiber adaptation is critical in order to locate the discontinuities.

3.6.3 Approximation

We now test the FT approximation on a set of benchmark approximation problems. The first set of test functions we use are taken from the Emulation/Prediction Test Problems set in [113]. In particular, we have chosen to examine a subset of functions which have more than two input dimensions. Secondly, we explore some of the effects of different algorithm parameter choices for the approximation of a quantity of interest arising from an elliptic PDE.

Simulation library benchmark problems

To standardize our algorithm for each benchmark function we fix the algorithmic parameters to those shown in Table 3.1. In particular, we utilize Legendre polynomial expansions with Gauss-Legendre quadrature for fiber adaptation and we perform rank adaptation using Algorithm 4.

For each problem we normalize each input to $[-1, 1]$ and compute the Relative RMSE error using ten thousand Monte Carlo samples as

$$\text{error} = \sqrt{\frac{\sum_{i=1}^{10000}(f(x_i) - \hat{f}(x_i))^2}{\sum_{i=1}^{10000} f(x_i)^2}}$$

Table 3.2 show our results. Except for the Robot Arm function, each of the test functions are found to be low rank. Note that we are not attempting to approximate

the functions with the least number of evaluations, instead we are attempting to decompose the functions to achieve an approximation tolerance in line with algorithmic parameters. One interesting aspect of note is the behavior for the Gramacy and Lee 2009 function, where the fifth and sixth input variables are not active. Our algorithm discovers this behavior as the corresponding ranks are found to be $r_4 = r_5 = 1$.

The results indicate that many of the benchmark problems are indeed low-rank. Since these benchmark problems come from a wide variety of fields and are representative of certain behaviors of more complex models, we believe the results promote the potential wide-spread applicability of the the approximation framework. The Robot Arm function appears to be the only high rank function in these benchmarks. Thus the application of cross approximation led to excessive basis adaptation and a un-converged maximum rank after 5 rank increase steps.

Further note that our goal is not to obtain the best possible approximation with the fewest number of function evaluations. Nor are we approaching this problem in a data-fit context where an approximation is built from a fixed set of data. Rather, we are decomposing functions into their low-rank representation. After setting the parameters of the decomposition, the algorithm *automatically* chooses where and how to evaluate the function to ensure particular error tolerances specified by the algorithm, and the resulting numbers of evaluations are but one realization of the framework we have created in this thesis. While we suspect that one could achieve good approximation results with fewer numbers of evaluations, the evaluations presented here take into account both approximation, estimation of rank, and certification of convergence.

Elliptic PDE

We next explore the effects of various parameters of the FT approximation algorithm on a model of subsurface flow typically encountered in uncertainty quantification (UQ) applications. Consider the following one-dimensional elliptic partial differential equation:

$$\frac{\partial}{\partial s}\left(k(s,\omega)\frac{\partial u}{\partial s}\right) = s^2, \tag{3.32}$$

Name (dimension)	Ranks	Number of evaluations / error	Comments
Borehole (8)	(1,2,2,2,3,3,2,2,1)	7032 / 1.8×10^{-5}	
OTL Circuit (6)	(1,3,2,2,2,2,1)	3264 / 1.5×10^{-4}	
Piston (7)	(1,2,4,4,3,2,2,1)	27472 / 1.0×10^{-2}	
Robot Arm (8)	(1,1,7,21,24,25,21,11,1)	45445 / 9.8×10^{-2}	No 1d basis adaptation
Wing Weight (10)	(1,2,2,2,2,2,2,2,2,2,1)	9520/ 3.0×10^{-11}	
Friedman (5)	(1,4,2,2,2,1)	1816 / 3.1×10^{-5}	
Gramacy and Lee 2009 (6)	(1,2,3,2,1,1,1)	4800 / 3.2×10^{-7}	x_5, x_6 not active
Dette and Pepelyshev (2010) 8-Dimensional (8)	(1,3,2,3,3,3,2,2,1)	5376 / 1.5×10^{-5}	
Dette and Pepelyshev (2010) Exponential (3)	(1,2,2,1)	3304/ 1.4×10^{-8}	

Table 3.2: Performance of the FT approximation algorithm on a set of multi-dimensional test functions from the Simulation Library Test Functions [113].

for $s \in (0,1)$ with $u(s)|_{s=0} = 0$ and $\frac{\partial u}{\partial s}|_{s=1} = 0$. In the UQ context, we would like to consider the effects of an unknown permeability field $k(s,\omega)$ on some function of the output pressure u. To this end, $k(s,\omega)$ is modeled as a random process due to the random variable ω, typically with a log-Gaussian distribution:

$$\log\left[k(s,\omega) - a\right] \sim \mathcal{N}(0, c(s,s'))$$

where the covariance kernel is $c(s,s') = \sigma^2 \exp(-\frac{|s-s'|}{2l^2})$. In order to obtain a finite dimensional representation of $k(s,\omega)$ we use the Karhunen-Loève expansion to express the random field as a weighted sum of the eigenfunctions of c. In particular, if we compute the eigenfunctions and eigenvalues of the correlation function $\int c(s,s')\phi_i(s')ds' = \lambda_i\phi_i(s)$, then any realization of this Gaussian process may be represented as $k(s,\omega) = \sum_{i=1}^{\infty} \sqrt{\lambda_i}\phi_i(s)\xi_i(\omega)$, where $\xi_i(\omega)$ are now Gaussian random variables. We approximate these eigenfunctions and eigenvalues on a 100 point discretized grid and truncate the expression after 24 modes.

The approximation objective of this problem is to represent some quantity of interest $Q(u(s,\xi_1,\ldots,\xi_{24}))$, where $u(s,\xi_1,\ldots,\xi_{24})$ is the solution of the PDE at a particular spatial locations s for a realization of the permeability field defined by the realizations of ξ_i. For simplicity, we fix $s = 0.7$ to obtain a quantity of interest that is only a function of the random variables $Q(\xi_1,\ldots,x_{24}) = u(0.7,\xi_1,\ldots,\xi_{24})$.

We will obtain an approximation for Q using 3 different parameter settings for (a,σ^2,l) that correspond to various difficulties of the problem. In particular, we will investigate 3 problems corresponding to an "easy" problem (P1) where $(a,\sigma^2,l) = (0.0, 0.1, 0.25)$, a "moderate" difficulty problem (P2) where $(a,\sigma^2,l) = (0.5, 1.0, 0.15)$ and a more difficult problem (P3) where $(a,\sigma^2,l) = (0.0, 1.0, 0.15)$.

We now seek to build an FT approximation of 24 dimensions to map from the KL modes ξ_i to the quantity of interest. In order to still use Legendre polynomial expansions to represent our one dimensional fibers we need to reparameterize the problem to one with uniform random variables on $[0,1]$. We can do this using the inverse CDF of a Gaussian random variable to make new variables $\xi_i = \Phi^{-1}(\hat{\xi}_i)$, where

Φ^{-1} is the inverse CDF $\hat{\xi}_i$ are the new, uniformly distributed, random variables. Now the approximation is constructed from $(\hat{\xi}_1, \hat{\xi}_2, \ldots, \hat{\xi}_{24}) \to u(0.7, \hat{\xi}_1, \ldots \hat{\xi}_{24})$.

Our experiments investigate how the error, number of function evaluations and maximum rank change with ϵ_{approx} and ϵ_{round}. We fix the cross parameters to $\delta_{\text{cross}} = 10^{-3}$ and a maximum 3 sweeps. The rank adaptation parameters are fixed at `kickrank` $=$ 5 and maximum 4 rank adaptations. Finally, we initialize the fiber approximations with a second order polynomial expansions. The results for the three problem setups described above are provided in Figure 3-8.

Several patterns are apparent in the results of Figure 3-8. We first consider the ranks for each problem, shown in the third column. We immediately see that the approximation accuracy practically does not impact the maximum rank found by the rank adaptation. The maximum rank found by the adaptation only changes as rounding tolerance decreases. Furthermore, we see that the rank increases as we move down the table. This corresponds to the notion of the difficulty of each problem, and is analogous to the eigenvalue decay. Furthermore, the rank is higher for the last problem showing that the problem is more difficult when $a = 0$ rather than $a = 0.5$. One further concept is that unlike in array-based tensor-train algorithms or the spectral tensor-train algorithm, the maximum rank attainable by the numerical procedure is *not bounded* by discretization level. Rounding is critical to restricting the growth of the rank.

The number of function evaluations shown in column 2 also display some interesting properties. In particular, for each of the three models, there are two separate regimes of patterns. Before the rounding tolerance becomes tight enough so that the rank increases, the number of function evaluations is unaffected by the tolerance. The number of evaluations is only affected by the fiber approximation accuracy. This makes sense because the number of function evaluations is proportional to $\mathcal{O}(ndr^2)$, where n can be thought of as the average function evaluations for each fiber, and in this regime, the rank is constant. Once the rank starts decreasing we see that the number of function evaluations grows with both tighter approximation and rounding tolerances. However, the changes in the number of evaluations change more rapidly

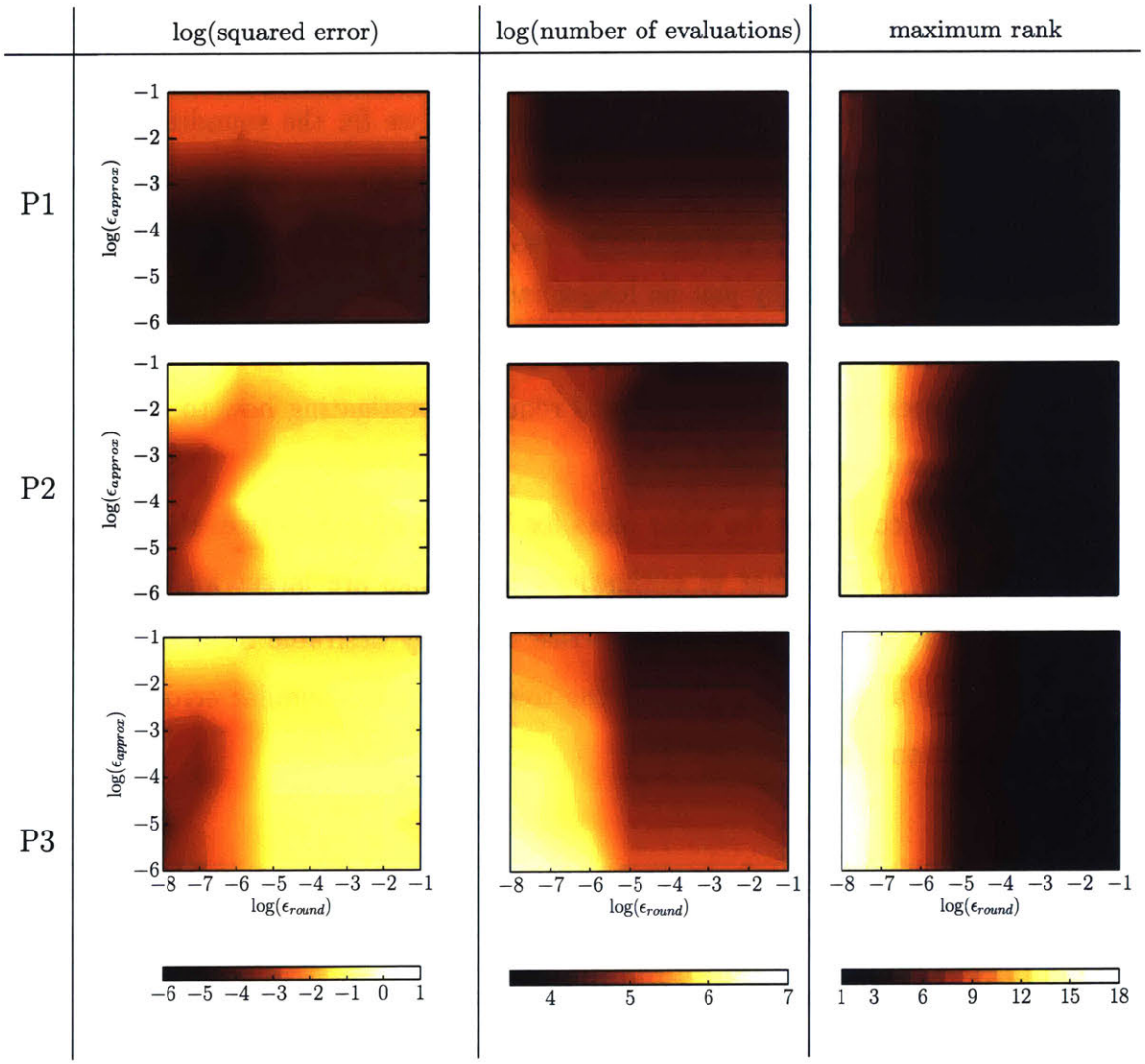


Figure 3-8: $\log(error^2)$ (left column), log(number of evaluations) (middle column), and maximum rank (right column) for three different configurations of the elliptic problem (3.32) corresponding to different combinations of (a, σ^2, l). In particular, the top row corresponds to $(0.0, 0.1, 0.25)$, middle row corresponds to $(0.5, 1.0, 0.15)$, and the bottom row corresponds to $(0.0, 1.0, 0.15)$.

with increasing rounding tolerance because reducing the rounding tolerance results in a rapid increase in rank.

Column 1 shows that the log of the error exhibits a similar pattern as the number of function evaluations. In particular, the error is fairly constant until the rank starts decreasing. Once the rank starts decreasing both approximation tolerance and rounding tolerance affect the error. Furthermore, if we fix the rounding tolerance and decrease the approximation tolerance we see a rapid change in approximate error followed by a relatively large plateau area, suggesting that beyond a given ϵ_{approx} the fiber approximation accuracy can no longer increase. Overall, these results suggest that it is possible to find a reasonable value for the rank before increasing the fiber approximation accuracy. Future work will require investigating how to adapt these two parameters in the best way.

Finally, if one compares the error plots for P2 and P3 one notices that the plots look similar, but the number of evaluations and ranks are larger for the bottom row than the middle row. This characteristic is highly desirable from an adapt-to-tolerance scheme as a given setting for the tolerances yields similar errors but with larger computational effort.

3.7 Summary

In this chapter we develop a set of low-rank function approximation techniques that extend the tensor-train decomposition using continuous linear algebra. Algorithms extended into the continuous domain include a cross approximation algorithm for decomposing a black box multivariate function into FT format, a rounding algorithm for reapproximating a FT with one of lower ranks, and a continuous alternating least squares framework for various computations.

Creating these algorithms using continuous linear algebra provides a flexible way for incorporating and exploiting more than just low-rank structure. For example, representing the univariate functions making up the cores of the FT with a polynomial approximation allows us to exploit function regularity. This characteristic is impor-

tant for problems, such as the solution of differential equations, that traditionally require discretization of high dimensional functions. The solutions of such problems are typically sensitive to the choice of discretization, and the problem of adapting the discretization to capture local features is critical for achieving high accuracy. Our continuous representation automatically performs adaptation to local features and does not require the specification of a tensor product set of candidate evaluation locations.

Our framework also enables polynomial time performance for various multilinear algebra algorithms applied to low-rank multivariate problems. We anticipate that these tools will be useful for the design of many algorithms. In the context of the solution of partial differential equations [93] provides a method for transitioning away from the traditional *discretize-then-solve* methodology, and future work will aim at extending these methods using the FT.

In the next two chapters we will demonstrate how to exploit these algorithms for stochastic optimal control and Gaussian filtering.

Chapter 4

Low-rank algorithms for stochastic optimal control

In this chapter we consider our first application of the FT decomposition. We use it to mitigate the curse of dimensionality encountered by stochastic optimal control. In particular, we show how to generate low-rank approximations to value functions and how standard dynamic programming algorithms can leverage the continuous computation framework. This work builds upon prior work [56] where the tensor-train representation was used for representing discretized value functions.

This chapter begins with an overview of the continuous-time continuous-space stochastic optimal control problem. The algorithms for its solution are based on discretizing the continuous system into a discrete Markov decision process (MDP), and then solving the MDP with value or policy iteration. These algorithms are detailed in Section 4.2. In Section 4.3 we propose approximate value or policy iteration algorithms that use rank-adaptive cross approximation to generate FT representations of value functions. We also demonstrate how the continuous FT representation enables efficient restriction and prolongation for multi-level methods. An analysis of the convergence properties and computational cost of the proposed algorithms is provided in Section 4.4. Several numerical examples arising from motion planning problems in robotics are demonstrated in in Section 4.5.

4.1 Continuous-time and continuous-space stochastic optimal control

Let us denote the set of integers and the set of reals by $\mathbb{Z}$ and $\mathbb{R}$, respectively. We denote the set of all positive real numbers by $\mathbb{R}_+$. Similarly, the set of positive integers is denoted by $\mathbb{Z}_+$. Let $d, d_u, d_w \in \mathbb{Z}_+$, $\mathscr{X} \subset \mathbb{R}^d$ and $\mathscr{U} \subset \mathbb{R}^{d_u}$ be compact sets with smooth boundaries and non-empty interiors, let $\mathscr{T} \subset \mathbb{R}_+$, and let $\{w(t) : t \geq 0\}$ be a d_w-dimensional Brownian motion defined on some probability space $(\Omega, \mathcal{F}, \mathbb{P})$, where Ω is a sample space, $\mathcal{F}$ is a σ-algebra, and $\mathbb{P}$ is a probability measure.

Continuous-time continuous-space stochastic optimal control (SOC) is an infinite-dimensional optimization problem with differential constraints defined by stochastic differential equations. The solution to SOC is a policy that maps each state to an optimal control action. SOC is often formulated as a dynamic programming (DP) problem in order to represent it in a form for which many classes of solution methods have been developed. In this section, we provide the background for SOC problems that can be formulated as two classical DP models: the fixed-cost finite horizon (FCFH) Markov decision process (MDP) and the discounted-cost infinite-horizon (DCIH) MDP.

Section 4.1.1 describes stochastic differential equations, Section 4.1.2 describes SOC cost functions, Section 4.1.3 describes Markov policies, and Section 4.1.4 describes the dynamic programming formulation.

4.1.1 Stochastic differential equations

A stochastic dynamical system is described by a stochastic ordinary differential equation (SDE) in the following differential form:

$$dx(t) = B(t, x(t), u(t))dt + \mathcal{D}(t, x(t))dw(t), \text{ for all } t \in \mathscr{T}, \tag{4.1}$$

where $B : \mathscr{T} \times \mathscr{X} \times \mathscr{U} \to \mathbb{R}^d$ is a vector-valued function, called the drift, and $\mathcal{D} : \mathscr{T} \times \mathscr{X} \to \mathbb{R}^{d \times d_w}$ is a matrix-valued function, called the diffusion. Strictly

speaking, for any admissible control process[1] $\{u(t) : t \geq 0\}$, the solution to this differential form is a stochastic process $\{x(t) : t \geq 0\}$ that solves the following integral equation:

$$x(t) = x(0) + \int_0^t B(\tau, x(\tau), u(\tau))\, d\tau \; . \\ + \int_0^t \mathcal{D}(\tau, x(\tau), u(\tau))\, dw(\tau), \text{ for all } t \text{ s.t. } x(t) \in \mathscr{X}, \tag{4.2}$$

where the last term on the right hand side is the usual Itô integral [91]. In this work, we assume that the drift and diffusion are measurable, continuous, and bounded functions. These conditions guarantee existence and uniqueness of the solution to (4.2) [91]. For further details regarding existence and uniqueness, we refer the reader to the work by Øksendal [91].

4.1.2 Cost functions

Next, we describe the SOC formulations that lead to FCFH and DCIH MDPs. Our description and notation closely follows the work by Fleming and Soner [44]. We first begin by defining notation and assumptions that are necessary throughout the exposition.

Let $\mathscr{O} \subset \mathscr{X}$ denote an open subset. If $\mathscr{O} \neq \mathbb{R}^d$ then let its boundary $\partial\mathscr{O}$ be a compact $(d-1)$-dimensional manifold of class C^3 (the set of 3-times differential functions). Also, let g, ψ denote continuous stage and terminal cost functions, respectively, that satisfy polynomial growth conditions

$$|g(t,x,u)| \leq C(1 + |x|^k + |u|^k)$$
$$|\psi(t,x)| \leq C(1 + |x|^k)$$

[1]Suppose the control process $\{u(t) : t \geq 0\}$ is defined on the same probability space $(\Omega, \mathcal{F}, \mathbb{P})$ which the Wiener process $\{w(t) : t \geq 0\}$ is also defined on. Then, $\{u(t) : t \geq 0\}$ is said to be *admissible* with respect to $\{w(t) : t \geq 0\}$, if there exists a filtration $\{\mathcal{F}_t : t \geq 0\}$ defined on $(\Omega, \mathcal{F}, \mathbb{P})$ such that $u(t)$ is $\mathcal{F}_t$-adapted and $w(t)$ is an $\mathcal{F}_t$-Wiener process. See [81] for the precise measure theoretic definitions.

for some constants $C \in \mathbb{R}, k \in \mathbb{N}$.

For the FCFH MDP, we a seek a control law that acts upon the system until a finite time is reached or until the state exits the space $\mathscr{O}$. Define $\mathscr{T} = [t_0, t_1)$ for fixed $t_0, t_1 \in \mathbb{R}_+$ with $t_0 < t_1 < \infty$. Furthermore, let $\mathscr{Q} = [t_0, t_1) \times \mathscr{O}$ denote the product space of times and states. Then, the *exit time* τ of the system is defined to be

$$\tau = \inf\{s : (s, x(s)) \notin \mathscr{Q}\}, \quad s > t.$$

For example, we have $\tau = t_1$, if for all times $t \leq s \leq t_1$ the state remains in the interior of the state space, i.e., $x(s) \in \mathscr{O}$. For any product state $(t, z) \in \mathscr{Q}$ and admissible process $u = \{u(t) : t \geq 0\}$, we can define a functional

$$c(t, z; u) = \mathbb{E}\left[\int_t^\tau g(s, x(s), u(s))ds + \psi(\tau, x(\tau))\right], \quad x(t) = z.$$

This functional describes the expected cost incurred by a system initialized with the initial condition (t, z) under a control u.

For the DCIH MDP, we consider only time-invariant dynamical systems where (4.1) is modified according to

$$dx(t) = B(x(t), u(t))dt + \mathcal{D}(x(t), u(t))dw(t), \quad x(0) = z, \text{ for } z \in \mathscr{O}.$$

In this case, the exit time must be modified because the system could evolve indefinitely. Therefore, we follow Fleming and Soner [44] and define the exit time τ as either the first time that the state $x(s)$ exits from $\mathscr{O}$, or we set $\tau = \infty$ if the state remains forever within $\mathscr{O}$, i.e., $x(s) \in \mathscr{O}$ for all $s \geq 0$. Within this formulation, we can still use a terminal cost ψ for the cases when $\tau < \infty$. In order to accommodate finite exit times, we use the indicator function $\chi_{\tau<\infty}$ that evaluates to one if the state exits $\mathscr{O}$ and to zero otherwise. The cost functional for the DCIH MDP is

$$c(z; u) = \mathbb{E}\left[\int_0^\tau e^{-\beta s} g(s, x(s), u(s))ds + \chi_{\tau<\infty} e^{-\beta\tau}\psi(\tau, x(\tau))\right], \quad x(0) = z,$$

where $\beta > 0$ is a discount factor. For this problem to be well-defined, we require that the cost until exit is bounded [44]

$$\mathbb{E}\left[\int_0^\tau \exp^{-\beta s} |g(s, x(s), u(s))| ds\right] \leq \infty.$$

4.1.3 Markovian policies

A *Markov control policy* is a mapping $\mu : \mathscr{T} \times \mathscr{X} \to \mathscr{U}$ that assigns a control input to each time and state. An admissible control is then obtained as a realization of the policy $u(t) = \mu(t, x(t))$.

For each of the problem formulations described above, we can correspondingly denote the cost functional associated with a specific Markov policy μ to be

$$c_\mu(t, z) = \mathbb{E}\left[\int_t^\tau g(s, x(s), \mu(s, x(s))ds + \psi(\tau, x(\tau))\right], \quad x(t) = z,$$

for the FCFH MDP, and

$$c_\mu(z) = \mathbb{E}\left[\int_0^\tau e^{-\beta s} g(s, x(s), \mu(s, x(s))ds + \chi_{\tau<\infty} e^{-\beta\tau}\psi(x(\tau))\right], \quad x(0) = z,$$

for the DCIH MDP. Let $y = (t, z)$ denote the product state of the FCFH MDP and $y = z$ refer to the state of the DCIH MDP. The goal of the stochastic control problem is to seek an *optimal* cost c_{μ^*} with the following property

$$c_{\mu^*}(y) = \inf_\mu c_\mu(y),$$

for all y subject to (4.1). Under certain conditions one can show that a Markov policy is at least as good as any other arbitrary $\mathcal{F}_t$-adapted policy; see for example Theorem 11.2.3 by Øksendal [91]. In this dissertation, we assume these conditions exist and only work with Markov control policies. Markov control policies allow us to avoid storing full state trajectories of the system when considering what action to apply. Since they only require knowledge of the current time and state, they are computationally efficient to use in practice.

4.1.4 Dynamic programming

In the dynamic programming problem, we seek an optimal value function $v(t, z)$ that is equal to the optimal cost functional

$$v(t, z) = \inf_{\mu} c_{\mu}(t, z) \text{ for all } z \in \mathscr{O}.$$

For continuous-time and continuous-space SOC the optimal value function is the solution of a Hamilton-Jacobi-Bellman (HJB) partial differential equation (PDE) [44]. The HJB PDE is defined using a function called a Hamiltonian. Let $\mathcal{S}_{+}^{d}$ denote the set of symmetric, nonnegative definite matrices. Let $\mathbf{A} \in \mathcal{S}_{+}^{d}$ and $\mathcal{A} = \mathcal{D}\mathcal{D}^{T}$, then the trace tr $\mathcal{A}\mathbf{A}$ is defined as

$$\text{tr } \mathcal{A}\mathbf{A} = \sum_{i,j}^{d} \mathcal{A}[i, j]\mathbf{A}[i, j].$$

Finally, for $(t, z) \in \mathscr{Q}$, $p \in \mathscr{X}$, $\mathbf{A} \in \mathcal{S}_{+}^{d}$, the *Hamiltonian* is defined as

$$\check{H}(t, z, p, \mathbf{A}) = \sup_{\bar{u} \in \mathscr{U}} \left[-B(t, z, \bar{u}) \cdot p - \frac{1}{2} \text{tr } \mathcal{A}(t, z, \bar{u})\mathbf{A} - g(t, z, \bar{u}) \right]$$

The dynamic programming equation, called the HJB PDE, for the FCFH MDP is then defined as

$$-\frac{\partial v}{\partial t} + \check{H}(t, z, \nabla v, D_x^2 v) = 0, \quad (t, z) \in \mathscr{Q},$$

where D_x^2 is the Hessian operator. Let

$$\partial^* \mathscr{Q} = ([t_0, t_1] \times \partial\mathscr{O}) \cup (\{t_1\} \times \mathscr{O}),$$

denote the boundary of the product state space. Then, the boundary conditions for the HJB PDE are given by

$$v(t, z) = \psi(t, z) \quad \text{for } (t, z) \in \partial^* \mathscr{Q}.$$

This PDE is of parabolic type. It is called uniformly parabolic if there exists a $c > 0$ such that

$$\sum_{i,j}^{d} \mathcal{A}[i,j](t,x,\bar{u})\xi_i\xi_j \geq c|\xi|^2, \tag{4.3}$$

for $\xi \in \mathbb{R}^d$, otherwise it is of degenerate parabolic type [44].

For the DCIH MDP, the Hamiltonian and corresponding HJB PDE are no longer time dependent. The HJB PDE is defined according to

$$\beta v + \check{H}(z, \nabla v, D_x^2 v) = 0, \quad z \in \mathscr{O},$$

with boundary conditions

$$v(z) = \psi(z), \quad z \in \partial\mathscr{O}.$$

If the diffusion satisfies condition (4.3) then the HJB PDE is a uniformly elliptic equation having a smooth and unique solution when $\mathscr{O}$ is bounded. Otherwise, the equation is of degenerate elliptic type [44].

4.2 Discretization-based solution algorithms

The Markov chain approximation (MCA) [81] and other discretization based methods, e.g., the method prescribed by Tsitsiklis [121], for solving the SOC problem rely on first discretizing the state space and dynamics described by (4.1) and then solving the resulting discrete-time and discrete-space MDP. The discrete MDP can be solved using standard techniques from dynamic programming such as value iteration (VI) or policy iteration (PI) or other approximate dynamic programming techniques [12, 13, 15, 81, 102].

In Section 4.2.1, we provide a brief overview of discrete MDPs. In Section 4.2.2 we describe the MCA framework for discretizing continuous SOC to a discrete MDP. Finally, in Section 4.2.3, we then describe three classical solution algorithms: value iteration, policy iteration, and multilevel methods.

4.2.1 Discrete-time and discrete-space Markov decision processes

The MDP resulting from the MCA with discretization level h is a tuple $\mathscr{M}^h = (\mathscr{X}^h, \mathscr{U}, p^h, g^h, \psi^h)$, where $\mathscr{X}^h$ is the set of discrete states, $p^h(\cdot,\cdot|\cdot) : \mathscr{X}^h \times \mathscr{X}^h \times \mathscr{U} \to [0,1]$ is function that denotes the transition probabilities satisfying $\sum_{z' \in \mathscr{X}^h} p^h(z, z'|\bar{u}) = 1$ for all $z \in \mathscr{X}^h$ and all $\bar{u} \in \mathscr{U}$, g^h is the stage cost of the discrete system, and ψ^h is the terminal cost of the discrete system.

The transition probabilities replace the drift and diffusion terms of the SDE as the description for the evolution of the state. For example, when the process is at state $z \in \mathscr{X}^h$ and action $\bar{u} \in \mathscr{U}$ is applied, the next state of the process becomes $z' \in \mathscr{X}^h$ with probability $p^h(z, z'|\bar{u})$. Define the state transitions to occur at discrete intervals $\mathscr{T}^h = \{t_0, t_1, t_2, \ldots t_k\}$.

In this discrete setting, Markov policies are now mappings $\mu : \mathscr{T}^h \times \mathscr{X}^h \to \mathscr{U}$ defined from a discrete state space rather than from the continuous space $\mathscr{X}$. Furthermore, the cost functional becomes a multidimensional array $c_\mu : \mathscr{T}^h \times \mathscr{X}^h \to \mathbb{R}$. The cost associated with a particular trajectory and policy, for a finite horizon, discrete time system, i.e., $t_0 = 0, t_1 = 1, t_2 = 2, \ldots$, is

$$c_\mu(t_0, z) = \mathbb{E}\left[\sum_{i=1}^{N} \gamma^i g^h\left(t_k, x\left(t_k\right), \mu\left(t_k, x\left(t_k\right)\right)\right) + \psi^h\left(t_N, x\left(t_N\right)\right)\right], \quad x(t_0) = z$$

for $z \in \mathscr{X}^h$, where $0 < \gamma < 1$ is the discount factor. The *Bellman equation*, a discrete analogue of the HJB PDE, describing the optimality of this discretized problem is

$$v^h(t_k, z) = \min_{\bar{u} \in \mathscr{U}} \left[g^h(t_k, z, \bar{u}) + \gamma \sum_{z' \in \mathscr{X}^h} p^h(z, z'|\bar{u}) v^h(t_{k+1}, z') \right],$$

where v^h is the optimal discretized value function. The solution of the Bellman equation satisfies

$$v^h(t, z) = \inf_\mu c_\mu(t, z),$$

and therefore also solves the discrete optimal stochastic control problem [13].

In this dissertation, we only consider the value function to be a function of the state and independent of time. In this case, we are required to assume that the stage costs are also independent of time, i.e., $g^h : \mathscr{X}^h \to \mathbb{R}$, and the Bellman equation may be modified according to

$$v^h(z) = \min_{\bar{u} \in \mathscr{U}} \left[g^h(z, \bar{u}) + \gamma \sum_{z' \in \mathscr{X}^h} p^h(z, z' | \bar{u}) v^h(z') \right]. \tag{4.4}$$

For those cases when the value function is time dependent, for example in finite horizon problems, one can always augment the state space with the time variable to create a new value function that is a function of the augmented (or product) state space.

4.2.2 Markov chain approximation method

The MCA method, developed by Kushner and co-workers [79–81], constructs a sequence of discrete MDPs such that the solution of the MDPs converge to the solution of the original continuous-time continuous-space problem.

Let $\{\mathscr{M}^{h_\ell} : \ell \in \mathbb{N}\}$ be a sequence of MDPs, where each $\mathscr{M}^{h_\ell} = (\mathscr{X}^{h_\ell}, \mathscr{U}, p^{h_\ell}, g^{h_\ell}, \psi^{h_\ell})$ is defined as before. Define $\partial\mathscr{X}^{h_\ell}$ as the subset of $\mathscr{X}^{h_\ell}$ that falls on the boundary of $\mathscr{X}$, i.e., $\partial\mathscr{X}^{h_\ell} = \partial\mathscr{X} \cap \mathscr{X}^{h_\ell}$. Let $\{\Delta t^\ell : \ell \in \mathbb{N}\}$, where $\Delta t^\ell : \mathscr{X}^{h_\ell} \to \mathbb{R}_+$, be a sequence *holding times*. Let $\{\xi_i^n : i \in \mathbb{N}\}$, where $\xi_i^n \in \mathscr{X}^{h_\ell}$, be a (random) sequence of states that describe the trajectory of $\mathscr{M}^{h_\ell}$. We use holding times as interpolation intervals to generate a continuous-time trajectory from this discrete trajectory as follows. With a slight abuse of notation, let $\xi^n : \mathbb{R}_{\geq 0} \to \mathscr{X}^{h_\ell}$ denote the continuous-time function defined as follows: $\xi^n(\tau) = \xi_i^n$ for all $\tau \in [t_i^\ell, t_{i+1}^\ell)$, where $t_i^\ell = \sum_{k=0}^{i-1} \Delta t^\ell(\xi_k)$. Let $\{u_i^\ell : i \in \mathbb{N}\}$, where $u_i^\ell \in \mathscr{U}$, be a sequence of control inputs defined for all $\ell \in \mathbb{N}$. Then, we define the continuous time interpolation of $\{u_i^\ell : i \in \mathbb{N}\}$ as $u^\ell(\tau) = u_i^\ell$ for all $\tau \in [t_i^\ell, t_{i+1}^\ell)$. An illustration of this interpolation is provided in Figure 4-1.

The following result by Kushner and co-workers characterizes the conditions under which the trajectories and value functions of the discrete MDPs converge to those of

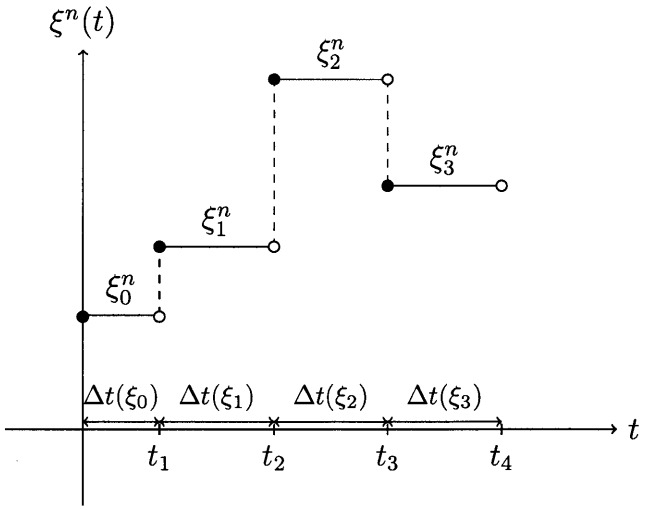


Figure 4-1: An illustration of a continuous-time interpolation of a discrete process arising from the Markov chain approximation.

the original continuous-time continuous-space stochastic system.

Theorem 6 (See Theorem 10.4.1 by Kushner and Dupuis [81]). *Suppose the sequence* $\{\mathscr{M}^{h_\ell} : \ell \in \mathbb{N}\}$ *of MDPs and the sequence* $\{\Delta t^\ell : \ell \in \mathbb{N}\}$ *holding times satisfy the following conditions: For any sequence of inputs* $\{u_i^\ell : i \in \mathbb{N}\}$ *and the resulting sequence of trajectories* $\{\xi_i^\ell : i \in \mathbb{N}\}$ *if*

$$\lim_{\ell \to \infty} \Delta t^\ell(z) = 0, \text{ for all } z \in \mathscr{X},$$

and

$$\lim_{\ell \to \infty} \frac{\mathbb{E}[\xi_{i+1}^\ell - \xi_i^\ell \,|\, \xi_i^\ell = z, u_i^\ell = \bar{u}]}{\Delta t^\ell(z)} = B(z, \bar{u}),$$
$$\lim_{\ell \to \infty} \frac{\mathrm{Cov}[\xi_{i+1}^\ell - \xi_i^\ell \,|\, \xi_i^\ell = z, u_i^\ell = \bar{u}]}{\Delta t^\ell(z)} = \mathcal{D}(z, \bar{u}),$$

for all $z \in \mathscr{X}$ *and* $\bar{u} \in \mathscr{U}$. *Then, the sequence* $\{(\xi^\ell, u^\ell) : \ell \in \mathbb{N}\}$ *of interpolations converges in distribution to* (x, u) *that solves the integral equation with differential form given by* (4.1). *Let* v^{h_ℓ} *denote the optimal value function for the MDP* $\mathscr{M}^{h_\ell}$. *Then, for all* $z \in \mathscr{X}^{h_\ell}$,

$$\lim_{\ell \to \infty} |v^{h_\ell}(z) - v(z)| = 0.$$

The conditions of this theorem are often called the *local consistency conditions.* Roughly speaking, the theorem states that the trajectories of the discrete MDPs will converge to the trajectories of the original continuous-time stochastic dynamical system if the local consistency conditions are satisfied. Furthermore, in that case, the value function of the discrete MDPs also converge to that of the original stochastic optimal control problem. A discretization that satisfies the local consistency conditions is called a *consistent discretization.* Once a consistent discretization is obtained, standard dynamic programming algorithms such as value iteration or policy iteration [12] can be used for its solution.

Discretization procedures

In this section, we provide a general discretization framework described by Kushner and Dupuis [81] along with a specific example. For the rest of the dissertation, we will drop the subscript ℓ from h_ℓ, and simply refer to the discretization level h with $h > 0$.

First, let $\mathscr{X}^h \subset \mathscr{X}$ denote a discrete set of states. For each state $z \in \mathscr{X}^h$ define a finite set of vectors $M(z) = \{\boldsymbol{v}_{i,z} : i < m(z)\}$, where $\boldsymbol{v}_{i,z} \in \mathbb{R}^d$ and $m(z) : \mathscr{X}^h \to \mathbb{N}$ is uniformly bounded. These vectors denote directions from a state z to a neighboring set of states $\{y : y = z + h\boldsymbol{v}_{i,z}, i \leq m(z)\} \subset \mathscr{X}^h$. A valid discretization is described by the functions $q_i^1(z) : \mathscr{X}^h \to \mathbb{R}$ and $q_i^0(z, \bar{u}) : \mathscr{X}^h \times \mathscr{U} \to \mathbb{R}$ that satisfy

$$
\begin{aligned}
B(z, \bar{u}) &= \sum_{\boldsymbol{v}_{i,z} \in M(z)} q_i^0(z, \bar{u}) \boldsymbol{v}_{i,z}, \text{ for all } \bar{u}, \\
\mathcal{D}(z) &= \sum_{\boldsymbol{v}_{i,z} \in M(z)} q_i^1(z) \boldsymbol{v}_{i,z} \boldsymbol{v}_{i,z}', \\
\sum_{\boldsymbol{v}_{i,z} \in M(z)} q_i^1(z) \boldsymbol{v}_{i,z} &= 0, \\
h q_i^0(z, \bar{u}) + q_i^1(z) &\geq 0, \\
q_i^1(z) &> 0,
\end{aligned}
$$

where the third condition is used to guarantee that each q_i^1 only contributes to the

variance of the chain and not the mean, and the fourth and fifth conditions will guarantee non-negative transition probabilities. After finding q_i^1 and q_i^0 that satisfy these conditions, the discrete MDP is described by the normalizing constant

$$q^h(z,\bar{u}) = \sum_{\boldsymbol{v}_{i,z} \in M(z)} \left[h q_i^0(z,\bar{u}) + q_i^1(z) \right]$$

that satisfies $q^h(z,\bar{u}) > 0$, the interpolation interval

$$\Delta t^h(z,\bar{u}) = \frac{h^2}{q^h(z,\bar{u})}$$

that must tend to zero as $h \to 0$, the transition probabilities

$$p^h(z, z + h\boldsymbol{v}_{i,z}|\bar{u}) = \frac{h q_i^0(z,\bar{u}) + q_i^1(z)}{q^h(z,\bar{u})},$$

the stage costs

$$g^h(z,\bar{u}) = \frac{\Delta t^h(z,\bar{u})}{q^h(z,\bar{u})} g(z,\bar{u}),$$

and, finally, the discount factor

$$\gamma = \exp(-\beta \Delta t^h).$$

It is easy to verify that these conditions satisfy local consistency, and we refer the reader to the work of Kushner and Dupuis [81] for more details.

Example: Upwind differencing. One valid discretization of the MCA is obtained using upwind differencing. This procedure tries to "push" the current state of the system in the direction of the drift dynamics on average, and we use it for all of the numerical examples in Section 4.5.

Consider the sample discretization of a two dimensional state space provided in Figure 4-2. The state is discretized with a spacing of h_1 in the first dimension and h_2 in the second dimension. In this example, the directions $\boldsymbol{v}_{i,z}$ are independent of z, and thus we drop the subscript z and refer simply to $\boldsymbol{v}_i$. Let the discretization level

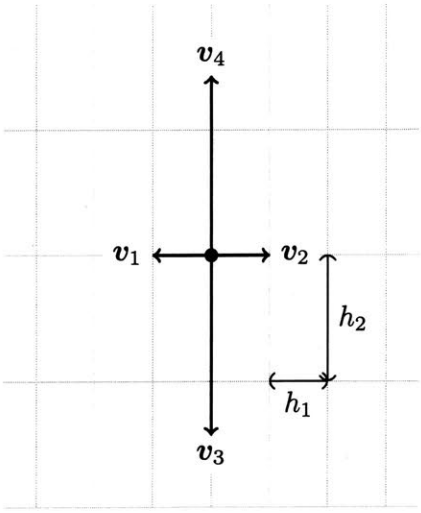


Figure 4-2: Sample discretization of a two-dimensional state space.

be defined according to $h = \min(h_1, h_2)$ so that transition directions $\boldsymbol{v}_i$ are defined according to

$$\boldsymbol{v}_1 = -\frac{h_1}{h}\boldsymbol{e}_1, \quad \boldsymbol{v}_2 = \frac{h_1}{h}\boldsymbol{e}_1, \quad \boldsymbol{v}_3 = -\frac{h_2}{h}\boldsymbol{e}_2, \quad \boldsymbol{v}_4 = \frac{h_2}{h}\boldsymbol{e}_2.$$

The directions $\boldsymbol{v}_i$ do not need to be the same length as the discretization size of the state space. In particular, their length is the ratio between the grid size in the particular direction and the finest discretization. This characteristic will ensure local consistency even for cases when the discretization is different in each direction by causing the transition probabilities to take the relative discretizations of each dimension into account.

In the two dimensional state space, the output of the drift can be indexed according to $B[1]$ and $B[2]$. Furthermore, assume that the diffusion is defined so that $\mathcal{A}(z) = \mathcal{D}(z)\mathcal{D}^T(z)$ is a diagonal matrix—$\mathcal{A}(z) = \text{diag}([\mathcal{A}[1,1](z), \mathcal{A}[2,2](z)])$. Let $B[i](z,\bar{u})^+ = \max(0, B[i](z,\bar{u}))$, and finally let $B[i](z,\bar{u})^- = \max(0, -B[i](z,\bar{u}))$. Then, a locally consistent discretization is one defined by

$$q_1^0(z,\bar{u}) = \frac{h}{h_1} B[1](z,\bar{u})^-, \quad q_0^1 = \left(\frac{h}{h_1}\right)^2 \frac{\mathcal{A}[1,1](z)^2}{2},$$
$$q_2^0(z,\bar{u}) = \frac{h}{h_1} B[1](z,\bar{u})^+, \quad q_1^1 = \left(\frac{h}{h_1}\right)^2 \frac{\mathcal{A}[1,1](z)^2}{2},$$
$$q_3^0(z,\bar{u}) = \frac{h}{h_2} B[2](z,\bar{u})^-, \quad q_2^1 = \left(\frac{h}{h_2}\right)^2 \frac{\mathcal{A}[2,2](z)^2}{2},$$
$$q_4^0(z,\bar{u}) = \frac{h}{h_2} B[2](z,\bar{u})^+, \quad q_3^1 = \left(\frac{h}{h_2}\right)^2 \frac{\mathcal{A}[2,2](z)^2}{2},$$

Suppose the drift is such that $B[1](z,\bar{u}) > 0$ and $B[2](z,\bar{u}) < 0$ then we can verify

$$\sum_{i \in M(z)} q_i^0(z,\bar{u}) \boldsymbol{v}_i = \underbrace{\frac{h}{h_1} B[1](z,\bar{u})^+}_{q_2^0(z,\bar{u})} \underbrace{\frac{h_1}{h} \boldsymbol{e}_1}_{\boldsymbol{v}_2} + \underbrace{\frac{h}{h_2} B[2](z,\bar{u})^-}_{q_3^0(z,\bar{u})} \underbrace{\left(-\frac{h_2}{h} \boldsymbol{e}_2\right)}_{\boldsymbol{v}_3}$$
$$= \begin{bmatrix} B[1](z,\bar{u}) \\ B[2](z,\bar{u}) \end{bmatrix} = B(z,\bar{u})$$

All the other conditions can be similarly verified.

We can also tally the computational cost of this upwind differencing procedure. The computation of the transition probabilities, for some state z and control $\bar{u}$, requires the evaluation of the drift and diffusion. Suppose that this evaluation requires n_{op} operations. Assembling each $q_i^0(z,\bar{u})$ and q_i^1 requires two operations: multiplication and division. Since there are $2d$ neighbors for each z, the evaluation of all of them requires $4d$ operations.

Next, the computation of the normalization q^h involves summing all of the q_i^0 and q_i^1, a procedure requiring $4d$ operations. Computing the interpolation interval requires a single division, computing the discrete stage cost requires a division and multiplication, and computing the discount factor requires exponentiation. Together, these operations mean that the computational complexity of discretizing the SOC for some state z and control $\bar{u}$ using upwind differencing is linear with dimension

$$\mathcal{O}(n_{\text{op}} + d). \tag{4.5}$$

Boundary conditions

The discretization methods described in the previous section apply to the interior nodes of the state space. To numerically solve optimal stochastic control problems, however, one typically needs to restrict the state space to a particular region. Furthermore, to utilize low-rank tensor methods in high dimensions, we design $\mathscr{O}$ to be a hypercube. Because of the state truncation, we assign boundary conditions for the discrete MDP. Three boundary conditions are commonly used: periodic, absorbing, and reflecting boundary conditions.

A periodic boundary condition maps one side of the domain to the other. For example consider $\mathscr{O} = (-1, 1)^2$. Then, if we define a periodic boundary condition for the first dimension, we mean that $z = (-1, \cdot)$ and $z' = (1, \cdot)$ are equivalent states.

Absorbing boundary conditions dictate that if the Markov process enters $\partial\mathscr{O}$ at the exit time τ, then the process terminates and terminal costs are incurred.

Reflecting boundary conditions are often imposed when one does not want to end the process at the boundary and periodic boundaries are not appropriate. In these cases, the stochastic process is modeled with a jump diffusion. The jump diffusion term is responsible for keeping the process within $\mathscr{O}$. In our case, we will assume that the jump diffusion term instantaneously "reflects" the process using an orthogonal projection back into the state space $\mathscr{O}$. For example, if the system state is $z \in \mathscr{O}$ and the Markov process transitions to $z' = z + h\boldsymbol{e}_k$ such that $z' \in \partial\mathscr{O}$, then the system immediately returns to the state z. Therefore, we can eliminate z' from the discretized state space and adjust the self transition probability to be

$$p^h(z, z|\bar{u}) \leftarrow p^h(z, z|\bar{u}) + p^h(z, z'|\bar{u}).$$

In other words, the probability of self transitioning is increased by the probability of transitioning to the boundary.

4.2.3 Value iteration, policy iteration, and multilevel methods

We now describe algorithms for solving the DCIH MDP given by (4.4). In particular, we describe the value iteration (VI) algorithm and the policy iteration (PI) algorithm. Then, we describe a multi-level algorithm that is able to use coarse-grid solutions to generate solutions of fine-grid problems. FT-based versions of these algorithms will then be described in Section 4.3.

DP equations

We now describe some notation and basic properties of the DP equations that are used within VI and PI. Let $\mathscr{R}^h$ be the set of real-valued functions $w^h : \mathscr{X}^h \to \mathbb{R}$. Let the functional $H^h : \mathscr{X}^h \times \mathscr{U} \times \mathscr{R}^h$ be defined as

$$H^h(z, \bar{u}, w^h) := g^h(z, \bar{u}) + \gamma \sum_{z' \in \mathscr{X}^h} p^h(z, z' | \bar{u}) w^h(z'). \tag{4.6}$$

Then, for a given policy μ we define operator $T^h_\mu : \mathscr{R}^h \to \mathscr{R}^h$ as

$$T^h_\mu(w^h)(z) := H^h(z, \mu(z), w^h), \quad \forall z \in \mathscr{X}^h, w^h \in \mathscr{R}^h$$

We denote another mapping $T^h : \mathscr{R}^h \to \mathscr{R}^h$ that corresponds to the Bellman equation (4.4) describing the optimal value function according to

$$T^h(w^h)(z) := \min_{\mu(z)} H^h(z, \bar{u}, w^h), \quad \forall z \in \mathscr{X}^h, w^h \in \mathscr{R}^h.$$

Using these operators we can denote two important fixed-point equations. The first describes the value function w^h that corresponds to a fixed policy μ

$$w^h = T^h_\mu(w^h). \tag{4.7}$$

The second equation describes the optimal value function v^h

$$v^h = T^h(v^h). \tag{4.8}$$

Assumptions for algorithm convergence

Three assumptions are necessary to guarantee existence and uniqueness of the above DP equations, and to validate their associated solution algorithms.

Assumption 1 (Assumption A1.1 by Kushner and Dupuis [81]). *The functions $p^h(z, z'|\bar{u})$ and $g^h(z, \bar{u})$ are continuous with respect to $\bar{u}$ for all $z, z' \in \mathscr{X}^h$.*

The second assumption involves contraction, and we provide a definition of contraction below.

Definition 19 (Contraction). *Let $\mathscr{Y}$ be a normed vector space with the norm $\|\cdot\|$. A function $f : \mathscr{Y} \to \mathscr{Y}$ is a contraction mapping if for some $\gamma \in (0, 1)$ we have*

$$\|f(y) - f(y')\| \leq \gamma \|y - y'\|, \quad \forall y, y' \in \mathscr{Y}.$$

The second assumption is now provided.

Assumption 2 (Assumption A1.2 by Kushner and Dupuis [81]). *(i) There is at least one admissible feedback policy μ such that T_μ^h is a contraction, and the infima of the costs over all admissible policies is bounded from below. (ii) T_μ^h is a contraction for any feedback policy for which the associated cost is bounded.*

The third assumption involves the repeated application T_μ^h.

Assumption 3 (Assumption A1.3 by Kushner and Dupuis [81]). *Let $\mathbf{P}_\mu = \{p^h(z, z'|\mu(z)) : z, z' \in \mathscr{X}^h\}$ be the matrix formed by the transition probabilities of the discrete-state MDP for a fixed policy μ. If the value functions associated with the use of policies $\mu_1, \ldots, \mu_n, \ldots$ in sequence, is bounded, then*

$$\lim_{n \to \infty} \mathbf{P}_{\mu_1} \mathbf{P}_{\mu_2} \cdots \mathbf{P}_{\mu_n} = 0$$

Value iteration algorithm

Value iteration is a fixed-point (FP) iteration for obtaining the optimal value function v^h. It works by starting with an initial guess $v_0^h \in \mathscr{R}^h$ and defining a sequence $\{v_k^h\}$

of value functions through the iteration $v_{k+1}^h = T^h(v_k^h)$. Theorem 7 guarantees the convergence of this algorithm.

Theorem 7 (Jacobi iteration, Theorem 6.2.2 by Kushner and Dupuis [81]). *Let μ be an admissible policy such that T_μ^h is a contraction. Then for any initial vector $w_0^h \in \mathscr{R}^h$, the sequence w_k^h defined by*

$$w_{k+1}^h = T_\mu^h(w_k^h) \tag{4.9}$$

converges to w^h, the unique solution to (4.7). *Assume Assumptions 1, 2, and 3. Then for any vector $v_0^h \in \mathscr{R}^h$, the sequence recursively defined by*

$$v_{k+1}^h = T^h(v_k^h) \tag{4.10}$$

converges to the optimal value function v^h, the unique solution to (4.8).

Indeed (4.10) is the FP iteration that is the value iteration algorithm. In Section 4.3.2, we will describe how to exploit low-rank structure within this algorithm.

Policy iteration algorithm

Policy iteration (PI) is another way to solve the MDP. Roughly, it is analogous to a gradient descent method, and our experiments indicate that it generally converges faster than VI. The basic idea is to start with a Markov policy μ_0 and to generate a sequence of policies $\{\mu_k\}$, according to

$$\mu_k = \arg\min_\mu \left[T_\mu^h(w_{k-1}^h)\right], \tag{4.11}$$

and associated value functions $\{w_k^h\}$, that satisfy (4.7), i.e.,

$$w_k^h = T_{\mu_k}^h(w_k^h). \tag{4.12}$$

Theorem 8 provides the conditions under which this iteration converges.

Theorem 8 (Policy iteration, Theorem 6.2.1 by Kushner and Dupuis [81]). *Assume Assumptions 1 and 2. Then there is a unique solution to* (4.8)*, and it is the infimum of the value functions over all time independent feedback policies. Let* μ_0 *be an admissible feedback policy such that the corresponding value function* w_0^h *is bounded. For* $k \geq 1$*, define the sequence of feedback policies* μ_k *and costs* w_k^h *recursively by* (4.11) *and* (4.12)*. Then* $w_k^h \to v^h$*. Under the additional condition given by Assumption 3,* v^h *is the infimum of the value functions over all admissible policies.*

Note that policy iteration requires the solution of a linear system (4.12). Furthermore, Theorem 7 states that since T_μ^h is a contraction mapping that a FP iteration can also be used to solve this system. This property leads to a modification of the policy iteration algorithm called optimistic policy iteration [13]. *Optimistic policy iteration* substitutes n_{fp} steps of FP iterations for solving (4.12) to create a more computationally efficient algorithm. The assumptions necessary for convergence of optimistic PI are the same as those for PI and VI, and we refer the reader to the work of Bertsekas [13] for more details. In Section 4.3.3, we will describe a low-rank version of optimistic PI that uses the FT decomposition.

Multi-level algorithms

Multigrid [19, 120] techniques have been successful at obtaining solutions to many problems by exploiting multiscale structure of the problem. For example, they are able to effectively leverage solutions of linear systems at coarse discretization levels for solving finely discretized systems.

We describe how to apply these ideas within DP for two purposes: the initialization of fine-grid solutions with coarse-grid solutions and for the solution of the linear system (4.12) within policy iteration. Since our experiments indicate that fine-grid problems typically require more iterations to converge, initialization with a coarse-grid solution offers the opportunity for dramatic speedups. Furthermore, since we expect the solution to converge to the continuous solution as the grid is refined, we expect the number of iterations required for convergence to decrease as the grid is refined.

The simplest multi-level algorithm is the one-way discretization algorithm that sequentially refines coarse-grid solutions of (4.8) by searching for solutions on a grid starting from an initial guess obtained from the solution of a coarser problem. This procedure was analyzed in detail for shortest path or MDP style problems by Chow and Tsitsiklis in [28]. The pseudocode for this algorithm provided in Algorithm 8. In Algorithm 8, a set of κ discretization levels $\{h_1, h_2, \ldots, h_\kappa\}$ such that for $j > i$ we have $h_j > h_i$, are specified. Furthermore, the operator I_{2h}^h interpolates the solution of the h_{k+1} grid onto the h_k grid. Then a sequence of problems, starting with the coarsest, are solved until the fine-grid solution is obtained.

Algorithm 8 One-way multigrid [28]

Require: Set of discretization levels $\{h_1, h_2, \ldots, h_\kappa\}$; Initial cost function $v_0^{h_\kappa}$

1: $v^{h_\kappa} \leftarrow$ Solve (4.8) starting from $v_0^{h_\kappa}$
2: **for** $k = \kappa - 1 \ldots 1$ **do**
3: $\quad v_0^{h_k} = I_{h_{k+1}}^{h_k} v^{h_{k+1}}$
4: $\quad v^{h_k} \leftarrow$ Solve (4.8) starting from $v_0^{h_k}$
5: **end for**

Multigrid techniques can also be used for solving the linear system (4.7) within the context of policy iteration. Recall that for a fixed policy μ, this system can be equivalently written using a linear operator Π_μ^h defined according to

$$\Pi_\mu^h(w^h)(z) \equiv w^h(z) - \gamma \sum_{z' \in \mathscr{X}^h} p^h\left(z, z' | \mu(z)\right) w^h(z'), \quad \forall z \in \mathscr{X}^h.$$

Therefore, for a fixed policy μ the corresponding value function satisfies

$$\Pi_\mu^h(w^h) = g^h. \tag{4.13}$$

Typically, we do not expect to satisfy this equation exactly, rather we will have an approximation $\widehat{w}^h$ that yields a non-zero residual

$$r^h = g^h - \Pi_\mu^h(\widehat{w}^h). \tag{4.14}$$

In addition to the residual, we can define the difference between the approximation

and the true minimum as $\Delta w^h = w^h - \widehat{w}^h$. Since, Π_μ^h is a linear operator, we can replace $\widehat{w}^h$ in (4.14) to obtain

$$\begin{aligned} r^h &= g^h - \Pi_\mu^h(w^h - \Delta w^h) \\ &= g^h - \Pi_\mu^h(w^h) + \Pi_\mu^h(\Delta w^h) \\ &= \Pi_\mu^h(\Delta w^h) \end{aligned}$$

Thus, if solve for Δw^h, then we can update $\widehat{w}^h$ to obtain the solution

$$w^h = \Delta w^h + \widehat{w}^h.$$

In order for multigrid to be a successful strategy, we typically assume that the residual r^h is "smooth," and therefore we can potentially solve for Δw^h on a coarser grid. The coarse grid residual is

$$r^{2h} = \Pi_\mu^{2h}(\Delta w^{2h})$$

where we now choose the residual to be the restriction, denoted by operator I_h^{2h}, of the fine-grid residual

$$r^{2h} = I_h^{2h} r^h.$$

Combining these two equations we obtain an equation for Δw^{2h}

$$\Pi_\mu^{2h}(\Delta w^{2h}) = I_h^{2h} r^h \tag{4.15}$$

Note that the relationship between the linear operators T_μ^{2h} and Π_μ^{2h} displayed by (4.7) and (4.13) lead to an equivalent equation for the error given by

$$\Delta w^{2h} = T_\mu^{2h}(\Delta w^{2h}; r^{2h}), \tag{4.16}$$

where we specifically denote that the stage cost is replaced by r^{2h}. Since T_μ^{2h} is a contraction mapping we can use the FP iteration (4.9) to solve this equation.

After solving the system (4.15) above we can obtain the correction at the fine grid

$$\widehat{w}^h \leftarrow \widehat{w}^h + I_{2h}^h \Delta w^{2h} \tag{4.17}$$

In order, to obtain smooth out the high frequency components of the residual r^h one must perform "smoothing" iteration instead of the typical iteration T_μ^h. These iterations are typically Gauss-Seidel relaxations or weighted Jacobi iterations. Suppose that we start with $\widehat{w}_k^h$, then using the weighted Jacobi iteration we obtain an update $\widehat{w}_{k+1}^h$ through the following two equations

$$\begin{aligned}\widetilde{w}^h &= T_\mu^h(\widehat{w}_k^h, g^h),\\ \widehat{w}_{k+1}^h &= \omega\widetilde{w}^h + (1-\omega)\widehat{w}_k^h,\end{aligned}$$

for $\omega > 1$. To shorten notation, we will denote these equations by the operator $T_{c,\mu}^h$ such that

$$\widehat{w}_{k+1}^h = T_{c,\mu}^h(\widehat{w}_k^h, g^h).$$

We have chosen to use the weighted Jacobi iteration since it can be performed by treating the linear operator T_μ^h as a black-box FP iteration, i.e., the algorithm takes as input a value function and outputs another value function. Thus, we can still wrap the low-rank approximation scheme around this operator. A relaxed Gauss-Seidel relaxation would require sequentially updating elements of $\widehat{w}_k^h$, and then using these updated elements for other element updates. For more details on the reasons for these smoothing iterations within multigrid, we refer to [19,81,120].

Combining all of these notions we can design many multigrid methods. We will demonstrate a low-rank version of two-level V-grid in Algorithm 11 in the next section.

4.3 Low-rank dynamic programming algorithms

In this section, we describe how to leverage the compressed continuous computation framework described in Chapters 2 and 3 to solve the dynamic programming equations

arising from SOC. Specifically, we describe how to represent value functions in FT format and how to perform FT-based versions of VI, PI, and multigrid.

4.3.1 FT representation of value functions

The MCA method is used to approximate a continuous-space stochastic control problem with a discrete state Markov decision process. In this framework, the continuous value functions w are approximated by their discrete counterparts w^h. To leverage low-rank decompositions, we focus our attention on situations where the discrete value functions represent the cost of discrete MDPs defined using a tensor-product discretization of the state space. Therefore, w^h can be interpreted as a d-way array. In order to combat the curse of dimensionality associated with storing and computing w^h we propose to use continuous tensor decompositions.

Let $\mathscr{X} = \mathscr{X}_1 \times \mathscr{X}_2 \times \cdots \times \mathscr{X}_d$ denote a tensor-product state space. The space of the ith state variable is defined as $\mathscr{X}_i = [a_i, b_i]$, for $a_i, b_i \in \mathbb{R}$ that have the property $a_i < b_i$. A *tensor-product discretization* $\mathscr{X}^h$ of $\mathscr{X}$ involves discretizing each dimension into n nodes to form $\mathscr{X}_i^h \subset \mathscr{X}_i$ where

$$\mathscr{X}_i^h = \{z_1^{(i)}, z_2^{(i)}, \ldots z_n^{(i)}\}, \quad \text{where } z_k^{(i)} \in \mathscr{X}_i \text{ for all } k.$$

A discretized value function w^h can therefore be viewed as a vector with n^d elements.

Practically, the MCA method guarantees that the solution to the discrete MDP approximates the solution to the SOC, i.e., $v^h \approx v$, for small enough discretizations. This approximation, however, is ill-defined since v^h is a multidimensional array and v is a multivariate function. Furthermore, a continuous control law requires the ability to determine the optimal control for a system when it is in a state that is not included in the discretization. Therefore, it is necessary to use the discrete value function $w^h \in \mathscr{R}^h$ to develop a value function that is a mapping from the continuous space $\mathscr{X}$ to the reals.

To this end, we slightly abuse notation and interpret w^h as both a value function of the discrete MDP and as an *approximation* to value function of the continuous

system. Furthermore, when generating this continuous space approximation, we are restricted to evaluations located only within the tensor-product discretization. In this sense, we can think of w^h both as an array $w^h : \mathscr{X}^h \to \mathbb{R}$ in the sense that it has elements

$$w^h[i_1, \ldots, i_d] \approx w(z_1^{(i_1)}, \ldots, z_d^{(i_d)}),$$

and simultaneously as a function $w^h : \mathscr{X} \to \mathbb{R}$ where

$$w^h(z_1^{(i_1)}, \ldots, z_d^{(i_d)}) \approx w(z_1^{(i_1)}, \ldots, z_d^{(i_d)}).$$

It is infeasible to store the value function on a grid with n^d elements due to the curse of dimensionality. We employ the FT decomposition to develop a low-rank representation of the value function.

Recall that the FT representation of a function f is given by (3.4) and defined by the set of FT cores $\{\mathcal{F}_i\}$ for $i = 1, \ldots, d$. Each of these cores is a matrix-valued function $\mathcal{F}_i : \mathscr{X}_i \to \mathbb{R}^{r_{i-1} \times r_i}$ that can be visualized as an two-dimensional array of scalar-valued univariate functions:

$$\mathcal{F}_i(x_i) = \begin{bmatrix} f_{1,1}^{(i)}(x_i) & \cdots & f_{1,r_i}^{(i)}(x_i) \\ \vdots & \ddots & \vdots \\ f_{r_{i-1},r_i}^{(i)}(x_i) & \cdots & f_{r_{i-1},r_i}^{(i)}(x_i) \end{bmatrix}.$$

Using this matrix-valued function representation for the cores, the evaluation of a function in the FT format can be expressed as $f(x_1, \ldots, x_d) = \mathcal{F}_1(x_1) \ldots \mathcal{F}_d(x_d)$.

Since we effectively can only compute w^h through evaluations at uniformly discretized tensor-product grids, we employ a *nodal* representation of each scalar-valued univariate function

$$f_{\alpha_1,\alpha_2}^{(i,h)}(x_i) = \sum_{\ell=1}^{n} a_{\alpha_1,\alpha_2,\ell}^{(i,h)} \phi_\ell^h(x_i), \tag{4.18}$$

where $a_{\alpha_1,\alpha_2,\ell}^{(i,h)}$ are the coefficients of the expansion, and the basis functions $\phi_\ell^h : \mathscr{X}_i \to$

$\mathbb{R}$ are hat functions:

$$\phi_{\ell,h}(x_i) = \begin{cases} 0 & \text{if } x_i < z_{\ell-1}^{(i)} \text{ or } x_i > z_{\ell+1}^{(i)} \\ 1 & \text{if } x_i = z_\ell^{(i)} \\ \frac{x_i - z_{\ell-1}^{(i)}}{z_\ell^{(i)} - z_{\ell-1}^{(i)}} & \text{if } z_{\ell-1}^{(i)} \leq x_i < z_\ell^{(i)} \\ \frac{z_{\ell+1}^{(i)} - x_i}{z_{\ell+1}^{(i)} - z_\ell^{(i)}} & \text{if } z_\ell^{(i)} < x_i \leq z_{\ell+1}^{(i)} \end{cases}.$$

Note that these basis functions yield a *linear element* interpolation[2] of the function when evaluating it for a state not contained within $\mathscr{X}^h$.

We denote the FT cores of the value functions for discretization level h as $\mathcal{F}_i^h$. Finally, evaluating w^h anywhere within $\mathscr{X}$ requires evaluating a sequence of matrix-vector products

$$w^h(x_1, x_2, \ldots, x_d) = \mathcal{F}_1^h(x_1)\mathcal{F}_2^h(x_2)\ldots\mathcal{F}_d^h(x_d), \quad \text{for all } x_i \in \mathscr{X}_i.$$

4.3.2 FT-based value iteration

In prior work [56], we introduced a version of VI where each update (4.10) was performed using a low-rank tensor interpolation scheme that selectively chose to update only a small fraction of all the states. Here, we follow a similar strategy, except we use the continuous space approximation algorithm described in Algorithm 4, and denoted by `ft-rankadapt`, to accommodate our continuous space approximation. By seeking a low-rank representation of the value function, we are able to avoid visiting every state in $\mathscr{X}^h$ and achieve significant computational savings. The pseudocode for low-rank VI is provided by Algorithm 9. In this algorithm, v_k^h denotes the value function approximation during the kth iteration.

The update step 4 treats the VI update as a black box function into which one feeds a state and obtains an updated cost. After k steps of FT-based VI we can

[2]If we would have chosen a piecewise constant reconstruction, then for all intents and purposes the FT would be equivalent to the TT. Indeed we have previously performed such an approximation for the value function [56].

Algorithm 9 FT-based Value Iteration (FTVI) (Modified from [56])

Require: Termination criterion $\delta_{\max}$; `ft-rankadapt` cross approximation and rounding tolerances $\epsilon = (\delta_{\text{cross}}, \epsilon_{\text{round}})$; Initial cost function in FT format v_0^h; Convergence tolerance $\delta_{\max}$

Ensure: Residual $\delta = \|v_k^h - v_{k-1}^h\|^2 < \delta_{\max}$

1: $\delta = \delta_{\max} + 1$.
2: $k = 0$
3: **while** $\delta > \delta_{\max}$ **do**
4: $\quad v_{k+1}^h = \texttt{ft-rankadapt}\left(T^h(v_k^h), \epsilon\right)$
5: $\quad k \leftarrow k + 1$
6: $\quad \delta = \|v_k^h - v_{k-1}^h\|^2$
7: **end while**

obtain a policy as the minimizer of $T^h(v_k^h)(z)$, for any state $z \in \mathscr{X}$.

4.3.3 FT-based policy iteration

As part of PI, for each μ_k, one needs to solve (4.12) for the corresponding value function w_k^h. Recall that this system has an equivalent number of unknowns as states in $\mathscr{X}^h$, so the number of unknowns scales exponentially with dimension for tensor-product discretizations. To efficiently solve this system in high dimensions, we seek *low-rank* solutions. A wide variety of low-rank linear system solvers have recently been developed that can potentially be leveraged for this task [37, 97].

We focus on optimistic policy iteration, where we utilize the contractive property of $T_{\mu_k}^h$ to solve (4.12) approximately using n_{fp} FP iterations. We leverage the low-rank nature of each intermediate value w_k^h by interpolating a new value function for each of these iterations in the same manner that we did for VI. Notice that this iteration (4.9) is much less expensive than the value iteration (4.10) because it does not involve any minimization.

The pseudocode for the FT-based optimistic policy iteration is provided in Algorithm 10. Note that in Line 4, we represent a policy μ_k *implicitly* through a value function. We make this choice, instead of developing a low-rank representation of μ_k, because the policies are generally not low-rank, in our experience! Indeed, discontinuities can arise due to regions of uncontrollability, and these discontinuities increase

Algorithm 10 FT-based Optimistic Policy Iteration (FTPI)

Require: Termination criterion $\delta_{\max}$; `ft-rankadapt` cross approximation and rounding tolerances $\epsilon = (\delta_{\text{cross}}, \epsilon_{\text{round}})$; Initial value function in FT format w_0^h; Number of FP sub-iterations iterations n_{fp}

1: $\delta = \delta_{\max} + 1$.
2: $k = 1$
3: **while** $\delta > \delta_{\max}$ **do**
4: $\quad \mu_k = \texttt{ImplicitPolicy}\left(\arg\min_{\mu}\left[T_{\mu}^{h}(w_{k-1}^{h})\right]\right)$
5: $\quad w_k^h = w_{k-1}^h$
6: $\quad$ **for** $\ell = 1$ **to** n_{fp} **do**
7: $\quad\quad w_k^h = \texttt{ft-rankadapt}\left(T_{\mu}^{h}\left(w_k^h\right), \epsilon\right)$
8: $\quad$ **end for**
9: $\quad \delta = \|w_k^h - w_{k-1}^h\|^2$
10: $\quad k \leftarrow k + 1$
11: **end while**

the rank of the policy. Instead, to evaluate an implicit policy μ_k at a location z, one is required to solve the optimization problem (4.11) using the fixed value function w_{k-1}^h. However, one can store the policy evaluated at nodes visited by the cross approximation algorithm to avoid recomputing them during each iteration of the loop shown in Line 6. Since, the number of nodes visited during the approximation stage should be relatively low, this does not pose an excessive algorithmic burden.

4.3.4 FT-based prolongation and interpolation operators

We now show how to perform interpolation and prolongation in FT format for use within the multi-level methods described in Section 4.2.3. Recall that the first ingredient of multigrid is a prolongation operator I_h^{2h} which takes functions defined on the grid $\mathscr{X}^h$ into a coarser grid $\mathscr{X}^{2h}$. The second ingredient is an interpolation operator I_{2h}^h which interpolates functions defined on $\mathscr{X}^{2h}$ onto the functions defined on $\mathscr{X}^h$. Many of the common operators used for I_h^{2h} and I_{2h}^h can take advantage of the low rank structure of any functions on which they are operating. In particular, performing these operations on function in low-rank format often simply requires performing their one-dimensional variants onto each univariate scalar-valued function of its FT core.

The *prolongation operator* that we use picks out values common to both $\mathscr{X}^h$ and $\mathscr{X}^{2h}$ according to

$$(I_h^{2h} w^h)(z) = w^{2h}(z), \quad \forall z \in \mathscr{X}^{2h}.$$

Practically, this operator requires a constant number of computational operations, only access to memory. Furthermore, the coefficients of $f_{\alpha_1,\alpha_2}^{(i,h)}$ (recall (4.18)) are reused to form the coefficients of $f_{\alpha_1,\alpha_2}^{(i,2h)}$.

Suppose that fine grid is $\mathscr{X}_i^h = \{z_1^{(i)}, z_2^{(i)}, \ldots, z_n^{(i)}\}$ and the coarse grid $\mathscr{X}_i^{2h} = \{z_{2l-1}^{(i)}\}_{l=1,\ldots,n/2}$ consists of half the nodes ($|\mathscr{X}_i^{2h}| = n/2$). Then the univariate functions making up the cores of w^{2h} are

$$\begin{aligned} f_{j,k}^{(i,2h)}(x_i) &= \sum_{\ell=1}^{n/2} a_{j,k,\ell}^{(i,2h)} \phi_\ell^{2h}(x_i), \\ &= \sum_{\ell=1}^{n/2} a_{j,k,2\ell-1}^{(i,h)} \phi_\ell^{2h}(x_i), \end{aligned}$$

where ϕ_ℓ^{2h} are the functions defined on the coarser grid. In the second line we have used every other coefficient from the finer grid as the coefficient of the corresponding coarse grid function[3].

The *interpolation operator* arises from the interpolation that the FT performs, and in our case, the use of hat functions leads to a linear interpolation scheme. This means that if the scalar-valued univariate functions making up the cores of w^{2h} are represented in a nodal basis obtained at the tensor product grid $\mathscr{X}^{2h} = \mathscr{X}_1^{2h} \times \ldots \times \mathscr{X}_d^{2h}$, then we obtain a nodal basis with twice the resolution defined on $\mathscr{X}^h = \mathscr{X}_1^h \times \ldots \times \mathscr{X}_d^h$ through interpolation of each core. This operation requires interpolating of each of the univariate functions making up the cores of w^{2h} onto the

[3]One could also chose more regular nodal basis functions as well, e.g., splines. For other basis functions there may be more natural prolongation and interpolation operators than we present here. We have chosen hat functions in this paper because they are proved to be well behaved in the face of discontinuities or extreme nonlinearities often encountered in the solution of the HJB equation.

fine grid. Thus, w^h becomes an FT with cores consisting of the univariate functions

$$f_{j,k}^{(i,h)}(x_i) = \sum_{\ell=1}^{n} a_{j,k,\ell}^{(i,h)} \phi_\ell^h(x_i) = \sum_{\ell=1}^{n} f_{j,k}^{(i,2h)}(z_\ell^{(i)}) \phi_\ell^h(x_i),$$

where we note that in the second equation we use evaluations of the coarser basis functions to obtain the coefficients of the new basis functions. In summary, both operators can be applied to each FT core of the value function separately. Both of these operations, therefore, scale linearly with dimension.

One example of the use of these operations is within the two-level V-grid algorithm provided by Algorithm 11. In this algorithm, an approximate solution for each equation is obtained by ℓ_i fixed point iterations at grid level h_i. An extension to other grid cycles and multiple levels of grids is straightforward and can be performed with all of the same operations.

Algorithm 11 Two-level FT-based V-grid for solving Equation (4.7)

Require: Discretization levels $\{h_1, h_2\}$ such that $h_1 < h_2$; Number of fixed point iterations at each level $\{\ell_1, \ell_2\}$; Initial value function $\widehat{w}_0^h$; policy μ; `ft-rankadapt` cross approximation and rounding tolerances $\epsilon = (\delta_{\text{cross}}, \epsilon_{\text{round}})$

$k = 0$
while not converged **do**
  for $\ell = 1, \ldots, \ell_1$ **do**
    $\widehat{w}_k^{h_1} \leftarrow$ `ft-rankadapt`$(T_\mu^{h_1}(w_k^{h_1}), \epsilon)$
  end for
  $r^h =$ `ft-rankadapt`$(T_{c,\mu}^h(\widehat{w}_k^{h_1}), \epsilon) - \widehat{w}_k^{h_1}$
  $\Delta w^{h_2} = 0$
  for $\ell = 1, \ldots, \ell_2$ **do**
    $\Delta w_{n+1}^{h_2} \leftarrow$ `ft-rankadapt`$(T_\mu^{h_2}(\Delta w_n^{h_2}; I_{h_1}^{h_2} r^h), \epsilon)$ # See (4.16)
  end for
  $\widehat{w}_{k+1}^{h_1} = \widehat{w}_k^{h_1} + I_{h_2}^{h_1} \Delta w^{h_2}$
  $k \leftarrow k + 1$
end while

4.4 Analysis

In this section, we address the convergence properties and computational complexity of the FT-based algorithms. First, we show the convergence and accuracy of

approximate FP iteration methods. Second, we discuss the algorithms' complexity.

To show convergence of our FT-based low-rank dynamic programming algorithms, we need to characterize the effect of a small approximation error at each FP iteration associated with value iteration and optimistic policy iteration. Furthermore, to prove their computational benefits, we show polynomial growth in complexity of these algorithms with state space dimension when the FT-ranks of the approximation to associated value functions grows polynomially with dimension.

The algorithms discussed in the previous section all rely on performing cross approximation of a function with a relative accuracy tolerance ϵ. Recall that in practice we can not guarantee that cross approximation will converge to an approximation that has a relative error ϵ. We can, however, give conditions under which an ϵ level approximation within VI/PI still allows convergence

In this section we describe the convergence and accuracy of the FT-based VI and PI. In Section 4.4.1, we show convergence of the algorithms when a guaranteed approximation error ϵ is made at each iteration. In Section 4.4.2, we show that this situation is satisfied by the cross approximation based algorithms when the value functions of interest have finite rank. In Section 4.4.3 we compute the computational cost of approximate VI in FT format.

4.4.1 Convergence of approximate fixed-point iterations

We start by showing that a small relative error made during each step of the relevant FP iterations results in a bounded overall approximation error.

We begin recalling the contraction mapping FP theorem.

Theorem 9 (Contraction mapping FP theorem, Proposition B.1 by Bertsekas [13]). *Let $\mathscr{R}$ be a complete vector space and $\bar{\mathscr{R}}$ be a closed subset. Then if $f : \bar{\mathscr{R}} \to \bar{\mathscr{R}}$ is a contraction mapping with modulus $\gamma \in (0,1)$, there exists a unique $w \in \bar{\mathscr{R}}$ such that*

$$w = f(w).$$

Furthermore, the sequence defined by $w_0 \in \bar{\mathscr{R}}$ and the iteration $w_k = f(w_{k-1})$ con-

verges to w for any $w \in \bar{\mathscr{R}}$ according to

$$\|w_k - w\| \leq \gamma^k \|w_0 - w\|, \quad k = 1, 2, \ldots$$

Theorem 9 can be used, for example, to prove the convergence of VI when the operator T^h is a contraction mapping. Our FT-based algorithms are based on approximations to contraction mappings. Therefore, it is important to understand when they converge and the accuracy which they attain. The cases of FT-based VI and FT-based FP iterations for solving (4.7) are addressed by Lemma 3 below.

Lemma 3 (Convergence of approximate FP iterations). *Let $\mathscr{R}^h$ be a closed subset of a complete vector space. Let $f : \mathscr{R}^h \to \mathscr{R}^h$ be a contractive mapping with modulus $\gamma \in (0, 1)$ and fixed point w^h. Let $\tilde{f} : \mathscr{R}^h \to \mathscr{R}^h$ be an approximate mapping such that*

$$\|\tilde{f}(w') - f(w')\| \leq \epsilon \|f(w')\|, \quad \forall w' \in \mathscr{R}^h, \tag{4.19}$$

for $\epsilon > 0$. Then, the sequence defined by $w_0^h \in \mathscr{R}^h$ and the iteration $w_k^h = \tilde{f}\left(w_{k-1}^h\right)$ for $k = 1, 2, \ldots$ satisfies

$$\|w_k^h - w^h\| \leq \epsilon \frac{1 - (\gamma\epsilon + \gamma)^k}{1 - (\gamma\epsilon + \gamma)} \|w^h\| + (\gamma\epsilon + \gamma)^k \|w_0^h - w^h\|.$$

Proof. The proof is a standard contraction argument. We are grateful to Ezra Tal for pointing out the following argument. We begin by bounding the difference between the kth iterate and the fixed point w^h:

$$\begin{aligned}
\|w_k^h - w^h\| &= \|w_k^h - f(w_{k-1}^h) + f(w_{k-1}^h) - w^h\| \\
&\leq \|w_k^h - f(w_{k-1}^h)\| + \|f(w_{k-1}^h) - w^h\| \\
&\leq \epsilon \|f(w_{k-1}^h)\| + \gamma \|\widehat{w}_{k-1}^h - w^h\| \\
&\leq \epsilon \|f(w_{k-1}^h) - w^h\| + \epsilon \|w^h\| + \gamma \|w_{k-1}^h - w^h\| \\
&\leq (\gamma\epsilon + \gamma) \|w_{k-1}^h - w^h\| + \epsilon \|w^h\|,
\end{aligned}$$

where the second inequality comes from the triangle inequality, the third comes from (4.19) and contraction, the fourth inequality arises again from the triangle inequality, the final inequality arises from contraction.

Using recursion results in

$$\begin{aligned}\|w_k^h - w^h\| &\leq (\gamma\epsilon + \gamma)\left[(\gamma\epsilon + \gamma)\left[(\gamma\epsilon + \gamma)\ldots + \epsilon\|w^h\|\right] + \epsilon\|w^h\|\right] + \epsilon\|w^h\| \\ &= \epsilon\|w^h\|\left[\sum_{\ell=0}^{k-1}(\gamma\epsilon + \gamma)^\ell\right] + (\gamma\epsilon + \gamma)^k\|\widehat{w}_0^h - w^h\|.\end{aligned}$$

Using the property of a sum of a geometric series yields our desired result

$$\|\widehat{w}_k^h - w^h\| \leq \epsilon\frac{1-(\gamma\epsilon+\gamma)^k}{1-(\gamma\epsilon+\gamma)}\|w^h\| + (\gamma\epsilon + \gamma)^k\|\widehat{w}_0^h - w^h\|.$$

□

Notice that, as expected, when $\epsilon = 0$ and $k \to \infty$ this result yields that the iterates w_k^h converge to the fixed point w^h. Second, the condition $\gamma\epsilon + \gamma < 1$ is required to avoid divergence. It effectively states that if the contraction modulus is small enough, a larger approximation error may be incurred. On the other hand, if the contraction modulus is large, then the approximation error must be small. In other words, this requirement can be thought as a condition for which the approximation can remain a contraction mapping; larger errors can be tolerated when the original contraction mapping has a small modulus, and vice-versa.

This result can be used within our MDP problems when T^h and T_μ^h are contraction mappings. In that situation, this result yields convergence for both the approximate VI algorithm and for solving the system (4.7) using a FP iteration, when the approximation errors can be guaranteed. The circumstances under which such guarantees can be made are covered in Section 4.4.2.

We can obtain an alternative bound, one that does not require any conditions on ϵ, if we assume that each iterate of an approximate FP iteration is bounded. The following result is an alternative to the previous theorem. It is more flexible with respect to the size of error ϵ as long as each iterate is bounded.

Lemma 4 (Convergence of approximate FP iterations with boundedness assumption). *Let $\mathscr{R}^h$ be a closed subset of a complete vector space. Let $f : \mathscr{R}^h \to \mathscr{R}^h$ be a contractive mapping with modulus $\gamma \in (0,1)$ and fixed point w^h. Let $\tilde{f} : \mathscr{R}^h \to \mathscr{R}^h$ be an approximate mapping such that*

$$\|\tilde{f}(w') - f(w')\| \leq \epsilon \|f(w')\|, \quad \forall w' \in \mathscr{R}^h, \tag{4.20}$$

for $\epsilon > 0$. Let $w_0^h \in \mathscr{R}^h$. Define a sequence the sequence $\{w_k^h\}$ according to the iteration $w_k^h = \tilde{f}\left(w_{k-1}^h\right)$ for $k = 1, 2, \ldots$. Assume that $\|w_k^h\| \leq \rho_1 < \infty$ so that $\|f\left(w_k^h\right)\| \leq \rho < \infty$. Then $\{w_k^h\}$ satisfies

$$\|w_k^h - f^{[k]}(w_0^h)\| \leq \frac{1-\gamma^k}{1-\gamma}\epsilon\rho, \quad \forall k, \tag{4.21}$$

so that,

$$\lim_{k\to\infty} \|w_k^h - w^h\| \leq \frac{\epsilon\rho}{1-\gamma}, \tag{4.22}$$

where $f^{[k]}$ denotes k applications of the mapping f.

Proof. The strategy for this proof again relies on standard contraction and triangle inequality arguments. Furthermore, it follows closely the proof of Proposition 2.3.2 (Error Bounds for Approximate VI) work by Bertsekas [13]. In that work, the proposition provides error bounds for approximate VI when an absolute error (rather than a relative error) is made during each approximate FP iteration. The assumption of boundedness that we use here will allow us to use the same argument as the one made by Bertsekas.

Note that (4.20) implies

$$\|w_k^h - f(w_{k-1}^h)\| = \|\tilde{f}(w_k^h) - f(w_{k-1}^h)\| \leq \epsilon \|f(w_{k-1}^h)\|. \tag{4.23}$$

Recall $f^{[k]}(w')$ denotes k applications of the operator f, i.e.,

$$f^{[k]}(w') = f^{[k-1]}(f(w')) = f^{[k-2]}(f(f(w'))) = \cdots = f(f(f(\cdots f(w')))).$$

Then using the triangle inquality, contraction, and (4.23), we have

$$\begin{aligned}\|w_k^h - f^{[k]}(w_0^h)\| &\leq \|w_k^h - f(w_{k-1}^h)\| + \|f(w_{k-1}^h) - f^{[2]}(w_{k-2}^h)\| + \cdots \\ &\quad + \|f^{[k-1]}(w_1^h) - f^{[k]}(w_0^h)\| \\ &\leq \epsilon\|f(w_{k-1}^h)\| + \gamma\epsilon\|f(w_{k-2}^h)\| + \cdots + \gamma^{k-1}\epsilon\|f(w_0^h)\|.\end{aligned}$$

The boundedness assumption yields

$$\|w_k^h - f^{[k]}(w_0^h)\| \leq \epsilon\rho + \gamma\epsilon\rho + \cdots + \gamma^{k-1}\epsilon\rho,$$

and the sum of a geometric series yields

$$\|w_k^h - f^{[k]}(w_0^h)\| \leq \frac{1-\gamma^k}{1-\gamma}\epsilon\rho.$$

Taking the limit $k \to \infty$ and using Theorem 9, where $\lim_{k\to\infty} f^{[k]}(w_0^h) = w^h$, yields (4.22). □

Note that (4.21) shows the difference between iterates of exact FP iteration and approximate FP iteration, and indicates that this difference can not grow larger than $\epsilon\rho$. Furthermore, this result does not require any assumption on the relative error ϵ.

4.4.2 Convergence of FT-based fixed-point iterations

For the above theorems to be applicable to the case of low-rank approximation using the `ft-rankadapt` algorithm, we need to be able to explicitly bound the error committed during each iteration of FT-based VI and each subiteration within optimistic PI. Recall that cross approximation has no known convergence results for functions whose separated forms are approximately low-rank as in the case of Section 3.2.2. However, for the case of finite-rank unfoldings in Section 3.2.1, the FT approximation can be guaranteed using the `ft-rankadapt` algorithm.

This guarantee stems from the proof of the existence of the CUR/skeleton decomposition for vector-valued functions provided in Section 2.3.1. In that proof, one

requires an upper bound for the rank and a routine to find linearly independent fibers to obtain an exact reconstruction. Therefore, we only consider the convergence of functions with finite FT rank to strictly adhere to the theorems of the previous section. This assumption is formalized below.

Assumption 4 (Bounded ranks of FT-based FP iteration). *Let $\mathscr{R}^h$ be a closed subset of a complete vector space. Let $f : \mathscr{R}^h \to \mathscr{R}^h$ be a contractive mapping with modulus $\gamma \in (0,1)$ and fixed point w^h. The sequence defined by the initial condition $w_0^h \in \mathscr{R}^h$ and the iteration $w_k^h = \texttt{ft-rankadapt}\left(f\left(w_{k-1}^h\right), \epsilon\right)$ has the property that the functions $\{f(w_k^h)\}$ have finite FT rank. In other words, each of the unfoldings of $f(w_k^h)$ have rank bounded by some $r < \infty$,*

$$\text{rank}\left[f(w_k^h)(x_{\leq l}, x_{>l})\right] < r < \infty,$$

for $l = 1, \ldots d-1$.

Since under Assumption 4 we are approximating a function with finite FT-ranks at each step of the fixed point iteration, we further assume that the `ft-rankadapt` algorithm converges. These assumptions lead to the satisfaction of the conditions required by Lemmas 3 and 4. The following result is immediate.

Theorem 10 (Convergence of the FT-based value iteration algorithm). *Let $\mathscr{R}^h$ be a closed subset of a complete vector space. Let the operator T^h of (4.8) be a contraction mapping with modulus γ and fixed point v^h. Define a sequence of functions according the an initial function $v_0^h \in \mathscr{R}^h$ and the iteration $v_k^h = \texttt{ft-rankadapt}\left(T^h\left(v_{k-1}^h\right), \epsilon\right)$. Assume Assumption 4. Furthermore, assume that $\|w_k^h\| \leq \rho_1 < \infty$ so that $\|f\left(w_k^h\right)\| \leq \rho < \infty$. Then, FT-based VI converges according to*

$$\lim_{k\to\infty} \|v_k^h - v^h\| \leq \frac{\epsilon\rho}{1-\gamma}.$$

In reality, we cannot guarantee these conditions. However, our numerical experiments indicate that these algorithms remain effective in situations when these theorems do not exactly hold.

4.4.3 Complexity of FT-based DP algorithms

We can also estimate the computational cost for each FP iteration based on the computational cost of the `ft-rankadapt` algorithm. Suppose that each dimension is discretized into n nodes, then a single interpolation of black box function having all FT ranks equal to r requires $\mathcal{O}(nr^2)$ evaluations of the black box function and $\mathcal{O}(nr^3)$ operations during the rounding (Algorithm 3) step of the rank estimation scheme. When the SOC is discretized using upwind differencing described in Section 4.2.2, we can make the following statement about the complexity of one step of an approximate FP iteration.

Proposition 17 (Complexity of the evaluation of (4.6)). *Let the evaluation of stage cost g^h, drift B, and diffusion $\mathcal{D}$ require n_{op} operations. Let the discretization $\mathscr{X}^h$ of the MCA method arise from a tensor product of n-node discretizations of each dimension of the state space. Let the resulting transition probabilities $p^h(z, z'|\bar{u})$ be computed according to the upwind scheme described in Section 4.2.2. Furthermore, let the value function w_k^h have ranks $\boldsymbol{r} = [r_0, r_1, \ldots, r_d]$ where $r_0 = r_d = 1$ and $r_k = r$ for $k = 1 \ldots d-1$. Then, evaluating*

$$g^h(z, \bar{u}) + \gamma \sum_{z' \in \mathscr{X}^h} p^h(z, z'|\bar{u}) w_k^h(z')$$

for a fixed $z \in \mathscr{X}^h$ and $\bar{u} \in \mathscr{U}$ requires $\mathcal{O}(n_{op} + d^2 n r^2)$ operations.

Proof. In the specified upwind discretization scheme, there exist $2d$ neighbors to which the transition probabilities are nonzero. Furthermore, in Section 4.2.2, we showed that computing all of these transition probabilities requires $\mathcal{O}(n_{\text{op}} + d)$ operations. The evaluation of the cost of each neighbor, $w_k^h(z')$, requires $\mathcal{O}(dnr^2)$ evaluations. Since this evaluation is required for $2d$ neighbors, a conservative estimate for this total cost is $\mathcal{O}(d^2 n r^2)$. Thus, the result is obtained by observing that the cost is dominated by the computation of the transition probabilities and the evaluation of value function at the neighboring grid points □

Using Proposition 17 and assuming that for each z the minimization over control

$\bar{u}$ requires κ evaluations of (4.6), the following result is immediate.

Theorem 11 (Computational complexity of the FT-based value iteration algorithm). *Let $\mathscr{R}^h$ be a closed subset of a complete vector space. Let the operator T^h of (4.8) be a contraction mapping with modulus γ and fixed point v^h. Let the evaluation of stage cost g^h, drift B, and diffusion $\mathcal{D}$ corresponding to T^h require n_{op} operations. Let the discretization $\mathscr{X}^h$ of the MCA method arise from a tensor product of n-node discretizations of each dimension of the state space.*

Define a sequence of functions according the an initial function $v_0^h \in \mathscr{R}^h$ and the iteration $v_k^h = \texttt{ft-rankadapt}\left(T^h\left(v_{k-1}^h\right), \epsilon\right)$. Assume that for each $z \in \mathscr{X}_h$, the minimization over control $\bar{u} \in \mathscr{U}$ requires κ evaluations of (4.6). Then, each iteration of FT-based VI involves cross approximation and rounding and requires

$$\mathcal{O}\left(dnr^2\kappa\left(n_{op} + d^2nr^2\right) + dnr^3\right)$$

operations.

Note that this computational cost is polynomial with dimension, and therefore, this algorithm mitigates the curse of dimensionality as long as the rank r of the problem does not grow exponentially with dimension.

4.5 Numerical examples

Next, we demonstrate the low-rank approaches discussed in this chapter on several representative stochastic optimal control problems. The simulation results in this section are obtained using one core of a 32 GB Intel i7-5930K CPU clocked at 3.50GHz with a 64-bit Ubuntu 14.04 LTS operating system. We also used the Compressed continuous computation (C^3) toolbox [54], that is a BSD licensed package available through GitHub. The low-rank dynamic programming algorithms are provided in a stochastic optimal control addition to this software that is released separately, also on GitHub [55].

4.5.1 Linear-Quadratic-Gaussian with bounded controls

In this section, we investigate the effect of boundary conditions and control bounds on a prototypical control problem. The system has a bounded state space, linear dynamics and quadratic cost. However, combined with a bounded control space, analytic solutions difficult to obtain. The dynamics are

$$dx_1 = x_2 dt + \sigma_1 dw_1(t)$$
$$dx_2 = u(t)dt + \sigma_2 dw_2(t),$$

and they represent a physical system with position x_1 and velocity x_2 with $\mathscr{O} = (-2, 2)^2$. The control input to this system is the acceleration u, and we consider different lower and upper bounds on the control space $\mathscr{U} = [u_{lb}, u_{ub}]$. The diffusions affecting each equation are constant with space and time. We consider a discounted-cost infinite-horizon problem, with discount $e^{-\frac{t}{10}}$. The stage cost is

$$g(x, u) = x_1^2 + x_2^2 + u^2, \tag{4.24}$$

and, when imposing absorbing boundary conditions, the terminal cost is

$$\psi(x) = 100, \quad x \in \partial\mathscr{O} \tag{4.25}$$

We first solve the problem for various parameter values to gain insight to the problem. Next, we discuss numerical results summarizing the convergence of the algorithm.

Parameter studies

We present computational experiments that assess when low-rank cost functions arise and what factors affect ranks. Our first computational experiment studies the effects of the control boundary conditions on a problem with absorbing boundaries. This computational experiment is performed over several different u_{lb}, u_{ub} combinations

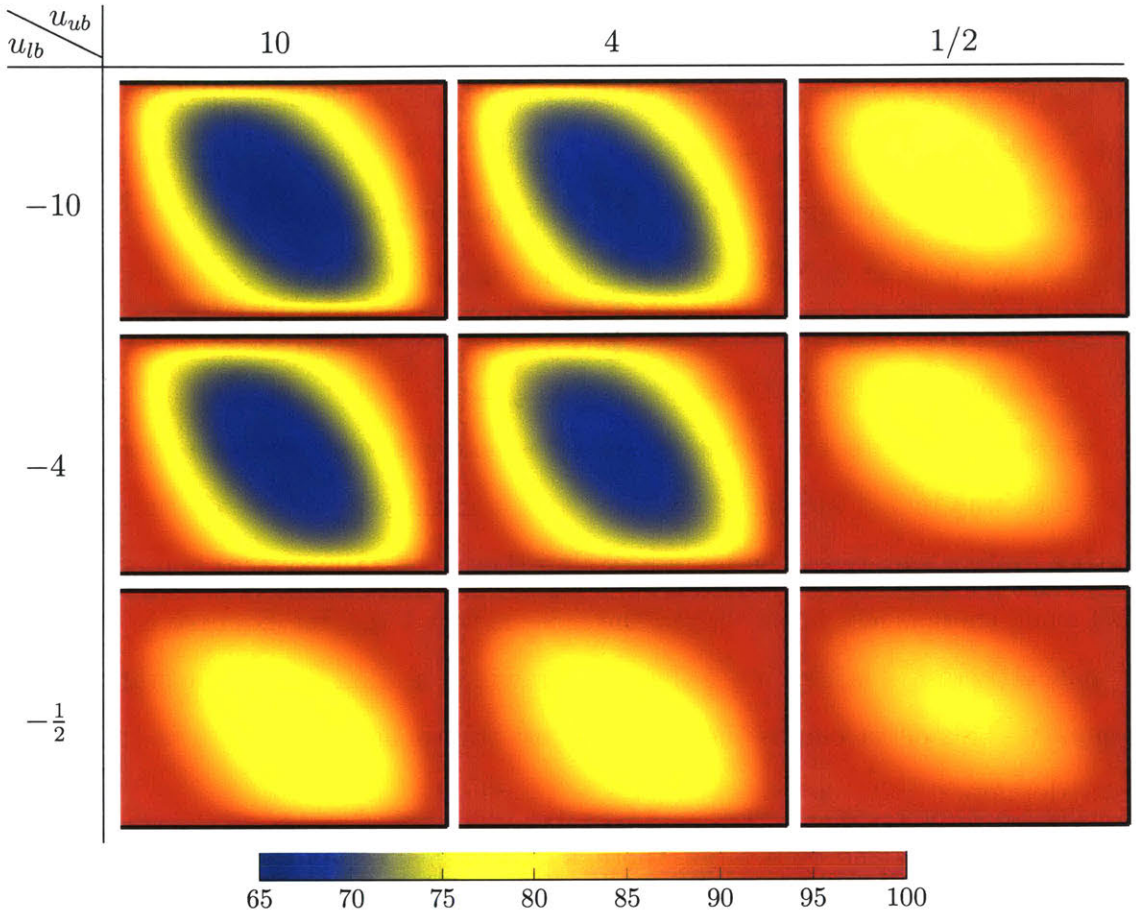


Figure 4-3: Cost functions and ranks of the solution to the LQG problem for varying control bounds $[u_{lb}, u_{ub}]$. Diffusion magnitudes are $\sigma_1 = \sigma_2 = 1$, and absorbing boundary conditions are used. The FT ranks found through the cross approximation algorithm with rounding tolerance $\epsilon_{\text{round}} = 10^{-7}$ were either 7 or 8. The x-axis of each plot denotes x_1 and the y-axis denotes x_2.

and the resulting optimal value functions are shown in Figure 4-3. We utilized FT-based policy iteration. The Markov chain approximation is used with a discretization of 60 points in each dimension.

Several phenomena are evident in Figure 4-3. When the range of the valid controls is wide, the value function is able to achieve smaller values, i.e., the blue region (indicating small costs) is larger with a wider control range. This characteristic is expected since the region from which the state can avoid the boundary is larger when more control can be exercised. However, the alignment of the value function, along

the diagonal stretching from the upper left to the upper right, is the same for all of the test cases. Only the magnitude of the value function changes, and therefore the ranks of the value functions are all either rank 7 or 8. Changing the bounds of the control space does not greatly affect the ranks of this problem.

Next, we consider the effect of the diffusion magnitude on the optimal value function and its rank. The results of this experiment are shown in Figure 4-4, and we see that the diffusion is influential for determining the rank of the problem. One striking pattern seen in Figure 4-4 is that as the diffusion decreases, the blue region grows in size. The blue region indicates low cost, and it intuitively represents the area from which a system can avoid the boundary. In other words, the system is more controllable when there is less noise. When the noise is very large, for example in the upper left panel, there is a large chance that the Brownian motion pushes the state into the boundary. This effect causes the terminal cost to propagate further into the interior of the domain.

As σ_2 decreases and σ_1 is large, there is less noise affecting the acceleration of the state, and the system remains controllable for a wide range of velocities. However, since the diffusion magnitude is large in the equation for velocity, the value function is only small when the position is close to the origin. In the area close to the origin there is less of a chance for the state to be randomly pushed into the absorbing region. Finally, when both diffusions are small, the system is controllable from a far greater range of states, as indicated by the lower right panel.

The ranks of the value functions follow the same pattern as controllability. The ranks are small for large diffusions and high for small diffusions. When the features of the function are aligned with the coordinate axes, the ranks remain low. As the function becomes more complex, due to small diffusions, the boundaries between guaranteed absorption and nonabsorption begin to have more complex shapes, resulting in increased ranks.

Next, we investigate the value functions associated with reflecting boundary conditions. The results of this experiment are shown in Figure 4-5. The results indicate that neither that value functions nor their ranks are much affected by the bounds of

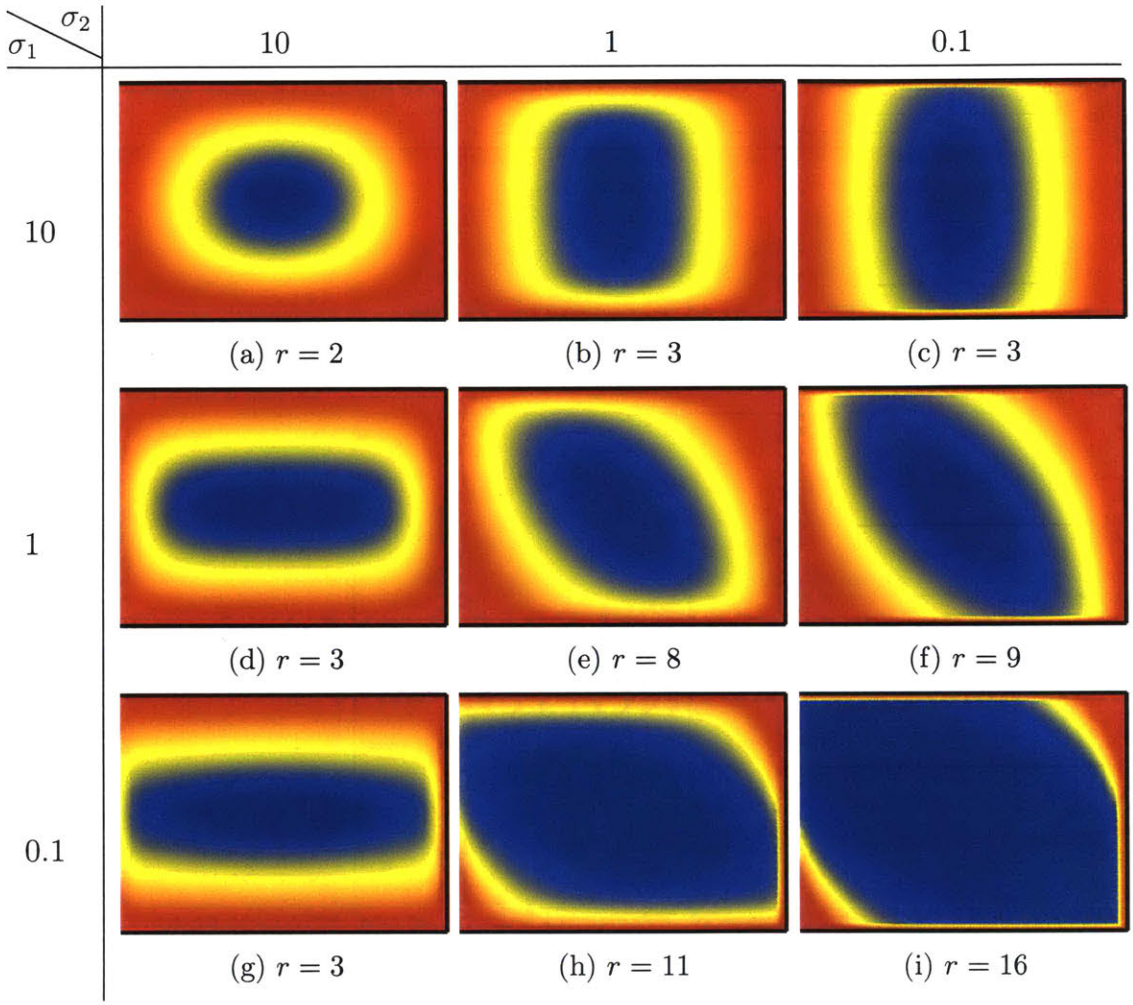


Figure 4-4: Cost functions and ranks of the solution to the LQG problem for different σ_1, σ_2. We fix $u_{lb} = -3$, $u_{ub} = 3$, and use absorbing boundary conditions. The FT ranks found through the cross approximation algorithm with rounding tolerance $\epsilon_{\text{round}} = 10^{-7}$ are indicated by the caption for each frame. The x-axis of each plot denotes x_1 and the y-axis denotes x_2. The color scale is different in each plot to highlight each function's shape.

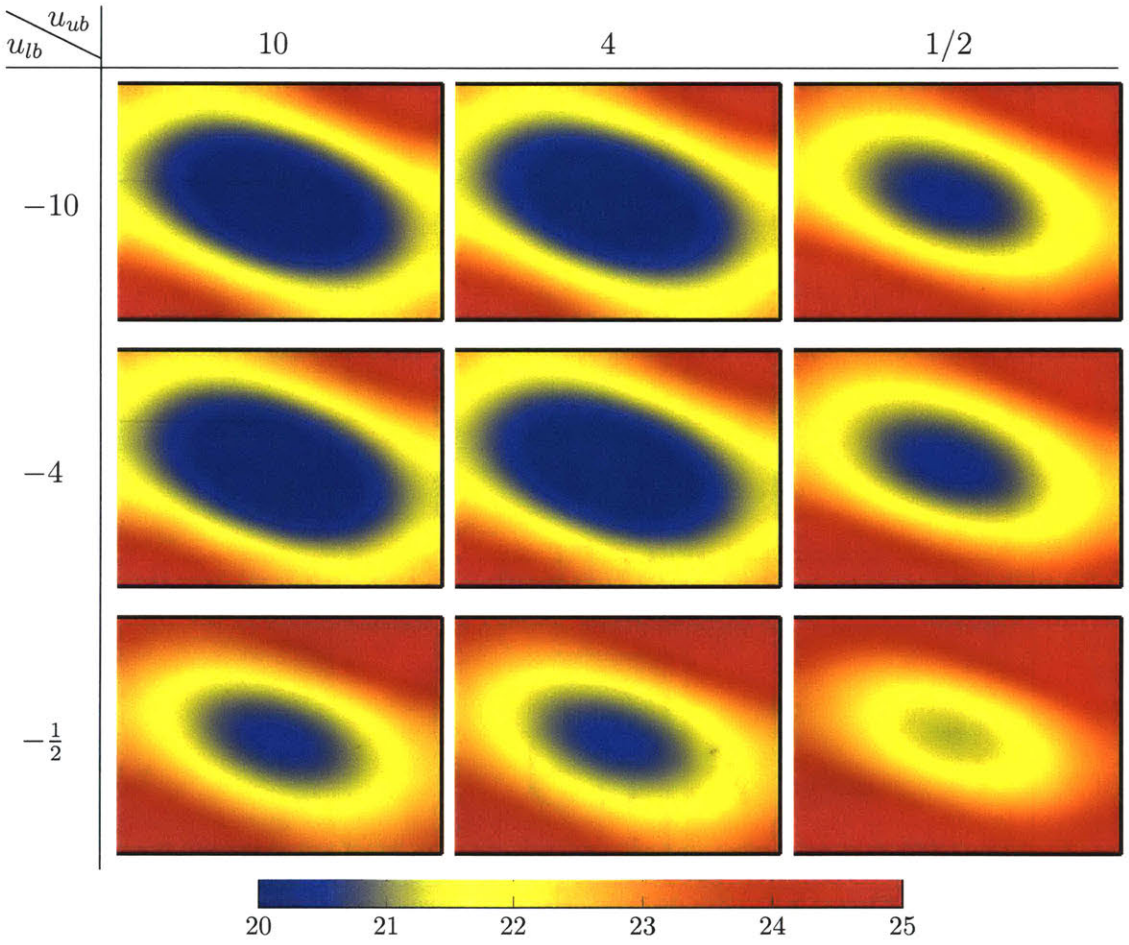


Figure 4-5: Reflecting boundary conditions with cost functions and ranks for different $[u_{lb}, u_{ub}]$, and the diffusion is set to $\sigma_1 = \sigma_2 = 1$. The FT ranks found through the cross approximation algorithm with rounding tolerance $\epsilon_{\text{round}} = 10^{-7}$ were 3 for all cases.

the control space. Interestingly, all of the value functions are rank 3, and all of the value functions appear to be quadratic. For a classical LQR problem the solution is quadratic, and therefore its rank is also 3. Thus, in some sense, reflecting boundary conditions more accurately represent a the classical LQG problem with unbounded state.

Next, we consider the effect of the diffusion magnitude on the optimal value function and its rank in the context of reflecting boundary conditions. The results of this experiment are shown in Figure 4-6, where it is evident that diffusion magnitude is again influential for determining the rank of this problem. The shapes and ranks of these value functions are similar to those in the case of absorbing boundary conditions. They follow the same pattern of increasing rank as the diffusion magnitude decreases.

Convergence

Next, we show values of the norm of the value function, the difference between iterates, and the fraction of states visited during each iteration throughout both the VI and PI algorithms. We begin with FT-based VI, whose pseudocode is provided by Algorithm 9. In Algorithm 9 we use an FT tolerances of $\delta_{\text{cross}} = \epsilon_{\text{round}} = 10^{-7}$. In Figure 4-7 we compare the convergence and the computational cost for solving the stochastic optimal control problem for varying discretization of the MCA method. Discretization sizes of $n = 25$, $n = 50$, and $n = 100$ nodes in each dimension are compared (note this corresponds to $25^2 = 625$,$50^2 = 2500$, and $100^2 = 10000$ total discretized states).

The results demonstrate that approximately the same value function norm is obtained regardless of discretization. However, convergence is much faster for coarse discretizations, and suggests the the one-way multigrid algorithm may be useful. Furthermore, we see the low-rank nature of the problem emerge because the fraction of discretized states evaluated during cross approximation for each iteration decreases with increasing grid resolution. For reference, exact VI would have the fraction of state space evaluated be 1 for each iteration. Thus, even in this two-dimensional

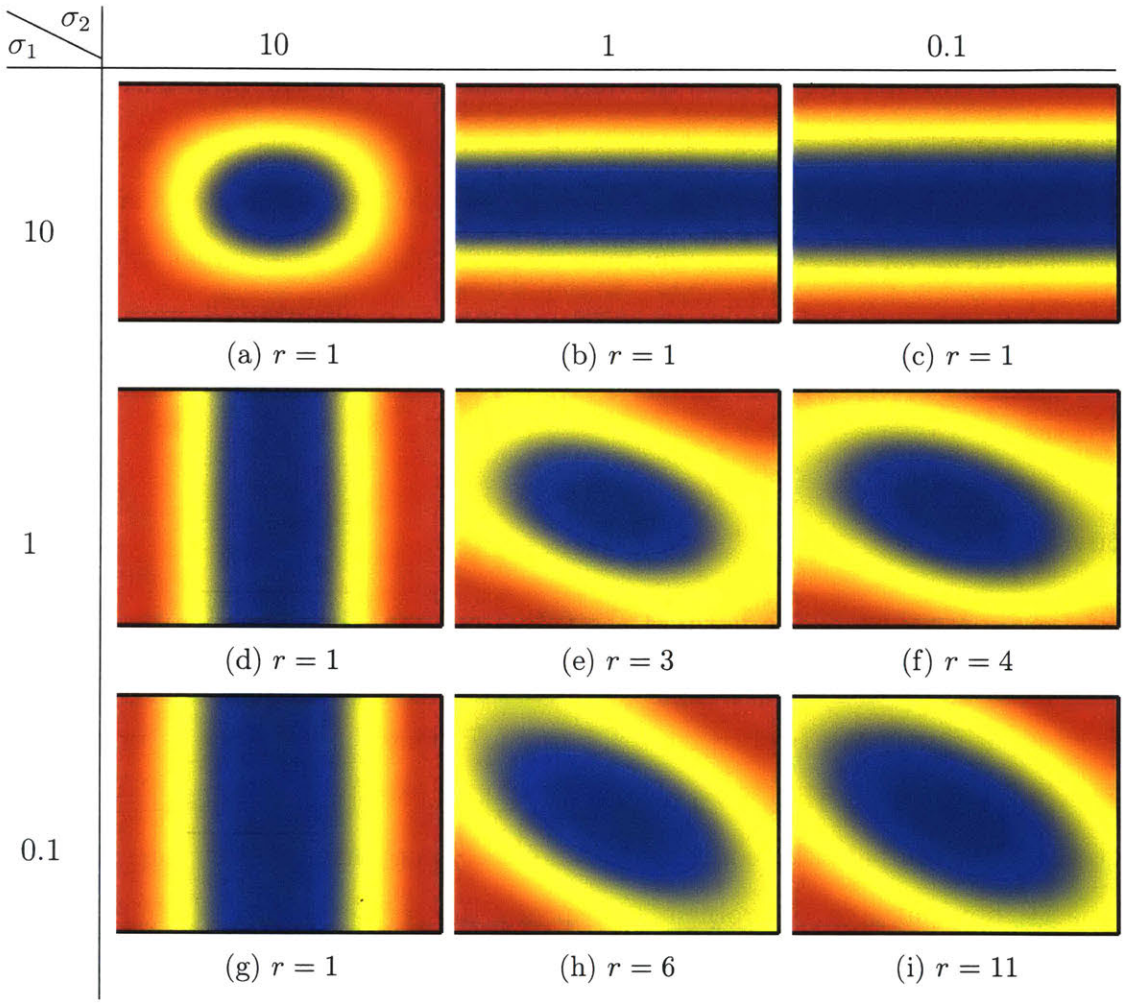


Figure 4-6: Cost functions and ranks of the solution to the LQG problem for different σ_1, σ_2. We fix $u_{lb} = -3$, $u_{ub} = 3$, and use reflecting boundary conditions. The FT ranks found through the cross approximation algorithm with rounding tolerance $\epsilon_{\text{round}} = 10^{-7}$ are indicated by the caption for each frame. The x-axis of each plot denotes x_1 and the y-axis denotes x_2. The color scale is different in each plot to highlight each function's shape.

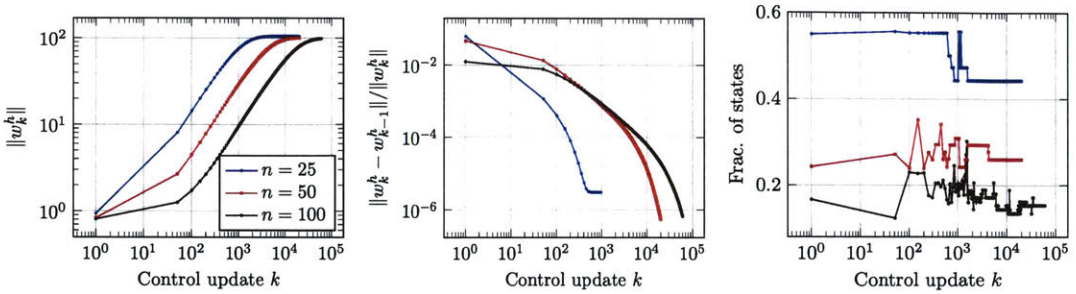


Figure 4-7: FT-based value iteration diagnostic plots for the linear quadratic problem with reflecting boundaries, $u(t) \in [-1, 1]$, and $\sigma_1 = \sigma_2 = 1$. The left panel shows that for all discretization levels the value function norm converges to approximately the same value. The middle panel shows the relative difference between value functions of sequential iterations. The right panel shows that the fraction of states evaluated within cross approximation decreases with increasing discretization resolution.

example, the low-rank algorithm achieves computational speedup between two to five times *for each iteration* over standard VI.

Next, we repeat this experiment for FT-based PI using Algorithm 10. In Algorithm 10 we use 10 sub-iterations to solve for the value function for every policy and FT tolerances of $\delta_{\text{cross}} = \epsilon_{\text{round}} = 10^{-7}$. Figure 4-8 shows the results for the value function associated with each control update; thus, the sub-iteration cost functions are not plotted. Note that, as expected, far fewer iterations are required for convergence. The results are qualitatively similar to the VI case; however, convergence occurs with approximately an order of magnitude fewer control updates.

4.5.2 Car dynamics

Next, we consider three and four state car dynamics. For all of the examples in this section, Algorithm 10 uses $n_{fp} = 10$, and cross approximation and rounding tolerances are set to 10^{-5}.

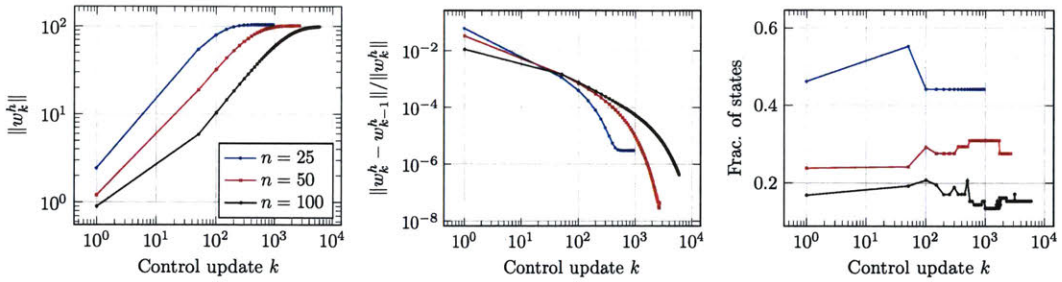


Figure 4-8: FT-based policy iteration diagnostic plots for the linear quadratic problem with reflecting boundaries, $u(t) \in [-1, 1]$, and $\sigma_1 = \sigma_2 = 1$. The left panel shows that for all discretization levels the value function norm converges to same value. The middle panel shows the relative difference between value functions of sequential iterations. The right panel shows that the fraction of states evaluated decreases with increasing discretization.

Dubin's car

The first car we consider is the standard three-state Dubin's car with stochasticity added

$$dx = \cos(\theta)dt + dw_1(t)$$
$$dy = \sin(\theta)dt + dw_2(t)$$
$$d\theta = u(t)dt + 10^{-2}dw_3(t)$$

for states $x \in (-4, 4)$, $y \in (-4, 4)$, and $\theta \in [-\pi, \pi]$. The control space consists of three options $\mathscr{U} = \{-1, 0, 1\}$. Boundary conditions are absorbing for the position dimensions (x, y) and periodic for the angle θ. An absorbing region is specified at the origin with a width of 0.5, and the terminal costs are

$$\psi(x, y, \theta) = \begin{cases} 0 & \text{for } (x, y) \in [-0.25, 0.25]^2 \\ 10 & \text{for } |x| \geq 2 \text{ or } |y| \geq 2 \end{cases}$$

A minimum time problem is posed, i.e., the stage costs are set to

$$g(x, u) = 1.$$

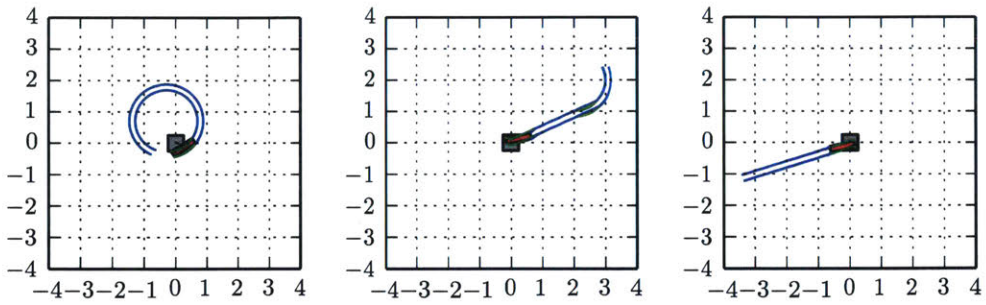


Figure 4-9: Trajectories of the Dubin's car for three initial conditions. Panels show the car when it arrives in the absorbing region.

The solution is obtained using one-way multigrid as discussed in Section 4.2.3. One hundred steps of FT-based PI are used and the state space is discretized into $n = 25$ nodes for each state. Then the result is interpolated onto a grid that is obtained by discretizing each dimension into $n = 50$ nodes, and the problem is solved with fifty steps of FT-based PI. This multigrid procedure is repeated for $n = 100$ and $n = 200$. A simulation of the resulting feedback controller for several initial conditions is shown in Figure 4-9.

Convergence diagnostic plots are shown in Figure 4-10. These plots demonstrate one of the advantages of the FT framework; since the value function is represented as function rather than an array, we can compare the norms of functions represented with different discretizations. In fact, the upper left panel of Figure 4-10 demonstrates that the norm *continuously* decreases when the discretization is refined.

Furthermore, these results suggest that the solution to this problem, with a rounding tolerance $\epsilon_{\text{round}} = 10^{-5}$, indeed has FT rank 12. This suggestion is supported by the fact that once the grid is refined enough, the fraction of states evaluated decreases, but the maximum rank levels off at 12. In some sense, we have found a characterization of some intrinsic complexity of the problem. Note that, the total number of states changes throughout the iterations depending on the discretization level, i.e., when $n = 25$ the total number of states is $25^3 = 15625$, when $n = 50$ the total number is $50^3 = 12500$, and the lower left panel shows the fraction of states with the corresponding discretization level.

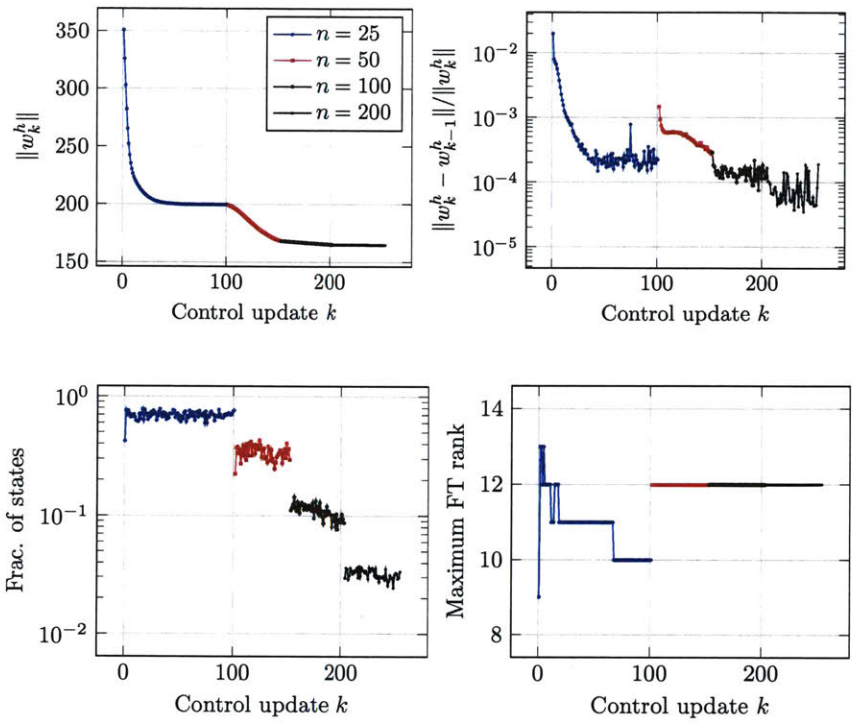


Figure 4-10: One-way multigrid for solving the Dubin's car control problem.

Understeered car

Next, we increase the state and control dimensions. We model an understeered car with control inputs that correspond to steering angle and acceleration. The states (x, y, θ, v) are now the x-position, the y-position, orientation angle, and velocity respectively. The control for steering angle is $u_1(t) \in [-15\frac{\pi}{180}, 15\frac{\pi}{180}]$ and for acceleration is $u_2(t) \in [-1, 1]$. The dynamics are given by

$$
\begin{aligned}
dx &= v\cos(\theta)dt + dw_1(t) \\
dy &= v\sin(\theta)dt + dw_2(t) \\
d\theta &= \frac{1}{1 + (v/v_c)}\frac{v}{L}\tan(u_1(t))dt + 10^{-2}dw_3(t) \\
dv &= \alpha u_2(t)dt + 10^{-2}dw_4(t)
\end{aligned}
$$

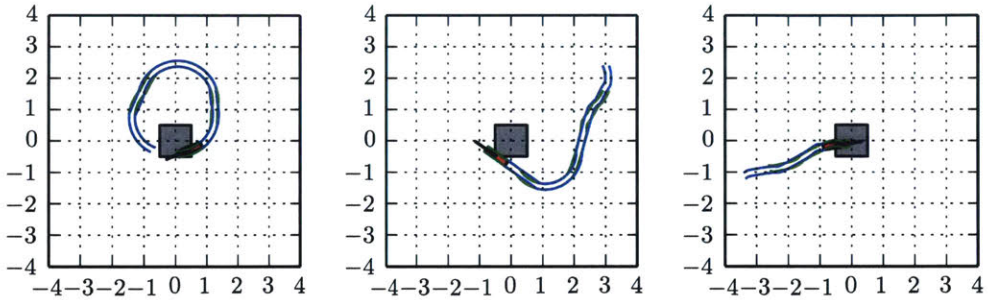


Figure 4-11: Trajectories of the understeered car for three initial conditions. Panels show the car when it arrives in the absorbing region.

where $v_c = 8\text{m/s}$ is the characteristic speed, $L = 0.2\text{m}$ is the length of the car, and $\alpha = 2$. The boundary conditions are absorbing for the the positions x and y, periodic for θ, and reflecting for v. The space for the positions and orientations are identical to that of the Dubin's car. The velocity is restricted to forward with $v \in [3, 5]$. The stage cost is altered to push the car to the center

$$g(x, y, \theta, v) = 1 + x^2 + y^2.$$

The terminal cost is the same as for the Dubin's car with an absorbing region at the origin of the x, y plane. Note that the dynamics of this example are not affine with respect to control. Therefore, this example does not fit into many standard frameworks that solve a corresponding linear HJB equation [67, 127]. The trajectories for various starting locations are shown in Figure 4-11.

Figure 4-12 shows the convergence diagnostic plots. Due to computational considerations, we fixed a maximum FT adaptation rank to 20. Therefore, instead of plotting the maximum rank (like we did for the Dubin's car), we plot the *average* FT rank in the lower right panel. The average rank varies more for coarse discretizations than for fine discretizations, but when $n = 100$ the average rank becomes clearly smaller and more consistent.

For this four dimensional state space, a discretization of 100 nodes in each dimension would require storage that is on the order of $\mathcal{O}(8 \times 100^4) = \mathcal{O}(0.8 \times 10^9)$ bytes

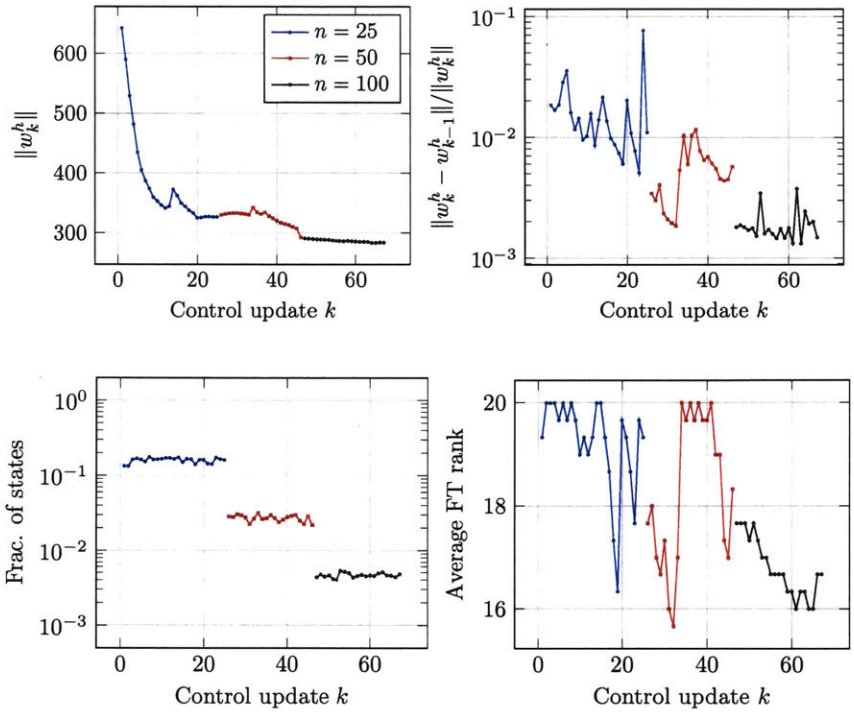


Figure 4-12: One-way multigrid for solving the understeered car control problem. Maximum rank FT rank is restricted to 20.

(suppose we are storing floating point values), or 800 MB. The compressed format leads to a storage requirement that is a little more than two orders of magnitude less, as indicated by the lower left panel. In fact, on the desktop that generated these results, storing the final value function requires almost exactly 1MB. These savings can be tremendously beneficial in constrained computing environments. For example, these controllers can potentially be used on embedded systems that have serious memory constraints.

4.5.3 Perching glider dynamics

We now solve a seven state, one control, problem with dynamics modeling an unpowered glider attempting to perch on a horizontal string [32, 89, 104]. The glider is described by flat-plate model in a two dimensional plane involving seven state variables $(x, y, \theta, \phi, v_x, v_y, \dot{\theta})$ specifying its x-position, y-position, angle of attack, elevator

angle, horizontal speed, vertical speed, and the rate of change of the angle of attack respectively. The input control is the rate of change of the elevator angle $u = \dot{\phi}$. A successful perch is defined by a horizontal velocity between 0 and 2 m/s, a vertical velocity between -1 and -3 m/s, and the x and y positions of the glider within a 5cm radius of the perch. Under these conditions, the experimental aircraft in [32] can attach to the string. For a diagram of this glider refer to either [104] or [89].

The dynamics of the glider are given by

$$\begin{aligned}
\mathbf{x}_w &= [x - l_w c_\theta, y - l_w s_\theta], \quad \dot{\mathbf{x}}_w = [\dot{x} + l_w \dot{\theta} s_\theta, \dot{y} - l_w \dot{\theta} c_\theta] \\
\mathbf{x}_e &= [x - l c_\theta - l_e, c_{\theta+\phi}, y - l s_\theta - l_e s_{\theta+\phi}] \\
\dot{\mathbf{x}}_e &= [\dot{x} + l\dot{\theta} s_\theta + l_e(\dot{\theta} + u) s_{\theta+\phi}, \dot{y} - l\dot{\theta} c_\theta - l_e(\dot{\theta} + u) c_{\theta+\phi}] \\
\alpha_w &= \theta - \tan^{-1}(\dot{y}_w, \dot{x}_w), \quad \alpha_e = \theta + \phi - \tan^{-1}(\dot{y}_e, \dot{x}_e) \\
f_w &= \rho S_w |\dot{\mathbf{x}}_w|^2 \sin(\alpha_w), \quad f_e = \rho S_e |\dot{\mathbf{x}}_e|^2 \sin(\alpha_e)
\end{aligned}$$

$$\begin{aligned}
dx &= v_x dt + 10^{-9} dw_1(t) \\
dy &= v_y dt + 10^{-9} dw_2(t) \\
d\theta &= \dot{\theta} dt + 10^{-9} dw_3(t) \\
d\phi &= u dt + 10^{-9} dw_4(t) \\
dv_x &= \frac{1}{m} \left(-f_w s_\theta - f_e s_{\theta+\phi}\right) dt + 10^{-9} dw_5(t) \\
dv_y &= \frac{1}{m} \left(f_w c_\theta + f_e c_{\theta+\phi} - mg\right) dt + 10^{-9} dw_6(t) \\
d\dot{\theta} &= \frac{1}{I} \left(-f_w l_w - f_e(l c_\phi + l_e)\right) dt + 10^{-9} dw_7(t)
\end{aligned}$$

where ρ is the density of air, m is the mass of the glider, I is the moment of inertia of the glider, S_w and S_e are the surface areas of the wing and tail control surfaces, l is the length from the center of gravity to the elevator, l_w is the half chord of the wing, l_e is the half chord of the elevator, c_γ denotes $\cos(\gamma)$, and s_γ denotes $\sin(\gamma)$. The parameters of the model are chosen to be the same as those in [104].

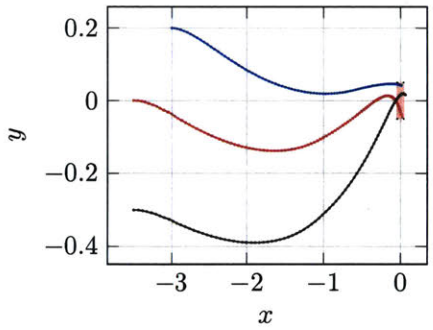


Figure 4-13: Trajectories of the perching glider for three initial conditions given by: $(-3, \frac{1}{5}, 0, 0, \frac{9}{2}, 0, 0)$ (blue), $(-\frac{7}{2}, 0, 0, 0, 5, 0, 0)$ (red), and $(-\frac{7}{2}, -\frac{3}{10}, 0, 0, 5.8, 0, 0)$ (brown). Target region is shown by shaded region that is centered at $(0, 0)$

The stage cost is specified as

$$g(x, y, \theta, \phi, v_x, v_y, \dot{\theta}) = 20x^2 + 50y^2 + \phi^2 + 11v_x^2 + v_y^2 + \dot{\theta}^2.$$

Absorbing boundary conditions are used, and an absorbing region is defined to encourage a successful perch. Let this region be defined for $x \in [-0.05, 0.05], y \in [-0.05, 0.05], v_x \in [-0.25, 0.25]$, and $v_y \in [-2.25, 2.25]$. The terminal cost for both of these regions is

$$\psi(x, y, \theta, \phi, v_x, v_y, \dot{\theta}) = \begin{cases} 0 & \text{if perched} \\ 600x^2 + 400y^2 + \frac{1}{9}\theta^2 + \frac{1}{9}\phi^2 + v_x^2 + (v_y + 1.5)^2 + \frac{1}{9}\left(\dot{\theta} + 0.5^2\right) & \text{otherwise.} \end{cases}$$

Several trajectories of the controlled system are shown in Figure 4-13. These trajectories are generated with several initial conditions. Notice that they all follow the same pattern of diving, then climbing before dropping into the perch. This behavior is similar to that observed in experiment [32, 89, 104].

Figure 4-14 shows the convergence diagnostic plots for the one-way multigrid algorithm used for solving this problem. Notice that we used discretization levels of $n = 20, 40, 80$ and 160 nodes for each dimension. Thus the number of discretized states at the finest discretization is $160^7 \approx 2.6 \times 10^{15}$. Furthermore, we limited the

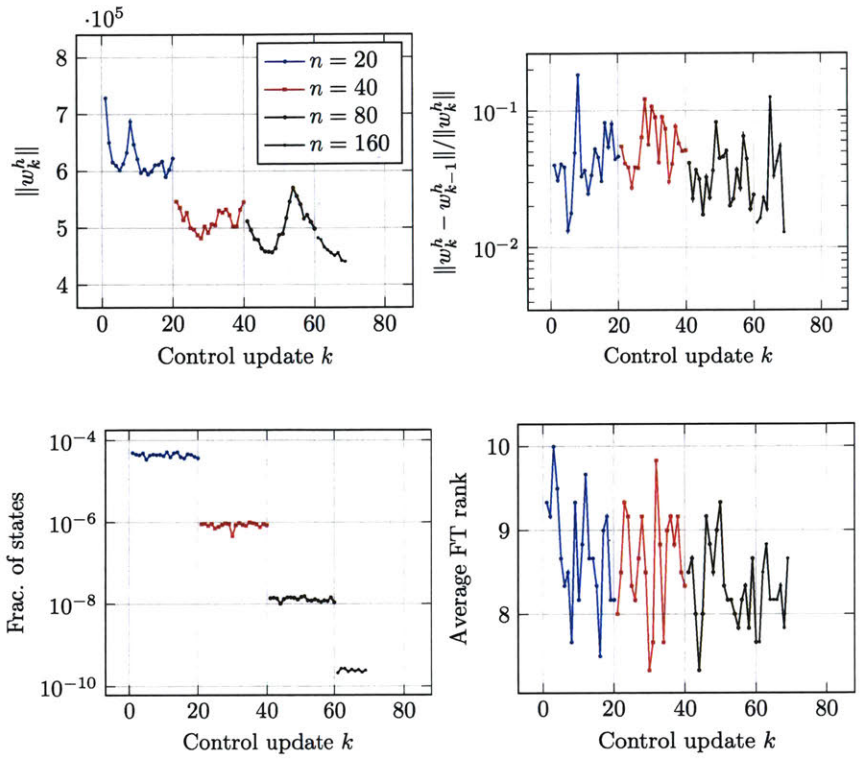


Figure 4-14: One-way multigrid for solving perching glider. Maximum rank FT rank is restricted to 10.

rank to a maximum of 10, and the panel in lower right panel shows that the average ranks are below this maximum threshold.

The final ranks of the value function translate into savings of approximately four orders of magnitude *per iteration* when $n = 20$ and a corresponding savings of ten orders of magnitude *per iteration* when $n = 160$. However, in this example it seems that the rank truncation is indeed affecting the convergence of the problem. A noisy and volatile decay of the value function norm is seen in the upper left panel. Furthermore, the difference between iterates, in the upper right panel, indicates a relative error of approximation 10^{-2} which is above our FT rounding threshold of 10^{-5}. Nonetheless, the resulting controller seems to be successful at achieving the desired behavior.

4.5.4 Quadcopter dynamics

Finally, we consider the problem of maneuvering a quadcopter through a small target region. This system has six states and three controls. The states (x, y, z, v_x, v_y, v_z) are the x-position in meters $x \in [-3.5, 3.5]$, y-position in meters $y \in [-3.5, 3.5]$, z-position in meters $z \in [-2, 2]$, x-velocity in meters per second $v_x \in [-5, 5]$, y-velocity in meters per second $v_y \in [-5, 5]$, and z-velocity in meters per second $v_z \in [-5, 5]$. The controls are the thrust (offset by gravity) $u_1 \in [-1.5, 1.5]$, the roll angle $u_2 \in [-0.4, 0.4]$, and the pitch angle $u_3 \in [-0.4, 0.4]$. The dynamics are given by

$$
\begin{aligned}
dx &= v_x dt + 10^{-1} dw_1(t) \\
dy &= v_y dt + 10^{-1} dw_2(t) \\
dz &= v_z dt + 10^{-1} dw_3(t) \\
dv_x &= \frac{u_1 - mg}{m} \cos(u_2) \sin(u_3) dt + 1.2 dw_4(t) \\
dv_y &= -\frac{u_1 - mg}{m} \sin(u_2) dt + 1.2 dw_5(t)) \\
dv_z &= \left(\cos(u_2) \cos(u_3) \frac{u_1 - mg}{m} + g \right) dt + 1.2 dw_6(t),
\end{aligned}
$$

and reflecting boundary conditions are used for every state. These dynamics are adapted from [22].

A target region is specified as a cube centered at the origin, and a successful maneuver is one which enters the cube with a forward velocity of one meter per second with less than 0.15m/s speed in the y and z directions. The terminal cost is assigned to be zero for this region, i.e.,

$$\psi(x, y, z, v_x, v_y, v_z) = 0, \quad \text{for } (x, y, z, v_x, v_y, v_z) \in [-0.2, 0.2]^3 \times [0.5, 1.5] \times [-0.2, 0.2]^2.$$

The stage cost is set to

$$g(x, y, z, v_x, v_y, v_z, u_1, u_2, u_3) = 60 + 8x^2 + 6y^2 + 8z^2 + 2u_1^2 + u_2^2 + 6u_3^2.$$

For computational considerations, the maximum FT-rank is restricted to 10. Theoretically, underestimating the ranks can potentially cause significant approximation errors, however, in this case we are still able to achieve a well performing controller. Investigating the effect of rank underestimation is an important area of future work.

Trajectories of the optimal controller for various initial conditions are shown in Figures 4-15 and 4-16. In Figure 4-15, the position and velocities each approach their respective absorbing conditions for each simulation. In the first simulation, the quadcopter quickly accelerates and decelerates into the goal region. The final positions do not lie exactly within the absorption region. This absorption region is, in a way, unstable since it requires the quadcopter enter with a forward velocity. The forward velocity requirement virtually guarantees that the quadcopter must eventually exit the absorption region. The second simulation starts with positive y and z velocities. The third simulation starts with a negative y velocity and the quadcopter far away from the origin.

Note that the controls are all fairly smooth except when the quadcopter state is near or in the absorption region. At this point, the roll angle (and pitch angle in the first and third simulations) oscillates rapidly around zero. This appears to be due to the flatness of the objective function. Since the quadcopter is so close to the absorption region the control inputs can change rapidly to ensure it stays there.

Figure 4-17 shows the convergence diagnostics. We used a one-way multigrid strategy with $n = 20$, $n = 60$, and $n = 120$ discretization nodes. The difference between iterates shown in the upper right panel is between $\mathcal{O}(1)$ and $\mathcal{O}(10)$. In fact, this means that the relative difference is approximately $\mathcal{O}(10^{-4})$, which matches well with our FT rounding tolerance ϵ_{round}. Thus, we can have some confidence that our rank underestimation scheme may not have incurred too great an error. The lower left panel indicates computational savings between 3 and 7 orders of magnitude per iteration. To make the size of this reduction more explicit, consider that storing the value function for $n = 120$ would require storing $10^6 = 2 \times 10^{12}$ floating point numbers (approximately 24 TB of data). However, in compressed format the final value function is stored in 778 KB.

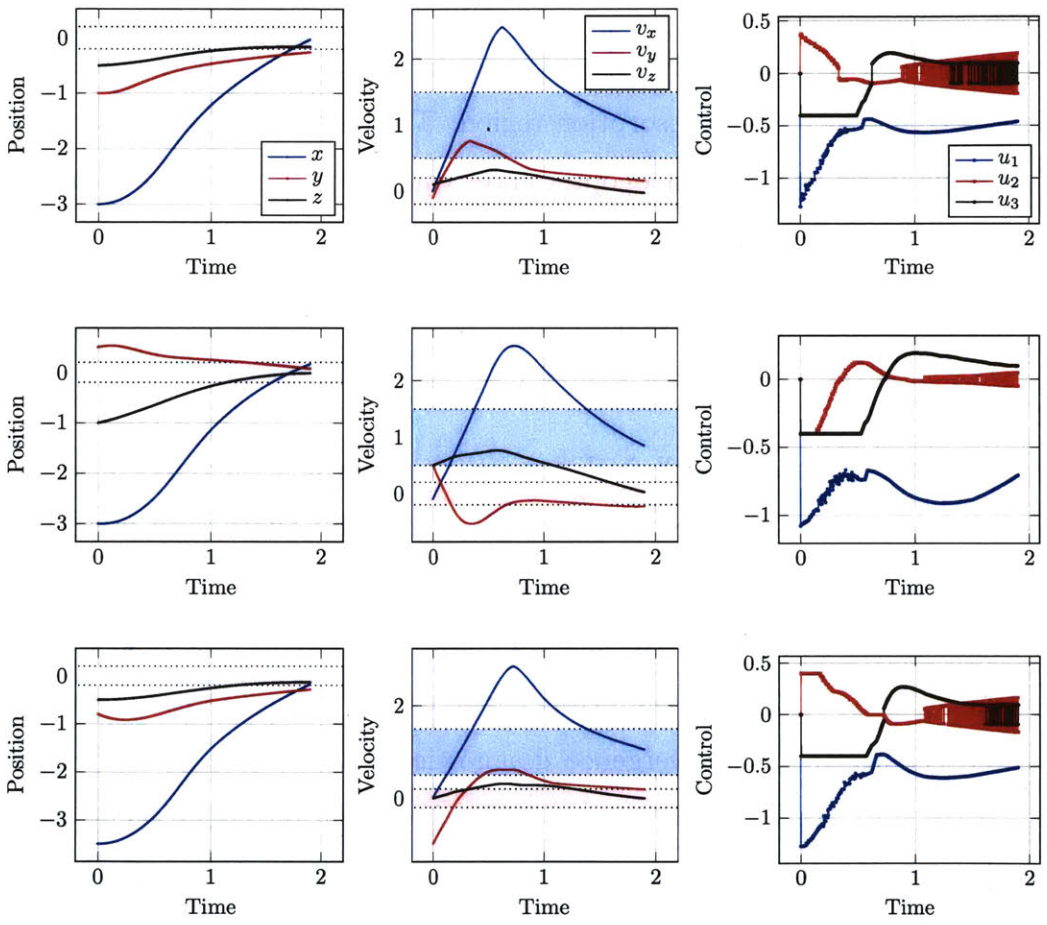


Figure 4-15: Three quadcopter simulations using optimal low-rank feedback controller. Shaded region indicates the target positions and velocities.

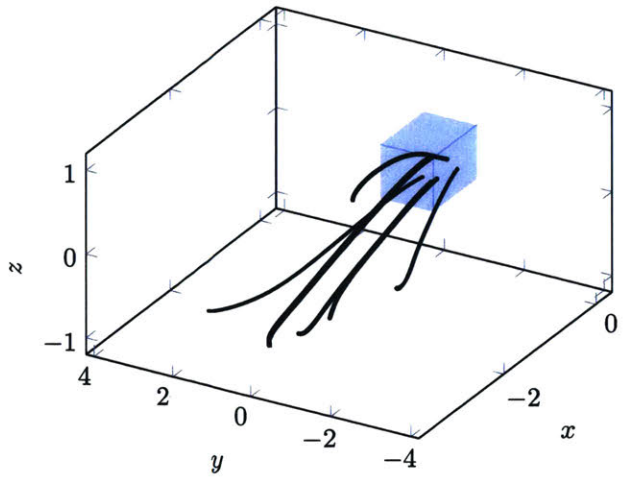


Figure 4-16: Three-dimensional trajectories showing the position of the quadcopter for various initial conditions.

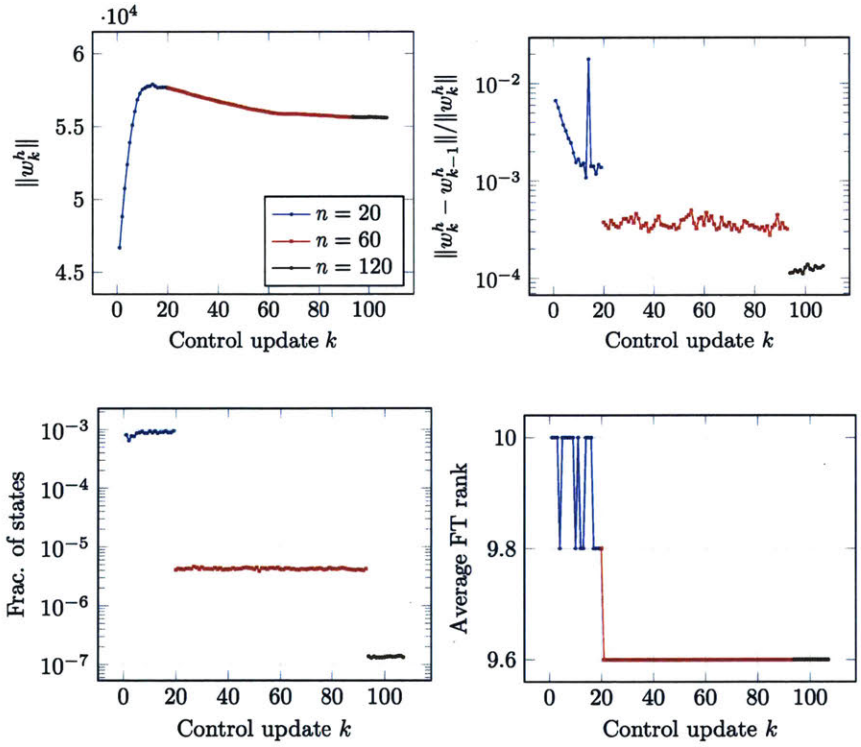


Figure 4-17: One-way multigrid for solving the quadcopter control problem. Maximum rank FT rank is restricted to 10.

Chapter 5

Low-rank algorithms for Gaussian filtering

The low-rank stochastic optimal control solution formulation in Chapter 4 relies on constructing optimal policies that are a function of the system state. However, in practice, autonomous systems typically do not have exact knowledge of the full system state. Instead, they observe the world through indirect and/or noisy measurements. In this chapter, we describe a probabilistic inference framework that convert these indirect and noisy observations into probability distributions that represent knowledge about the state of the system.

This *data assimilation* task is vitally important in many fields beyond autonomy. Tasks ranging from weather prediction [126] and ocean monitoring [24, 48, 84] to reentry vehicle tracking [33] all have in common the fact that they are systems that are difficult to observe in their entirety. Still, scientists and engineers are often required to both estimate or infer the system state and to use their inferences for predicting future events. This task is possibly most familiar in the context of weather prediction, where state uncertainty is often represented through an ensemble of plausible system states, and future prediction is made by propagating these ensemble members through models to assess the probability of certain weather events occurring in the future.

The particular task, whereby one incorporates observations of a dynamical system and uses these observations to both develop a probabilistic description of the system

state and to make future predictions is called *filtering*. In this chapter, we consider filtering algorithms for a system that evolves in *continuous* time and is observed at *discrete* time intervals. This generic system describes many practical problems of interest.

The filtering problem is computationally challenging for multiple reasons. Most importantly, it requires performing Bayesian inference repeatedly and, for some applications, in real time. Secondly, it requires propagating a probability distribution through a dynamical system. Both of these tasks are enormously challenging problems in their own right. Our contribution in this chapter is applying the low-rank functional decompositions described in Chapter 3 to an integration-based Gaussian framework for filtering. In this framework, equations are developed for the propagation of the mean and covariance of the distribution. These equations involve computing several multivariate integrals, and we perform this high-dimensional integration using the function-train.

5.1 Filtering problem

We now describe the continuous time state dynamics and the discrete time observation model, discuss existing literature for solving this problem, and detail the integration-based Gaussian filtering framework that we extend using low-rank techniques.

5.1.1 Continuous time state evolution

We consider a data assimilation problem in which that state $x(t) \in \mathscr{X} \subset \mathbb{R}^d$ evolves in continuous time according to a stochastic differential equation, and observations $y \in \mathscr{Y} \subset \mathbb{R}^{d_y}$ occur at discrete time intervals $t_k \in \mathscr{T} \subset [0, \infty)$. The evolution of this joint system is given by

$$dx = B(t, x)dt + \mathcal{D}(t, x)dw(t), \tag{5.1}$$

$$y_k = H(t_k, x(t_k)) + \eta_k, \quad \eta_k \sim \mathcal{N}(0, \mathbf{V}), \tag{5.2}$$

where $w(t) \in \mathbb{R}^{d_w}$ denotes Brownian motion, the drift $B : \mathscr{T} \times \mathscr{X} \to \mathbb{R}^d$ denotes the evolution of the system according to some "known" or baseline dynamics, the diffusion $\mathcal{D} : \mathscr{T} \times \mathscr{X} \to \mathbb{R}^{d \times d_w}$ accounts for model error or uncertainty in the state evolution, the observation operator $H : \mathscr{T} \times \mathscr{X} \to \mathbb{R}^{d_y}$ maps the system state to observations, and $\eta_k \in \mathbb{R}^{d_y}$ is Gaussian observation noise with zero mean and covariance $\mathbf{V}$. Note that as opposed to Chapter 4, the dynamics are not dependent on an external input.

As discussed in Chapter 3, $x(t)$ is a stochastic process endowed with a probability distribution. The objective of filtering is determining this probability distribution for the current time and future times given all the observations obtained prior to that current time. To be more specific, denote by $\boldsymbol{y}_t$ the collection of observations obtained prior to time t, i.e., $\boldsymbol{y}_t$ has the following elements

$$\boldsymbol{y}_t[k] = y_k, \quad \forall k \text{ such that } t_k < t. \tag{5.3}$$

Then, filtering concerns characterizing the function $p(t, x(t)|\boldsymbol{y}_t)$, where, for a fixed t, $p(t, x(t)|\boldsymbol{y}_t))$ is a probability density function for the distribution of $x(t)$.

The density $p(t, x(t)|\boldsymbol{y}_t)$ is implicitly defined through the a sequence of solutions to a Fokker-Planck partial differential equation

$$\begin{aligned}\frac{\partial}{\partial t} p(t, x|\boldsymbol{y}_{t_{k+1}}) = &- \sum_{i=1}^{d} \frac{\partial}{\partial x_i} \left[B\left[i\right](t, x) p(t, x|\boldsymbol{y}_{t_{k+1}}) \right] + \\ &\frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \frac{\partial^2}{\partial x_i \partial x_j} \left[\sum_{\ell=1}^{d_w} \mathcal{D}[i, \ell](t, x) \mathcal{D}[j, \ell](t, x) p(t, x|\boldsymbol{y}_{t_{k+1}}) \right]\end{aligned} \tag{5.4}$$

for $t \in (t_k, t_{k+1}]$, and an application of Bayes rule

$$p(t_{k+1}, x(t_{k+1})|\boldsymbol{y}_{t_{k+2}}) = \frac{p(y_{t_{k+1}}|t_{k+1}, x(t_{k+1}), \boldsymbol{y}_{t_{k+1}}) p(t_{k+1}, x(t_{k+1})|\boldsymbol{y}_{t_{k+1}})}{\int p(y_{t_{k+1}}|t_{k+1}, x(t_{k+1}), \boldsymbol{y}_{t_{k+1}}) p(t_{k+1}, x(t_{k+1})|\boldsymbol{y}_{t_{k+1}}) dx}, \tag{5.5}$$

where the relationship $\boldsymbol{y}_{t_{k+2}} = \left[\boldsymbol{y}_{t_{k+1}}, y_{t_{k+1}}\right]$, allows us to pose the likelihood, $p(y_{t_{k+1}}|t_{k+1}, x, \boldsymbol{y}_{t_{k+1}})$, for just $y_{t_{k+1}}$ instead of for the entire history of observations. These two equations are commonly called the *forecast* and *analysis*, respectively. They reflect the fact that (5.4) is responsible for propagating the uncertainty over

time, and (5.5) is responsible for analyzing data as it becomes available.

We can provide an explicit equation for the likelihood given both our assumption about Gaussian noise in (5.2) and the fact that (5.2) implies that the distribution on y_k is conditionally independent of any previous y_i for $i < k$ given the current state $x(t_k)$. This explicit representation is

$$p(y_{t_{k+1}}|t_{k+1}, x(t_{k+1}), \boldsymbol{y}_{t_{k+1}}) = p(y_{t_{k+1}}|t_{k+1}, x(t_{k+1})) \propto \exp\left(-\frac{1}{2}\|y_{t_{k+1}} - H(t, x(t_{k+1}))\|_{\mathbf{V}}\right), \tag{5.6}$$

where the covariance weighted norm $\|\cdot\|_{\mathbf{V}}$ is defined as

$$\|\boldsymbol{z}\|_{\mathbf{V}} = \langle \mathbf{V}^{-1}\boldsymbol{z}, \boldsymbol{z}\rangle^2, \quad \boldsymbol{z} \in \mathbb{R}^{d_y}.$$

We now detail some literature that tackles the solution to the filtering problem.

5.1.2 Existing algorithmic frameworks

In general, closed form solutions do not exist for these equations except for specific cases. The simplest such case occurs when the state and observation dynamics are linear and the prior beliefs on the state and the observation errors are Gaussian. In this case all distributions involved are Gaussian and the associated implementation is the celebrated Kalman (discrete time) or Kalman-Bucy (continuous times) filter [72, 73]. The Benes filter [9] is another analytic filter for problems where the nonlinear drift term satisfies a specific structure, which in one dimension is $\partial f/\partial x + f^2 = ax^2 + bx + c$.

Instead, numerical methods must be used to approximate the optimal solutions. Many of these numerical methods can be adapted from those that solve a related problem where the observations are also obtained in continuous time. In the discrete-time observation case, these methods default to solving the Fokker-Planck and Bayesian inference problems separately. One such method [81] is based upon the MCA method discussed in Chapter 3. Other methods that are based on the time discretization of Kushner's [77, 78] or Zakai's equation [128], describing the evolution of the normal-

ized and unnormalized distributions, respectively, are provided in [11, 30, 68]. Yet another method which seeks spectral solutions using Wiener chaos expansions can be found in [86]. While a great deal of other solution methods have been proposed, ones that allow scaling to large dimensions are still unavailable.

The numerical methods described above have fallen out of favor since they quickly encounter the curse of dimensionality. For this reason, the field has turned to developing particle or ensemble based algorithms that represent probability distributions with a collection of particles and Bayesian inference is performed with importance sampling. These algorithms are called particle filters (PF) [27, 41, 49, 100, 124]. Particles are easy to propagate through dynamics and to update with Bayes rule. However, the most general form of the PF suffers from particle degeneracy problems over long time frames, and it requires many particles to accurately capturing the probability distribution. For some problems the number of required particles scale exponentially with the number of dimensions and with the inverse variance of the observation log likelihood [110]. These types of problems can potentially be avoided in some situations where good proposals can be made [109].

As opposed to fully Bayesian filters, there exist a class of Gaussian filters which enforce all forecast and analysis distributions to be Gaussian. The earliest of these is based on linearization of nonlinear dynamics and is called the extended Kalman filter (EKF). Another family of Gaussian based filters is called the ensemble Kalman filter (EnKF) [21, 43] where similarly to the particle filter, particles are maintained. However, they differ from the PF because Gaussian distributions are assumed during the analysis steps, though some efforts to include some non-Gaussianity have recently been made proposed [83]. The EnKF is mainly used for high dimensional problems where maintaining many particles can be prohibitively expensive; fewer particles are needed when Gaussian assumptions are made. To truly scale the EnKF to higher dimensions more structure, such as tapering, must be added to the covariance matrix [46]. Hybrids between the EnKF and PF have also been created [45], they specify a parameter governing the spectrum between a full EnKF and a full PF that is automatically tuned in the filtering procedure.

Finally, a third family of Gaussian filters are quadrature/integration filters that includes the unscented Kalman filter (UKF) [71] the Gauss-Hermite (GH) Kalman filter [69], the cubature Kalman filter [1], and others [125]. These filters attempt to exactly calculate the means and covariances of the forecast and analysis steps of the Kalman filter through quadrature. The major difficulties with these filters is stability for high-order quadrature rules and scaling with dimension. For example, Wu [125] showed that only the GH filter has good stability properties as the dimension grows; however, it is also one of the only integration-based filters that encounters the curse of dimensionality. Many of the other integration-based Gaussian filters have better scaling properties but lose their stability in higher dimensions [125].

The integration formulation of Gaussian filtering makes it particularly attractive for the application of high-dimensional integration schemes. For this reason, we will explore the application of the FT in this context for retaining the good stability properties of Gaussian filtering without incurring exponential growth in computational expense.

5.1.3 Integration based Gaussian filtering equations

We now describe the equations for integration-based Gaussian filtering. In Section 5.2, we demonstrate how to take advantage of existing low-rank structure to solve these equations in a computationally efficient manner. For simplicity, we consider the case where the diffusion is only a function of time and not state, i.e., $\mathcal{D} : \mathscr{T} \to \mathbb{R}^{d \times d_w}$. Gaussian based filtering is an exact filter when the following assumptions hold.

Assumption 5 (Gaussian filtering). *Let filtering be performed for times $0 \leq t \leq T$. Let $M : [0, T] \to \mathbb{R}^d$ represent the mean and $C : [0, t] \to \mathbb{R}^{d \times d}$ represent the covariance of the filtered process $x(t)$. Gaussian filtering assumes that the initial distribution*

$$x(0) \sim \mathcal{N}(M(0), C(0)),$$

the forecast distributions for $k = 0, 1, \ldots$

$$x(t)|\boldsymbol{y}_{t_{k+1}} \sim \mathcal{N}(M(t), \mathcal{C}(t)), \quad t \in (t_k, t_{k+1}),$$

and the analyzed distributions

$$x(t_{k+1})|\boldsymbol{y}_{t_{k+2}} \sim \mathcal{N}(M(t_{k+1}), \mathcal{C}(t_{k+1})),$$

are all Gaussian.

Under Assumption 5, one can derive the following equations for the propagation of the mean and covariance [106].[1] Assume that no observations are obtained during the time interval $(t_k, t_{k+1}]$, then the forecast equations for the evolution of the mean and covariance are ordinary differential equations (ODEs)

$$\frac{dM}{dt} = \int_{\mathscr{X}} B(t,x) p_G(x; M(t), \mathcal{C}(t)) dx, \tag{5.7}$$

$$\frac{d\mathcal{C}}{dt} = \mathcal{D}(t)\mathcal{D}(t)^T + \mathbb{C}\text{ov}\left[x, B(t,x)\right] + \mathbb{C}\text{ov}\left[B(t,x), x\right], \tag{5.8}$$

where $p_G(x; M(t), \mathcal{C}(t))$ denotes the probability density of a Gaussian random variable with mean $M(t)$ and $\mathcal{C}(t)$, i.e.,

$$p_G(x; M(t), \mathcal{C}(t)) \propto \exp\left(-\frac{1}{2}\|x - M(t)\|_{\mathcal{C}(t)}\right),$$

and

$$\mathbb{C}\text{ov}\left[x, B(t,x)\right] = \int_{\mathscr{X}} \left(B(t,x) - \frac{dM}{dt}\right)(x - M(t))\, p_G(x; M(t), \mathcal{C}(t)) dx.$$

Denote $\boldsymbol{m}_{k+1}$ and $\mathbf{C}_{k+1}$ as the values of the mean and covariance following the

[1] Recall that for simplicity we are assuming, in this section, that the diffusion is not a function of the state, i.e., $\mathcal{D} : \mathscr{T} \to \mathbb{R}^{d \times d_y}$.

above integrating, i.e.,

$$\boldsymbol{m}_{k+1} = \int_{t_k}^{t_{k+1}} \frac{dM}{dt} dt,$$
$$\mathbf{C}_{k+1} = \int_{t_k}^{t_{k+1}} \frac{d\mathcal{C}}{dt} dt.$$

The analysis stage then requires updating these means and covariance using a newly obtained observation y_{k+1} at time t_{k+1}. This update results in the following equations for the mean and covariance of the filtered distribution

$$M(t_{k+1}) = \boldsymbol{m}_{k+1} + \mathbb{C}\text{ov}\left[x, H\left(t_{k+1}, x\right)\right]\left(\mathbb{C}\text{ov}\left[H\left(t_{k+1}, x\right), H\left(t_{k+1}, x\right)\right] + \mathbf{V}\right)^{-1}\left(y_{t_{k+1}} - \mu_{k+1}\right) \tag{5.9}$$

$$\begin{aligned}\mathcal{C}(t_{k+1}) = \mathbf{C}_{k+1} \\ - \mathbb{C}\text{ov}\left[x, H\left(t_{k+1}, x\right)\right]\left(\mathbb{C}\text{ov}\left[H\left(t_{k+1}, x\right), H\left(t_{k+1}, x\right)\right] + \mathbf{V}\right)^{-1}\mathbb{C}\text{ov}\left[H\left(t_{k+1}, x\right), x\right]\end{aligned} \tag{5.10}$$

where

$$\mu_{k+1} = \int_{\mathscr{X}} H(t_k, x) p_G(x; \boldsymbol{m}_{k+1}, \mathbf{C}_{k+1}) dx$$
$$\mathbb{C}\text{ov}\left[x, H\left(t_{k+1}, x\right)\right] = \int_{\mathscr{X}} (x - \boldsymbol{m}_{k+1})\left(H(t_{k+1}, x) - \mu_{k+1}\right) p_G(x; \boldsymbol{m}_{k+1}, \mathbf{C}_{k+1}) dx$$
$$\mathbb{C}\text{ov}\left[H\left(t_{k+1}, x\right), H\left(t_{k+1}, x\right)\right] =$$
$$\int_{\mathscr{X}} \left(H(t_{k+1}, x) - \mu_{k+1}\right)\left(H(t_{k+1}, x) - \mu_{k+1}\right) p_G(x; \boldsymbol{m}_{k+1}, \mathbf{C}_{k+1}) dx$$

The key point about these equations is that the computations required are integrals with respect to a Gaussian density. Thus, we will use our FT framework to perform these integrals by approximating the drift and observation operators in FT format. Secondly, Assumption 5 is typically not valid in practice, and therefore these equations become *approximations* for the solution of the filtering problem. Intuitively, the approximations incur less error as the distributions "become more Gaussian." This implies that in the cases where the data is frequent and informative or when the models are close to linear these equations may describe the filtered distribution accurately.

Furthermore, we note that while these assumptions may not yield an accurate approximation of the *distribution*, they often have good *tracking* performance. Indeed, much of the filtering literature only even measures the quality of data assimilation algorithms through tracking performance.

5.2 Low-rank filtering algorithms

In this section, we describe how to apply low-rank integration algorithms for the evaluation of the integrals in Section 5.1.3. In order to use low-rank tensor-based algorithms for integration, the integration must be performed with respect to a tensor-product measure. However, the relevant equations, as presented, were all integrals with respect to a correlated density $p_G(x; \boldsymbol{m}, \mathbf{C})$ for some mean $\boldsymbol{m}$ and non-diagonal covariance $\mathbf{C}$. To this end, a variable transformation $x \to \nu$ must be performed before integration. The new random variable $\nu = (\nu_1, \ldots, \nu_d) \in \mathbb{R}^d$ has a standard multivariate normal distribution, i.e., $\nu \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. In the case of correlated Gaussians, the mapping between these variables is linear and given by

$$x = \boldsymbol{m} + \mathbf{C}^{1/2}\nu.$$

Using this mapping, we define a new drift term $\hat{B}(t, \nu)$ as

$$\hat{B}(t, \nu) = B(t, M(t) + \mathcal{C}(t)^{1/2}\nu),$$

so that the right hand side of (5.7) becomes

$$\begin{aligned} \frac{dM}{dt} &= \int B(t, x) p_G(x; M(t), \mathcal{C}(t)) dx \\ &= \int \hat{B}(t, \nu) p_G(\nu; \mathbf{0}, \mathbf{I}) d\nu. \end{aligned} \tag{5.11}$$

Thus, the first step involved with evaluating the right-hand-side of this ODE, in the context of time time integration schemes, is transforming the equation to one with independent variables. Similar mappings can be used for the integrals required for

the propagation of the covariance in (5.8) and for those involved in the Bayes update steps (5.9) and (5.10).

Next, an integration scheme is devised for $\hat{B}(t,\nu)$. We consider two schemes, the first is low-rank tensor product quadrature and the second is FT integration.

5.2.1 Low-rank tensor product quadrature

Since, the integrals of interest are with respect to a Gaussian measure, low-rank tensor-product quadrature requires specifying a Gaussian-Hermite quadrature rule for each dimension. Suppose that dimension i uses n_i quadrature points and weights $\{(\nu_i^1, w_i^1)\dots(\nu_i^{n_i}, w_i^{n_i})\}$. Next, suppose that *every output* of $\hat{B}(t,\nu)$ is discretized onto the tensor product grid defined by a tensorization of the Gaussian quadrature rules. Let $\boldsymbol{\mathcal{B}}^{(k)} \in \mathbb{R}^{n_1\times\cdots\times n_d}$ denote the kth output of the drift evaluated on the grid at a fixed t, i.e.,

$$\boldsymbol{\mathcal{B}}^{(k)}[i_1,\dots,i_d] = \hat{B}[k](t,\{\nu_1^{i_1},\dots,\nu_d^{i_d}\})$$

Then the approximation of the integral for the k-th output is

$$\int \hat{B}[k](t,\nu)p_G(\nu,\mathbf{0},\mathbf{I})d\nu \approx \sum_{i_1}^{n_1}\dots\sum_{i_d}^{n_d}\boldsymbol{\mathcal{B}}^{(k)}[i_1,\dots,i_d]w_1^{i_1}\dots w_d^{i_d}.$$

Clearly, such a computation scales exponentially with dimension. Thus, one first decomposes $\boldsymbol{\mathcal{B}}^{(k)}$ into its TT format, and then performs d tensor-vector contractions with $\mathcal{O}(dnr^2)$ operations [95]. Since d such operations are required, one for each dimension, the total cost for one evaluation of the equations for the evolution of the mean is $\mathcal{O}(d^2nr^2)$. The dominating cost is computing the TT decomposition of $\hat{B}[k]$ for each $k = 1,\dots,d$, and it requires a total of $\mathcal{O}(d^2nr^3)$ operations.

This strategy can be employed for computing integrals for $\mathbb{C}\mathrm{ov}(B,x)$, $\mathbb{C}\mathrm{ov}(H,x)$, and $\mathbb{C}\mathrm{ov}(H,H)$. However, determining a good number of quadrature nodes is still a challenging problem. The continuous tensor-train of Chapter 3 alleviates this problem through fiber level adaptation.

5.2.2 FT-based integration filter

Generally, the use of the function-train within this integration task is straightforward. One first generates an FT approximation of the integrand, then one integrates the approximation according to integration scheme described in Section 3.6.1. As with the low-rank tensor product integration just described, integrating vector-valued or matrix-valued functions requires approximating each of the outputs separately.

The major difference between the integration for Gaussian filtering and the numerous integration examples described in Chapter 3 is that the integration we are considering here is respect to a Gaussian measure. To this end, we need the FT cores to consist of scalar-valued functions that can be integrated with respect to the Gaussian measure so that we can use the standard integration algorithm from Section 3.6.1. Therefore, the specific implementation of the continuous framework we use represents each univariate function in an expansion of Hermite polynomials $\{\mathrm{He}_1, \mathrm{He}_2, \ldots\}$ where

$$\int_{\mathbb{R}} \mathrm{He}_i(s)\,\mathrm{He}_j(s)\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}s^2}ds = \delta_{i,j}$$

Thus, when we build an FT representation of some function $f(\nu)$, the $k-$th core will consist of the scalar-valued functions

$$f_k^{(i,j)}(\nu_k) = \sum_{\ell=1}^{P} a_{i,j,\ell}^k\,\mathrm{He}_\ell(\nu_k), \quad \text{for } i = 1,\ldots,r_{k-1},\ j = 1,\ldots,r_k, \text{ and } a_{i,j,\ell} \in \mathbb{R},$$

such that

$$\int f_k^{(i,j)}(\nu_k)\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}{\nu_k}^2}d\nu_k = a_{i,j,1}^k.$$

In summary, every time we encounter an integral during the forecasting or analysis steps of the integration based Gaussian filter we perform three steps:

1. Map the dependent integration variables to a set of independent variables

2. Build an FT approximation of the integrand using Hermite polynomials expansions for univariate functions

3. Integrate the FT approximation using the scheme in Section 3.6.1

This scheme has an advantage over the tensor product quadrature scheme due to its automated adaptation of each fiber.

Note that the theorems and matrix-factorizations presented in Chapters 2 and 3 apply to functions defined on the unit interval with a uniform measure. Here, we are considering functions defined on the entire real line with a Gaussian measure. Two practical changes must be made within the rank-adaptive cross approximation algorithm of Section 3.3.2 to account for these differences. First, whenever an inner product is performed, for example within a QR decomposition, the inner product is weighted according to the Gaussian measure. Second, maximization of a univariate function, for example within an LU or `maxvol` computation, is replaced by maximization of the function multiplied by the density.

5.3 Numerical examples

We now present three examples of low-rank integration-based Gaussian filtering. In the first example, we will use a tensorized Gauss-Hermite quadrature rule as described in Section 5.2.1, and in the second two examples we will use the FT-based integration scheme described in Section 5.2.2.

The first example is a model for a re-entry vehicle [106]. The state space is five dimensional and the dynamics are nonlinear

$$
\begin{aligned}
dx_1 &= x_3(t)dt \\
dx_2 &= x_4(t)dt \\
dx_3 &= d(t)x_3(t)dt + g(t)x_1(t)dt + \sigma_1 dw_3(t) \\
dx_4 &= d(t)x_4(t)dt + g(t)x_2(t)dt + \sigma_2 dw_4(t) \\
dx_5 &= \sigma_3 dw_5(t),
\end{aligned}
$$

where

$$
\begin{aligned}
r(t) &= \sqrt{x_1^2(t) + x_2^2(t)} \\
v(t) &= \sqrt{x_3^2(t) + x_4^2(t)} \\
b(t) &= b_0 \exp(x_5(t)) \\
d(t) &= b(t) \exp\left(\frac{r_0 - r(t)}{H_0}\right) v(t) \\
g(t) &= -\frac{Gm_0}{r^3(t)}.
\end{aligned}
$$

where G is the Gravitational constant and m_0 is the mass such that $Gm_0 = 3.986 \times 10^5$, $r_0 = 6374$ is the radius, and finally $b_0 = -0.59783$ and $H_0 = 13.406$ are constants required for the computation of the drag $d(t)$. The diffusion magnitudes are $\sigma_1 = \sigma_2 = 2.4064 \times 10^{-4}$ and $\sigma_3 = 10^{-6}$. The observations are two dimensional and the observation operator is also nonlinear

$$
\begin{aligned}
r_k &= \sqrt{(x_1(t_k) - r_0)^2 + (x_2(t_k))^2} + \eta_1, \quad \eta_1 \sim \mathcal{N}(0,1) \\
\theta_k &= \tan^{-1}\left(\frac{x_2(t_k)}{x_1(t_k) - r_0}\right) + \eta_2, \quad \eta_2 \sim \mathcal{N}(0,1)
\end{aligned}
$$

The dynamics are integrated for 100 seconds and observations are obtained every 3 seconds. Time integration was performed with the DOPRI5 integrator in the SciPy package [70] for Python. The purpose of this experiment is to show that the TT-based low-rank tensor-quadrature integration provides virtually equivalent results to full tensor quadrature integration. Furthermore, for the Gauss-Hermite filter (GHF) the scaling with the number of quadrature nodes should be polynomial whereas for the TT-based low-rank quadrature the scaling should be linear. Both effects are seen in Figure 5-1.

For the rest of the examples we perform integration with the FT based integration scheme. Our second example is the chaotic Lorenz 1963 three dimensional dynamics

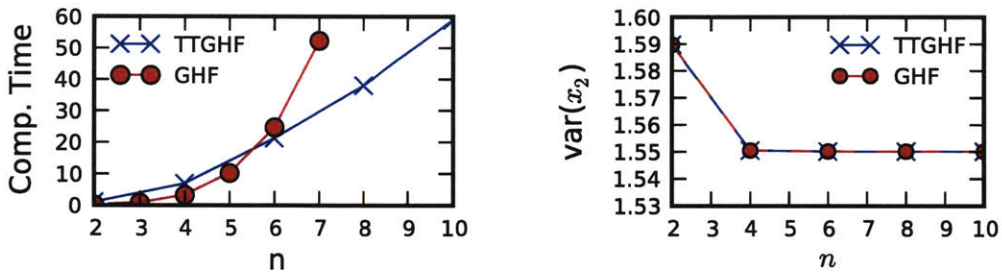


Figure 5-1: Comparison of computational time and variance obtained between fully tensorized Gauss-Hermite filter (GHF) and the low-rank tensor-train GHF (TTGHF).

with a radar-type nonlinear observation.

$$
\begin{aligned}
dx_1 &= r_1(-x_1 + x_1) + \sigma dw_1(t) \\
dx_2 &= r_2 x_1 - x_1 x_3 + \sigma dw_2(t) \\
dx_3 &= -r_3 x_3 + x_1 x_2 + \sigma dw_3(t) \\
y_k &= \sqrt{\left(x_1 - \frac{1}{2}\right)^2 + x_2^2 + x_3^2} + \xi_k, \quad \xi_k \sim \mathcal{N}(0,1)
\end{aligned}
$$

with $r_1 = 10$, $r_2 = 28$, $r_3 = \frac{8}{3}$, $\sigma = 10^1$, RKF45 adaptive time-integration is used. Observations are obtained with a time step of $\Delta t = 0.5$s, i.e., observation times are $t_k = 10\Delta t$. The dynamics are at most rank 3 due to quadratic terms resulting from a transformation of x to ν (the rank of a quadratic is at most $d/2$). Figure 5-2 shows the results of the filtering. The lower right panel shows the observations obtained using the black dots and the trajectory of a hypothetical continuous observation operator using the dashed line. First, we see that there is an initial "burn-in" time for the state to reach its attractor. During this burn-in we see rapid increases in the uncertainty since the dynamics quickly push the state towards the attractor. Next, note that the mean tracks the true system state for the entire duration. Furthermore, the true system state is within the uncertainty bounds at all times.

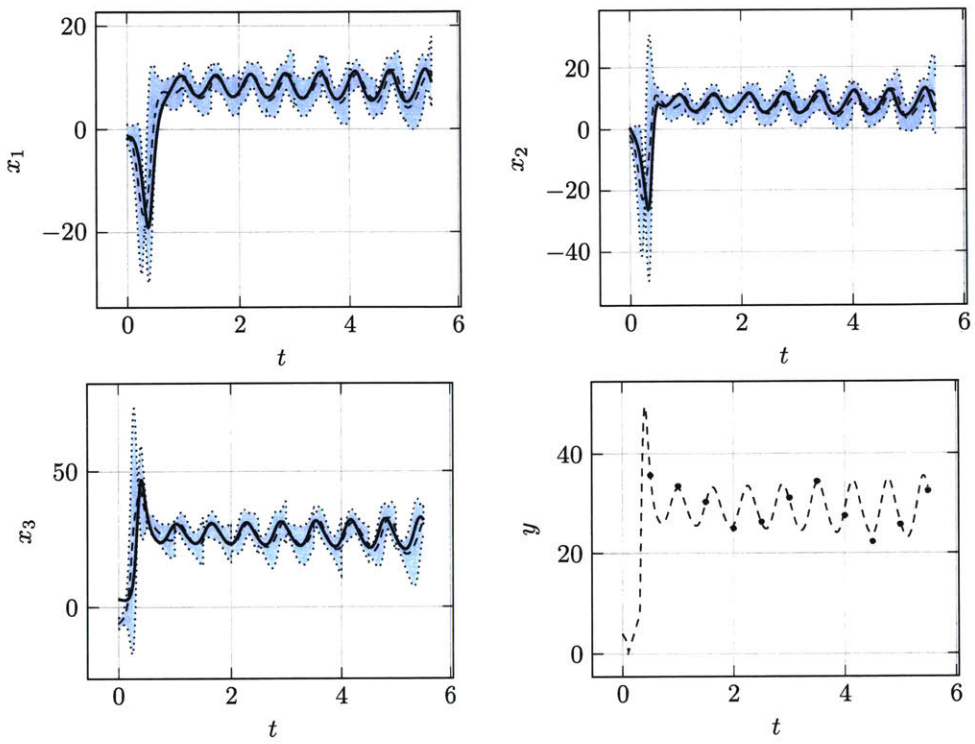


Figure 5-2: Filtering of the Lorenz 63 system. Uncertainty is represented with the shaded region and the mean estimate is represented by the dashed line in all but the bottom right panel. The true system trajectory is given by the solid black line. The bottom right panel shows the realized data (black dots) and a hypothetical observation trajectory (dashed line).

The final example is a 20 dimensional chaotic Lorenz 1996 system given by

$$
\begin{aligned}
dx_i &= (x_{i+1} - x_{i-2})x_{i-1} - x_i + F + \sigma dw_i(t), \qquad \text{for } i = 1, \ldots, 20 \\
x_0 &= x_d \\
x_{-1} &= x_{d-1} \\
x_{d+1} &= x_1 \\
y_k^j &= x_{2j-1}(t_k) + \xi_k, \quad \xi_k \sim \mathcal{N}(0,1), \qquad \text{for } j = 1, \ldots 10
\end{aligned}
$$

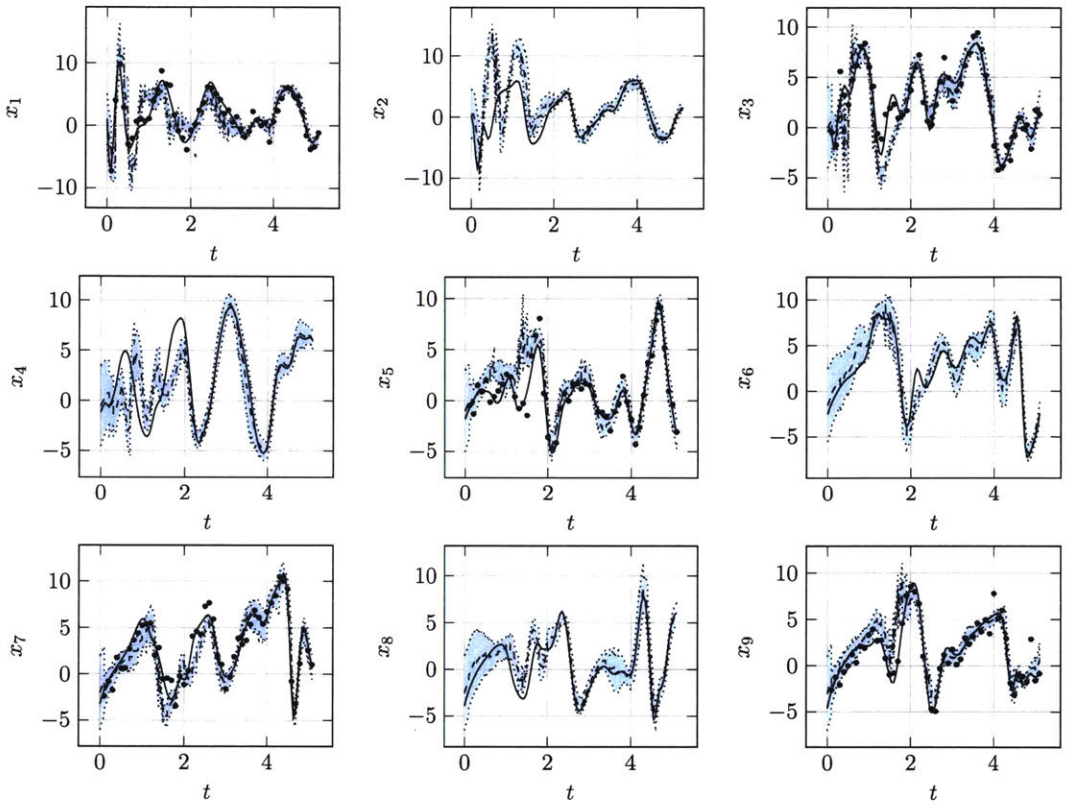


Figure 5-3: Traces of the mean (dashed line), two times standard deviation (shaded region), observations (black-dots) and system trajectory (solid line) for specific system states for the Lorenz96 chaotic system.

We use $F = 8$ to stay within the chaotic regime and fix the diffusion to $\sigma = 0.25$. Integration is performed with an adaptive RKF45 scheme. Observations of every other variable are taken everything $\Delta t = 0.1$ seconds. The resulting traces for a subset of the state variables is provided in Figure 5-3. Figure 5-3 shows that the mean tracks the true system state for both the observed and unobserved variables, but the uncertainty is noticeably smaller for the observed variables. Furthermore, the tracking and uncertainty are noticeably worse when $t < 2$ for many of the variables. This characteristic is due to the fact that no burn-in was used before filtering is started. Furthermore, this fact is actually a testament to the stability of the FT approximation scheme for the Gaussian filtering since the state does not diverge, and

good tracking as well as a good description of uncertainty is eventually obtained.

Chapter 6

Conclusion

6.1 Summary

A framework for computation with multivariate functions was introduced and applied to stochastic optimal control and to data assimilation. The new set of tools can be used to encode adaptive, efficient, and accurate numerical algorithms for computation with functions. The research primarily builds upon two areas: tensor decompositions and function approximation. Tensor decompositions mitigate the curse of dimensionality by discovering low-rank structure, and function approximation is used to accurately compute with functions.

The application of this low-rank computational framework to stochastic optimal control has provided us with an opportunity to solve higher dimensional problems than previously possible. For example, we are able to extend the effectiveness of value and policy iteration to higher dimensional systems. We envision that since stochastic optimal control is such a general and widespread problem formulation, that these tools can positively affect many domains.

The application of the framework to integration-based Gaussian filtering has also pushed the limits of a methodology that is often discounted for high dimensional problems. Indeed, while the requirement of accurate integration-based methods for propagating the mean and covariance of a distribution through a nonlinear dynamical system was once viewed as a hinderance to high dimensional computation, we have

shown that integration algorithms that leverage low-rank structure can prove effective. Thus, this research has provided an avenue for further work that exploits the stability of integration based Gaussian filtering within higher dimensional applications.

Now, to restate our contributions, those at a high level include

1. Development of a framework for continuous computation with multivariate functions by leveraging low-rank tensor decompositions,

2. Application of a low-rank framework for the solution of dynamic programming equations arising in stochastic optimal control, and

3. Application of continuous computation to integration-based Gaussian filtering.

Contributions to continuous computation include

1. Development of maximum-volume based CUR/skeleton decompositions of vector-valued functions,

2. Extension of continuous matrix factorizations to the case of QR and LU factorization of matrix-valued functions, and

3. Continuous versions of cross approximation, rounding, and alternating least squares for the function-train.

Contributions to stochastic optimal control and data assimilation include

1. Utilization of the function-train decomposition within value and policy iteration for solving Bellman's equation,

2. Utilization of the function-train decomposition for multi-level schemes by enabling evaluations of optimal value functions and policies in continuous domains, and

3. Utilization of the function-train for multivariate integration within integration-based Gaussian filtering.

6.2 Future work

While this research has formulated and shown the advantages of the continuous computational framework, there are many ways in which this work can be extended.

Linear solvers are one of the pillars of linear algebra, and we would like to develop corresponding continuous linear solvers that deal with continuous linear operators. The primary motivation of solving linear systems is to be able to solve partial differential equations encountered within uncertainty quantification and optimal control. Indeed, many low-rank solvers have complemented the recent development of tensor-decompositions [37, 38, 97, 116]. Furthermore, continuous algorithms have also been developed for solving certain PDEs for univariate and bivariate systems [93]. Bridging these methods has the potential to allow adaptive and efficient solutions to high dimensional PDEs.

Another direction of interest, particularly for machine learning/data science applications, is the development of low-rank data fitting algorithms. Strides have been made to accomplish this task, for example see [40] for data fitting with the canonical decomposition, using alternating least squares techniques. It would be interesting and beneficial to tackle these problems with the continuous alternating least squares algorithm that incorporates rank adaptivity discussed in Chapter 3. "Big data" problems also pose an interesting data fitting application since tensor methods may also be used to first compress the data before creating data fitting models.

There are many directions to take low-rank computation within control as well. Linear temporal logic and differential games are both areas that borrow heavily from the stochastic optimal control formulation and can be can be reframed in terms of dynamic programming equations. It will be useful to exploit low-rank structure in these situations to enable the solution of higher dimensional problems.

Finally, it would be interesting to utilize low-rank continuous algorithms within the context of challenging Bayesian inverse problems. For example, one can interpret Bayes rule as a simple multiplication of two functions, the prior and the likelihood. Both of these functions can potentially contain low rank structure that can be

discovered with the framework and algorithms that we have developed. The resulting deterministic algorithms can potentially be highly advantageous over the current state-of-the-art sampling methods.

Appendix A

Additional lemmas and proofs

A.1 Proof of Theorem 2

Proof. Let us denote $E \equiv F - C[\bar{\mathbf{F}}^{*,k}]^{-1}R$, such that $E : \mathcal{X} \to \mathbb{R}^n$. Let $\boldsymbol{\alpha} = [(i_1, z_1), \ldots (i_\ell, z_\ell)]$ and $\boldsymbol{y} = [y_1, \ldots, y_\ell]$ denote the fibers used to construct $\bar{\mathbf{F}}^{*,k}$. Now, we seek to bound the absolute value of E for each $x \in \mathcal{X}$. Consider any $(\ell+1)\times(\ell+1)$ sub-matrix of F formed by augmenting $\bar{\mathbf{F}}^{*,k}$ by one row and column of values defined by $x_{>k} \in \mathcal{X}_{>k}$ and $(i, x_{\leq k}) \in \{1, \ldots, n\} \times \mathcal{X}_{\leq k}$ such that

$$\mathbf{F} = \begin{bmatrix} \bar{\mathbf{F}}^{*,k} & \boldsymbol{b} \\ \boldsymbol{c}^T & a \end{bmatrix},$$

where each choice of $x_{>k}$ maps to a vector $\boldsymbol{b}$ according to $\boldsymbol{b}[j] = F[i_j](z_j, x_{>k})$ for $j = 1 \ldots \ell$. Relatedly, each choice of $(i, x_{\leq k})$ maps to a vector $\boldsymbol{c}$ according to $\boldsymbol{c}[j] = F[i](x_{\leq k}, y_j)$. The values for a analogously correspond to a selection of $(i, x_{\leq k}, x_{>k})$. Now define $\gamma = a - \boldsymbol{c}^T[\bar{\mathbf{F}}^{*,k}]^{-1}\boldsymbol{b}$ and note that γ corresponds to the value of E obtained by the combination $(i, x_{\leq k}, x_{>k})$ chosen to augment $\bar{\mathbf{F}}^{*,k}$. Also, $\gamma = 0$ iff $\mathbf{F}$ is singular due to the following property of the determinant,

$$\det\left(\begin{bmatrix} \bar{\mathbf{F}}^{*,k} & \boldsymbol{b} \\ \boldsymbol{c}^T & a \end{bmatrix}\right) = \det(\bar{\mathbf{F}}^{*,k}) \det(a - \boldsymbol{c}^T[\bar{\mathbf{F}}^{*,k}]^{-1}\boldsymbol{b}) \tag{A.1}$$

for any vectors $\boldsymbol{b}, \boldsymbol{c}$ and scalar a as long as $\bar{\mathbf{F}}^{*,k}$ is invertible. If $\gamma = 0$, then the value of E to which it refers is zero.

Now we consider the case where $\gamma \neq 0$ and that $\mathbf{F}$ is non-singular. Using property (A.1), we see

$$|\det(\mathbf{F})| = |\det(\bar{\mathbf{F}}^{*,k})||\gamma|$$

Furthermore, recall that we can write the inverse of a matrix using the matrix of cofactors $\mathbf{C}$

$$\mathbf{F}^{-1} = \frac{1}{\det(\mathbf{F})}\mathbf{C}^T \tag{A.2}$$

The maximal volume property of $\bar{\mathbf{F}}^{*,k}$ now implies that $||\mathbf{C}^T||_C = |\det(\bar{\mathbf{F}}^{*,k})|$. Together with (A.2) this implies

$$||\mathbf{F}^{-1}||_C = \frac{1}{|\det(\mathbf{F})|}|\det(\bar{\mathbf{F}}^{*,k})| = |\gamma^{-1}|. \tag{A.3}$$

From here, we seek to obtain an upper bound for $|\gamma|$ because this corresponds to the maximum value of E corresponding choices of $(i, x_{\leq k}, x_{>k})$. Therefore, we would like to obtain a lower bound for $||\mathbf{F}^{-1}||_C$. We can obtain this bound by starting with the Frobenius norm on $\mathbf{F}$ to obtain a bound using its singular values

$$||\mathbf{F}^{-1}||_F^2 = \sum_{i=1}^{\ell+1} \sigma_i^{-2}(\mathbf{F}) \leq \|\mathbf{F}^{-1}\|_C^2 \, (\ell+1)^2 \,,$$

and since $\sigma_{\ell+1}^{-2}(\mathbf{F}) \leq \sum_{i=1}^{\ell+1} \sigma_i^{-2}(\mathbf{F})$ we have $\sigma_{\ell+1}^{-2}(\mathbf{F}) \leq ||\mathbf{F}^{-1}||_C^2(\ell+1)^2$. Finally, we obtain

$$||\mathbf{F}^{-1}||_C > \frac{1}{\sigma_{\ell+1}(\mathbf{F})(\ell+1)}$$

Now using (A.3) we have $|\gamma| \leq \sigma_{\ell+1}(\mathbf{F})(\ell+1)$. From Lemma 5 in Appendix A.2 we can relate the singular values of the matrix $\mathbf{F}$ to the singular values of F to obtain the inequality

$$\sigma_{\ell+1}(\mathbf{F}) \leq M\rho^2(\ell+1)^{2\varepsilon}\sigma_{\ell+1}(F),$$

for $\varepsilon > 1/2$. This inequality provides us with the final result. □

A.2 Eigenvalues of a discretized kernel

Recall that we seek to bound the smallest singular value $\sigma_m(\mathbf{F})$ of a sub-matrix $\mathbf{F} \in \mathbb{R}^{m \times m}$ of a vector-valued function by the corresponding singular values of the continuous vector-valued function $\sigma_m(F)$. The sub-matrix $\mathbf{F}$ can be viewed as a discretized version of F. We use this result in the proof of Theorem 2 where $m = \ell + 1$ and ℓ is the rank of the CUR decomposition of interest.

This bound arises from the result in [90] where one considers a compact linear integral operator K with kernel $k(s,t)$ defined as

$$(Kf)(s) \equiv \int_{\Omega} k(s,t) f(t) dt, \quad s \in \Omega \tag{A.4}$$

In particular one can define a discretization of k using ℓ values $[s_1, \ldots, s_\ell]$ such that $\mathbf{K}[i,j] = k(s_i, s_j)$. Then one can bound the eigenvalues of $\mathbf{K}$ by the corresponding eigenvalues of K. The result is summarized in the lemma below.

Lemma 5 (Interlacing eigenvalues of discretized operator [90]). *Let $K : L^2(\Omega) \to L^2(\Omega)$ be a symmetric, positive definite, compact linear integral operator with continuous kernel $k(s,t)$ that has eigendecomposition (from Mercer's theorem)*

$$k(s,t) = \sum_{p=1}^{\infty} \lambda_p \Psi_p(s) \Psi_p(t),$$

where $\{\lambda_p\}_{p=1}^{\infty}$ are eigenvalues and the eigenfunctions $\{\Psi_p(t)\}_{p=1}^{\infty}$ form an orthonormal basis for $L^2(\Omega)$. Furthermore, assume that the eigenfunctions are uniformly bounded: $|\Psi_p(s)| \leq \rho$ for all p and s. Let $\mathbf{K} \in \mathbb{R}^{\ell \times \ell}$ denote a discretized version of k. Then it holds that

$$\lambda_p(\mathbf{K}) \leq |\Omega| \rho^2 \zeta(2\epsilon) p^{2\varepsilon} \lambda_p(k), \quad p < \ell, \quad \text{for any } \varepsilon > 1/2 \tag{A.5}$$

where ζ is the Riemann zeta function, $\zeta(2\varepsilon) = \sum_{p=1}^{\infty} \frac{1}{p^{2\varepsilon}}$.

If $\mathbf{F}$ is the discretization of a continuous but non-symmetric kernel F, and the singular functions of the corresponding integral operator are uniformly bounded, then an analogous result for the singular values of $\mathbf{F}$ can be shown to hold.

A.3 Proof of Theorem 3

The proof of Theorem 3 is the continuous analogue of the corresponding theorem by Oseledets [95].

Proof. We start with unfolding function f^1. Since its rank is r_1, the function can be decomposed as

$$f^1(x_1, x_{>1}) = U_1(x_1)V_1(x_{>1}), \tag{A.6}$$

where $U_1 = [u_1^{(1)}, \ldots, u_r^{(1)}]$ is a vector-valued function with orthogonal columns and $V_1 = [v_1^{(1)}, \ldots, v_r^{(1)}]$ is a vector-valued function with orthonormal columns. This type of decomposition can arise from the functional SVD (2.4) in which the singular values have been absorbed into U_1.

Using these orthogonality conditions we can define each $v_i^{(1)}$ as

$$v_i^{(1)}(x_{>1}) = \frac{\left\langle f^1(s, x_{>1}), u_i^{(1)}(s) \right\rangle}{\left\langle u_i^{(1)}(s), u_i^{(1)}(s) \right\rangle} = \left\langle f^1(s, x_{>1}), w_i^{(1)}(s) \right\rangle \tag{A.7}$$

If we consider the unfoldings of each right singular function

$$\left[(v_i^{(1)})^k\right] = v_i^{(1)}(\{x_2, \ldots x_k\}, \{x_{k+1}, \ldots, x_d\}),$$

for $k = 2, \ldots, d-1$, then we we can show that both rank $\left[(v_i^{(1)})^k\right] \leq r_k$ for $k = 2, \ldots, d-1$ and that $(V_1)^k$ is also at most rank k according to the extended SVD definition (2.5).

We start by noting that unfolding k (or separated form f^k of f) can be decomposed into

$$f^k(x_{\leq k}, x_{>k}) = F(x_{\leq k})G(x_{>k}), \tag{A.8}$$

where F and G are both vector-valued functions consisting of r_k functions. Now we plug these results into the definition of the k-th unfolding of $v_i^{(1)}$

$$\begin{aligned}(v_i^{(1)})^k &= v_i^{(1)}(\{x_2, \dots x_k\}, \{x_{k+1}, \dots, x_d\}) \\ &= \left\langle f^1(s, x_{>1}), w_i^{(1)}(s) \right\rangle \\ &= \left\langle F(s, x_2, \dots, x_k) G(x_{>k}), w_i^{(1)}(s) \right\rangle \\ &= \left\langle F(s, x_2, \dots, x_k), w_i^{(1)}(s) \right\rangle G(x_{>k}) \\ &= H_i(x_2, \dots x_k) G(x_{>k}), \end{aligned} \tag{A.9}$$

where $H_i(x_2, \dots, x_k) = \langle F(s, x_2, \dots, x_k), w_i(s) \rangle$ is also a vector-valued function consisting of r_k functions such that

$$H_i[j](x_2, \dots, x_k) = \left\langle F[j](s, x_2, \dots, x_k), w_i^{(1)}(s) \right\rangle \quad j = 1 \dots r_k. \tag{A.10}$$

We have shown that unfolding k of $v^{(1)}$ is a rank r_k function, i.e.

$$\text{rank} \left[(v_i^{(1)})^k \right] \leq r_k \tag{A.11}$$

Furthermore, we can see that if we create a matrix-valued function whose rows are H_i, $\mathcal{H} = [H_1, H_2, \dots H_{r_1}]$ then the quasimatrix V_1 has an extended SVD, also of rank r_k,

$$V_1(x_{>1})^k = \mathcal{H}(x_2, \dots, x_k) G(x_{>k}) \tag{A.12}$$

Now we can proceed recursively and use the extended SVD of a quasimatrix to separate x_2 from the other dimensions $x_{>2}$ in V_1

$$[V_1(x_{>1})]^{k=2} = V_1(x_2, x_{>2}) = \mathcal{U}_2(x_2) V_2(x_{>2}), \tag{A.13}$$

where $\mathcal{U}_2$ is a $r_1 \times r_2$ matrix-valued function with orthogonal columns and V_2 is a quasimatrix consisting r_2 orthonormal functions. From here we see that $\mathcal{F}_1 = U_1$, $\mathcal{F}_2 = \mathcal{U}_2$, and we can proceed with V_2 as we did on V_1 to obtain the rest of the cores.

□

A.4 Proof of Theorem 4

The proof is a continuous analogue of that provided by Oseledets [98].

Proof. The case $d = 2$ follows from the definition of the functional SVD. Now we consider the case $d > 2$, the first unfolding can be decomposed into

$$f^1(x_1, x_{>1}) = U_1(x_1)V_1(x_{>1}) + e^1(x_1, x_{>1})$$

where $U_1 : \mathscr{X}_1 \to \mathbb{R}^{r_1}$ consist of the first r left singular functions of f^1, and $V_1 : \mathscr{X}_{>1} \to \mathbb{R}^{r_1}$ consists of the first r right-singular functions of f^1. Further note that due to the properties of the SVD we have that $u_i^{(1)} = U_1[i]$ satisfy $\left\langle u_i^{(1)}(s), e^1(s, x_{>1}) \right\rangle = 0$ for $i = 1 \dots r_1$.

V_1 is a multivariate vector-valued function, and as such, also potentially has a low-rank extended SVD. Suppose that we generate a rank r_2 approximation $\hat{V}_1$ of V_1. Then if we define an approximation to f as $\hat{f}(x_1, x_{>1}) = U_1(x_1)\widehat{V}_1(x_{>1})$, we can bound its error according to

$$\begin{aligned}
\int \left(f - \hat{f}\right)^2 dx &= \int \left(f^1 - U_1\widehat{V}_1\right)^2 dx \\
&= \int \left(f^1 - U_1\left(\widehat{V}_1 + V_1 - V_1\right)\right)^2 dx \\
&\leq \int \left(f^1 - U_1V_1\right)^2 dx + \int \left(U_1\left(V_1 - \widehat{V}_1\right)\right)^2 dx \\
&\leq \varepsilon_1^2 + \left\langle V_1 - \widehat{V}_1, V_1 - \widehat{V}_1 \right\rangle.
\end{aligned}$$

We can now repeat the process for V_1 to bound the error between V_1 and $\widehat{V}_1$. Suppose that the $k = 2$ unfolding of V_1, $V_1^2(x_2, x_{>2})$, has a rank r_2 extended SVD

$\mathcal{U}_2(x_2)V_2(x_{>2})$ such that

$$\|V_1^2(x_2, x_{>2}) - \mathcal{U}_2(x_2)V_2(x_{>2})\| \leq \varepsilon_2. \tag{A.14}$$

where $\mathcal{U}_2 : \mathcal{X}_2 \to \mathbb{R}^{r_1 \times r_2}$ and $V_2 : \mathcal{X}_{>2} \to \mathbb{R}^{r_2}$. We will shortly show that this fact follows from the approximate low-rank nature of the unfolding functions f^k. For now, assume that V_2 has the corresponding property that we can approximate it with a rank r_3 expansion $\widehat{V}_2$ to obtain an explicit expansion for $\widehat{V}_1$ as

$$\widehat{V}_1(x_{>1}) = \mathcal{U}_2(x_2)\widehat{V}_2(x_{>2}), \tag{A.15}$$

Using (A.15) for $\widehat{V}_1$ we obtain

$$\begin{aligned}
\left\langle V_1 - \widehat{V}_1, V_1 - \hat{V}_1 \right\rangle &= \left\langle V_1 - \mathcal{U}_2\widehat{V}_2, V_1 - \mathcal{U}_2\widehat{V}_2 \right\rangle \\
&= \left\langle V_1 - \mathcal{U}_2(\widehat{V}_2 + V_2 - V_2), V_1 - \mathcal{U}_2(\widehat{V}_2 + V_2 - V_2) \right\rangle \\
&\leq \varepsilon_2^2 + \left\langle V_2 - \widehat{V}_2, V_2 - \widehat{V}_2 \right\rangle
\end{aligned}$$

Now, if we assume that V_ℓ can be approximated with a rank $r_{\ell+1}$ function $\mathcal{U}_{\ell+1}V_{\ell+1}$, and therefore $V_{\ell+1}$ can be approximated by a rank $r_{\ell+2}$ function $\widehat{V}_{\ell+1}$, the above argument can proceed recursively to obtain

$$\left\langle V_1 - \widehat{V}_1, V_1 - \widehat{V}_1 \right\rangle \leq \sum_{\ell=2}^{d-1} \varepsilon_\ell^2,$$

which yields the desired result.

It remains to show that the rank $r_{\ell+1}$ extended SVDs of V_ℓ have errors smaller than $\varepsilon_{\ell+1}$, and therefore the errors resulting from using finite rank approximations $\widehat{V}_\ell$ are correspondingly bounded bounded. We show these facts using similar arguments to those in Theorem 3 (see (A.9)). We begin by showing that the $k = 2, \ldots, d-1$ unfoldings of V_1 are also of extended SVD rank r_k within ε_k. Recall that in (A.7) we

represented each scalar-valued function making up V_1 as

$$v_i^{(1)}(x_{>1}) = \left\langle f^1(s, x_{>1}), w_i^{(1)}(s) \right\rangle, \tag{A.16}$$

and then we used the unfoldings of f to prove that the unfoldings of $v_i^{(1)}$ are at most r_k. Similarly, if we now consider that the unfoldings of f can be represented as

$$f^k(x_{\leq k}, x_{>k}) = F(x_{\leq k})G(x_{>k}) + e^k, \tag{A.17}$$

such that the first term is a rank r_k expansion and $\|e^k\| \leq \varepsilon_k$ then for each unfolding $k = 2, \ldots, d$ we have

$$\begin{aligned}
(v_i^{(1)})^k &= v_i^{(1)}(\{x_2, \ldots x_k\}, \{x_{k+1}, \ldots, x_d\}) \\
&= \left\langle f(s, x_{>1}), w_i^{(1)}(s) \right\rangle \\
&= \left\langle F(s, x_2, \ldots, x_k)G(x_{>k}) + e^k(s, x_2, \ldots, x_k), w_i^{(1)}(s) \right\rangle \\
&= \left\langle F(s, x_2, \ldots, x_k), w_i^{(1)}(s) \right\rangle G(x_{>k}) + \left\langle e^k(s, x_2, \ldots, x_k), w_i^{(1)}(s) \right\rangle \\
&= H_i(x_2, \ldots x_k)G(x_{>k}) + \hat{e}^k,
\end{aligned}$$

where $H_i(x_2, \ldots, x_k) = \langle F(s, x_2, \ldots, x_k), w_i(s) \rangle$ is also a vector-valued function consisting of r_k functions and $\|\hat{e}^k\| \leq \varepsilon_k$ since the contraction of e^k with respect to an orthonormal function $w_i^{(1)}(s)$ cannot increase in norm.

Now we have shown that V_1 can be approximated by a rank r expansion up to an accuracy of ε_k, and recall that we denoted this expansion for $k = 2$ as $\mathcal{U}_2(x_2)V_2(x_{>2})$ in (A.14). We can recursively apply this argument to $V_2, V_3, \ldots$

$$\|V_\ell(x_{\ell+1}, x_{>\ell+1}) - \mathcal{U}_{\ell+1}(x_{\ell+1})V_{\ell+1}(x_{>\ell+1})\| \leq \varepsilon_k, \quad \ell = 1, \ldots, d-1,$$

to obtain the required error bounds. In the end, the functions $U_1, \mathcal{U}_2, \mathcal{U}_3, \ldots, \mathcal{U}_{d-1}$, and V_d will make up the cores of the FT with the specified error.

□

A.5 Proof of Theorem 5

The following proposition will be used in the proof.

Proposition 18. *Let* $\mathbf{A} \in \mathbb{R}^{n\times 1}$, $\mathbf{B} \in \mathbb{R}^{m\times 1}$, *and* $\mathbf{C} \in \mathbb{R}^{n\times m}$. *Then*

$$\mathbf{A}^T\mathbf{C}\mathbf{B} = \mathrm{Tr}\left(\mathbf{A}\mathbf{B}^T\mathbf{C}^T\right) \tag{A.18}$$

Proof. Let $\mathbf{D} = \mathbf{A}\mathbf{B}^T\mathbf{C}^T$, $\mathbf{D} \in \mathbb{R}^{n\times n}$.

$$\begin{aligned}
\mathrm{Tr}\left(\mathbf{D}\right) &= \sum_{i=1}^{n} \mathbf{D}[i,i] = \sum_{i=1}^{n}\sum_{j=1}^{m} \mathbf{A}[i,1]\mathbf{B}[j,1]\left(\mathbf{C}^T\right)[j,i] \\
&= \sum_{i=1}^{n}\sum_{j=1}^{m} \mathbf{A}[i,1]\mathbf{B}[j,1]\mathbf{C}[i,j] \\
&= \mathbf{A}^T\mathbf{C}\mathbf{B}
\end{aligned}$$

□

The proof strategy is based on the calculus of variations.

Proof. Let $\mathcal{H} : \mathscr{Y}_2 \to \mathbb{R}^{n_1\times m_1}$ and consider the first variation of J defined as

$$\delta J(\mathcal{W}, \mathcal{H}) = \lim_{\epsilon\to 0} \frac{J(\mathcal{W}+\epsilon\mathcal{H}) - J(\mathcal{Z})}{\epsilon} = \frac{d}{d\epsilon} J(\mathcal{W}+\epsilon\mathcal{H})|_{\epsilon=0} \tag{A.19}$$

Denote $g(x_1, y_2, y_3) = \mathcal{G}_1(y_1)\mathcal{G}_2(y_2)\mathcal{G}_3(y_3)$, then we have

$$\begin{aligned}
\delta J(\mathcal{W}, \mathcal{H}) &= \frac{d}{d\epsilon} J(\mathcal{W} + \epsilon\mathcal{H})|_{\epsilon=0} \\
&= \frac{d}{d\epsilon}\Big[\int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} \left(\mathcal{L}(y_1)\left(\mathcal{W}(y_2) + \epsilon\mathcal{H}(y_2)\right)\mathcal{R}(y_3)\right. \\
&\qquad\qquad \left. - g\left(y_1, y_2, y_3\right)\right)^2 dy_1 dy_2 dy_3 \Big]_{\epsilon=0} \\
&= \frac{d}{d\epsilon}\Big[\int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} \left(\mathcal{L}(y_1)\mathcal{W}(y_2)\mathcal{R}(y_3)\right)^2 dy_1 dy_2 dy_3 + \\
&\qquad \epsilon^2 \int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} \left(\mathcal{L}(y_1)\mathcal{H}(y_2)\mathcal{R}(y_3)\right)^2 dy_1 dy_2 dy_3 + \\
&\qquad \int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} g(y_1, y_2, y_3)^2 dy_1 dy_2 dy_3 + \\
&\qquad 2\epsilon \int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} \mathcal{L}(y_1)\mathcal{W}(y_2)\mathcal{R}(y_3)\mathcal{L}(y_1)\mathcal{H}(y_2)\mathcal{R}(y_3) dy_1 dy_2 dy_3 - \\
&\qquad 2 \int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} \mathcal{L}(y_1)\mathcal{W}(y_2)\mathcal{R}(y_3) g(y_1, y_2, y_3) dy_1 dy_2 dy_3 - \\
&\qquad 2\epsilon \int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} \mathcal{L}(y_1)\mathcal{H}(y_2)\mathcal{R}(y_3) g(y_1, y_2, y_3) dy_1 dy_2 dy_3 \Big]_{\epsilon=0}
\end{aligned}$$

$$\begin{aligned}
&= \Big[2\epsilon \int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} \left(\mathcal{L}(y_1)\mathcal{H}(y_2)\mathcal{R}(y_3)\right)^2 dy_1 dy_2 dy_3 + \\
&\qquad 2 \int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} \mathcal{L}(y_1)\mathcal{W}(y_2)\mathcal{R}(y_3)\mathcal{L}(y_1)\mathcal{H}(y_2)\mathcal{R}(y_3) dy_1 dy_2 dy_3 - \\
&\qquad 2 \int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} \mathcal{L}(y_1)\mathcal{H}(y_2)\mathcal{R}(y_3) g(y_1, y_2, y_3) dy_1 dy_2 dy_3 \Big]_{\epsilon=0} \\
&= 2\Big[\int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} \mathcal{L}(y_1)\mathcal{W}(y_2)\mathcal{R}(y_3)\mathcal{L}(y_1)\mathcal{H}(y_2)\mathcal{R}(y_3) - \\
&\qquad \mathcal{L}(y_1)\mathcal{H}(y_2)\mathcal{R}(y_3) g(y_1, y_2, y_3) dy_1 dy_2 dy_3 \Big], \qquad \text{(A.20)}
\end{aligned}$$

where the second equality uses the definition of J, the third equality comes from expanding the square, the fourth equality comes from taking the derivative, and the last equality comes from setting $\epsilon = 0$.

Now, using the fact that $\mathcal{L}(y_1)\mathcal{H}(y_2)\mathcal{R}(y_3) = \mathcal{R}^T(y_3)\mathcal{H}^T(y_2)\mathcal{L}_1^T(y_1)$ and the orthonormality condition of $\mathcal{R}$ we can simplify the first term of integrand in the last

equation

$$\int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} \mathcal{L}(y_1)\mathcal{W}(y_2)\mathcal{R}(y_3)\mathcal{L}(y_1)\mathcal{H}(y_2)\mathcal{R}(y_3)dy_1dy_2dy_3 =$$
$$\int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} \mathcal{L}(y_1)\mathcal{W}(y_2)\mathcal{R}(y_3)\mathcal{R}^T(y_3)\mathcal{H}^T(y_2)\mathcal{L}_1^T(y_1)dy_1dy_2dy_3$$
$$= \int_{\mathscr{Y}_1\times\mathscr{Y}_2} \mathcal{L}(y_1)\mathcal{W}(y_2)\mathcal{H}^T(y_2)\mathcal{L}^T(y_1)dy_1dy_2$$
$$= \int_{\mathscr{Y}_1\times\mathscr{Y}_2} \mathrm{Tr}\left(\mathcal{L}(y_1)^T\mathcal{L}(y_1)\mathcal{H}(y_2)\mathcal{W}^T(y_2)\right)dy_1dy_2$$
$$= \int_{\mathscr{Y}_2} \mathrm{Tr}\left(\mathcal{H}(y_2)\mathcal{W}^T(y_2)\right)dy_2, \qquad \text{(A.21)}$$

where Proposition 18 applied to matrix-valued functions is to obtain the third equality. Combining (A.21) with (A.20), the first variation becomes

$$\delta J(\mathcal{W},\mathcal{H}) = 2\Big[\int_{\mathscr{Y}_2} \mathrm{Tr}\left(\mathcal{H}(y_2)\mathcal{W}^T(y_2)\right)dy_2$$
$$- \int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} \mathcal{L}(y_1)\mathcal{H}(y_2)\mathcal{R}(y_3)g(y_1,y_2,y_3)dy_1dy_2dy_3\Big] \qquad \text{(A.22)}$$

A necessary condition for an extremum is $\delta J(\mathcal{W}^*,\mathcal{H}) = 0$. To verify this condition

we substitute (3.17) into (A.21)

$$\begin{aligned}
&\int_{\mathscr{Y}_2} \mathrm{Tr}\left(\mathcal{H}(y_2)\left[\mathcal{W}^*\right]^T(y_2)\right) dy_2 = \\
&\quad \int_{\mathscr{Y}_2} \mathrm{Tr}\left(\mathcal{H}(y_2)\left(\left[\int_{\mathscr{Y}_1} \mathcal{L}^T(y_1)\mathcal{G}_1(y_1)dy_1\right]\mathcal{G}_2(y_2)\left[\int_{\mathscr{Y}_3} \mathcal{G}_3(y_3)\mathcal{R}^T(y_3)dy_3\right]\right)^T\right) dy_2 \\
&\quad = \int_{\mathscr{Y}_2} \mathrm{Tr}\left(\mathcal{H}(y_2)\left(\int_{\mathscr{Y}_1}\int_{\mathscr{Y}_3} \mathcal{L}^T(y_1)\mathcal{G}_1(y_1)\mathcal{G}_2(y_2)\mathcal{G}_3(y_3)\mathcal{R}^T(y_3)dy_1dy_3\right)^T\right) dy_2 \\
&\quad = \int_{\mathscr{Y}_2} \mathrm{Tr}\left(\mathcal{H}(y_2)\left(\int_{\mathscr{Y}_1}\int_{\mathscr{Y}_3} \mathcal{L}^T(y_1)g(y_1,y_2,y_3)\mathcal{R}^T(y_3)dy_1dy_3\right)^T\right) dy_2 \\
&\quad = \int_{\mathscr{Y}_2} \mathrm{Tr}\left(\mathcal{H}(y_2)\left(\int_{\mathscr{Y}_1}\int_{\mathscr{Y}_3} \mathcal{L}^T(y_1)\mathcal{R}^T(y_3)g(y_1,y_2,y_3)dy_1dy_3\right)^T\right) dy_2 \\
&\quad = \int_{\mathscr{Y}_2} \mathrm{Tr}\left(\mathcal{H}(y_2)\int_{\mathscr{Y}_3}\int_{\mathscr{Y}_3} \mathcal{R}(y_3)\mathcal{L}(y_1)g(y_1,y_2,y_3)dy_1dy_3\right) dy_2 \\
&\quad = \int_{\mathscr{Y}_2}\int_{\mathscr{Y}_1}\int_{\mathscr{Y}_3} \mathrm{Tr}\left(\mathcal{H}(y_2)\mathcal{R}(y_3)\mathcal{L}(y_1)\right) g(y_1,y_2,y_3)dy_1dy_3dy_2 \\
&\quad = \int_{\mathscr{Y}_1\times\mathscr{Y}_2\times\mathscr{Y}_3} \mathcal{L}(y_1)\mathcal{H}(y_2)\mathcal{R}(y_3)g(y_1,y_2,y_3)dy_1dy_2dy_3
\end{aligned}$$

where the last equality is obtained using another application of Proposition 18 to matrix-valued functions. Using this result in (A.22) shows that the condition

$$\delta J(\mathcal{W}^*, \mathcal{H}) = 0$$

is satisfied. □

Bibliography

[1] I. Arasaratnam and S. Haykin. Cubature Kalman filters. *IEEE Transactions on Automatic Control*, 54(6):1254–1269, June 2009.

[2] R. Archibald, A. Gelb, R. Saxena, and D. Xiu. Discontinuity detection in multivariate space for stochastic simulations. *Journal of Computational Physics*, 228(7):2676–2689, April 2009.

[3] R. Archibald, A. Gelb, and J. Yoon. Polynomial fitting for edge detection in irregularly sampled signals and images. *SIAM Journal on Numerical Analysis*, 43(1):259–279, January 2005.

[4] Z. Battles and L. N. Trefethen. An extension of MATLAB to continuous functions and operators. *SIAM Journal on Scientific Computing*, 25(5):1743–1770, January 2004.

[5] K. Batygin and M. E. Brown. Evidence for a distant giant planet in the solar system. *The Astronomical Journal*, 151(2):22, January 2016.

[6] M. Bebendorf. Approximation of boundary element matrices. *Numerische Mathematik*, 86(4):565–589, October 2000.

[7] M. Bebendorf. Adaptive cross approximation of multivariate functions. *Constructive Approximation*, 34(2):149–179, January 2010.

[8] R. E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton university press, 1961.

[9] V. E. Beneš. Exact finite-dimensional filters for certain diffusions with nonlinear drift. *Stochastics*, 5(1-2):65–92, June 1981.

[10] T. Bengtsson, P. Bickel, and B. Li. Curse-of-dimensionality revisited: Collapse of the particle filter in very large scale systems. In D. Nolan and T. Speed, editors, *Probability and Statistics: Essays in Honor of David A. Freedman*, volume 2 of *Collections*, pages 316–334. Institute of Mathematical Statistics, Beachwood, Ohio, USA, 2008.

[11] A. Bensoussan, R. Glowinski, and A. Rascanu. Approximation of the Zakai equation by the splitting up method. *SIAM Journal on Control and Optimization*, 28(6):1420–1431, November 1990.

[12] D. P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific Belmont, 2007.

[13] D. P. Bertsekas. *Abstract dynamic programming.* Athena Scientific, Belmont, MA, 2013.

[14] D. P. Bertsekas and J. N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, August 1991.

[15] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*, volume 3 of *Optimization and Neural Computation Series.* Athena Scientific, 1996.

[16] G. Beylkin and M. J. Mohlenkamp. Numerical operator calculus in higher dimensions. *Proceedings of the National Academy of Sciences*, 99(16):10246–10251, July 2002.

[17] D. Bigoni, A. P. Engsig-Karup, and Y. M. Marzouk. Spectral Tensor-Train decomposition. *SIAM Journal on Scientific Computing*, 38(4):A2405–A2439, January 2016.

[18] C. Boutsidis and D. P. Woodruff. Optimal CUR matrix decompositions. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing - STOC '14.* Association for Computing Machinery (ACM), 2014.

[19] W. L. Briggs, S. F. McCormick, et al. *A Multigrid Tutorial.* SIAM, 2000.

[20] E. Brunskill, L. Kaelbling, T. Lozano-Perez, and N. Roy. Continuous-State POMDPs with Hybrid Dynamics. *International Symposium on Artificial Intelligence & Mathematics ISAIM*, 2008.

[21] G. Burgers, P. J. van Leeuwen, and G. Evensen. Analysis scheme in the ensemble Kalman filter. *Monthly Weather Review*, 126(6):1719–1724, June 1998.

[22] L. R. G. Carrillo, A. E. D. López, R. Lozano, and C. Pégard. *Quad Rotorcraft Control: Vision-based Hovering and Navigation.* Springer Science & Business Media, 2012.

[23] J. D. Carroll and J.-J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika*, 35(3):283–319, September 1970.

[24] J. A. Carton and B. S. Giese. A reanalysis of ocean climate using Simple Ocean Data Assimilation (soda). *Monthly Weather Review*, 136(8):2999–3017, 2008.

[25] A. R. Cassandra. A survey of POMDP applications. *Working Notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Processes*, pages 17–24, 1998.

[26] M. Chevreuil, R. Lebrun, A. Nouy, and P. Rai. A least-squares method for sparse low rank approximation of multivariate functions. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):897–921, 2015.

[27] A. Chorin, M. Morzfeld, and X. Tu. Implicit particle filters for data assimilation. *Communications in Applied Mathematics and Computational Science CAMCoS*, 5(2):221–240, November 2010.

[28] C.-S. Chow and J. Tsitsiklis. An optimal one-way multigrid algorithm for discrete-time stochastic control. *IEEE Transactions on Automatic Control*, 36(8):898–914, 1991.

[29] A. Cichocki. Tensor networks for big data analytics and large-scale optimization problems. *arXiv preprint arXiv:1407.3124*, July 2014.

[30] J. M. C. Clark. The design of robust approximations to the stochastic differential equations of nonlinear filtering. *Communication Systems and Random Process Theory*, 25:721–734, 1978.

[31] P. R. Conrad, Y. M. Marzouk, N. S. Pillai, and A. Smith. Accelerating asymptotically exact MCMC for computationally intensive models via local approximations. *Journal of the American Statistical Association*, pages 00–00, October 2015.

[32] R. Cory and R. Tedrake. Experiments in fixed-wing UAV perching. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, pages 1–12. AIAA Reston, VA, American Institute of Aeronautics and Astronautics (AIAA), August 2008.

[33] P. J. Costa. Adaptive model architecture and extended Kalman-Bucy filters. *IEEE Transactions on Aerospace and Electronic Systems*, 30(2):525–533, 1994.

[34] F. Daum and J. Huang. Curse of dimensionality and particle filters. In *2003 IEEE Aerospace Conference Proceedings*, volume 4, pages 1979–1993. Institute of Electrical & Electronics Engineers (IEEE), March 2003.

[35] D. Day and L. Romero. Roots of polynomials expressed in terms of orthogonal polynomials. *SIAM Journal on Numerical Analysis*, 43(5):1969–1987, January 2005.

[36] V. De Silva and L. Lim. Tensor rank and the ill-posedness of the best low-rank approximation problem. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1084–1127, January 2008.

[37] S. V. Dolgov. TT-GMRES: solution to a linear system in the structured tensor format. *Russian Journal of Numerical Analysis and Mathematical Modelling*, 28(2), January 2013.

[38] S. V. Dolgov and D. V. Savostyanov. Alternating minimal energy methods for linear systems in higher dimensions. *SIAM Journal on Scientific Computing*, 36(5):A2248–A2271, January 2014.

[39] D. L. Donoho et al. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture*, pages 1–32, 2000.

[40] A. Doostan, A. Validi, and G. Iaccarino. Non-intrusive low-rank separated approximation of high-dimensional stochastic models. *Computer Methods in Applied Mechanics and Engineering*, 263:42–55, August 2013.

[41] A. Doucet, S. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10(3):197–208, 2000.

[42] P. Drineas, R. Kannan, and M. W. Mahoney. Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. *SIAM Journal on Computing*, 36(1):158–183, January 2006.

[43] G. Evensen. The ensemble Kalman filter: theoretical formulation and practical implementation. *Ocean Dynamics*, 53(4):343–367, November 2003.

[44] W. H. Fleming and H. M. Soner. *Controlled Markov Processes and Viscosity Solutions*. Springer New York, 2006.

[45] M. Frei and H. R. Kunsch. Bridging the ensemble Kalman and particle filters. *Biometrika*, 100(4):781–800, July 2013.

[46] R. Furrer and T. Bengtsson. Estimation of high-dimensional prior and posterior covariance matrices in Kalman filter variants. *Journal of Multivariate Analysis*, 98(2):227–255, Feburary 2007.

[47] A. Genz. Testing multidimensional integration routines. In *Proc. of International Conference on Tools, Methods and Languages for Scientific and Engineering Computation*, pages 81–94. Elsevier North-Holland, Inc., 1984.

[48] M. Ghil and P. Malanotte-Rizzoli. Data assimilation in meteorology and oceanography. *Advances in Geophysics*, 33:141–266, 1991.

[49] N. Gordon, D. Salmond, and A. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEE Proceedings F (Radar and Signal Processing)*, volume 140, pages 107–113. IET, Institution of Engineering and Technology (IET), 1993.

[50] S. A. Goreinov, I. V. Oseledets, D. V. Savostyanov, E. E. Tyrtyshnikov, and N. L. Zamarashkin. How to find a good submatrix. *Matrix Methods: Theory, Algorithms and Applications*, pages 247–256, April 2010.

[51] S. A. Goreinov and E. E. Tyrtyshnikov. The maximal-volume concept in approximation by low-rank matrices. *Contemporary Mathematics*, 280:47–51, 2001.

[52] S. A. Goreinov, E. E. Tyrtyshnikov, and N. L. Zamarashkin. A theory of pseudoskeleton approximations. *Linear Algebra and its Applications*, 261(1):1–21, 1997.

[53] S. A. Goreinov, N. L. Zamarashkin, and E. E. Tyrtyshnikov. Pseudo-skeleton approximations by matrices of maximal volume. *Mathematical Notes*, 62(4):515–519, oct 1997.

[54] A. Gorodetsky. C^3: Compressed Continuous Computation library. `https://github.com/goroda/Compressed-Continuous-Computation`.

[55] A. Gorodetsky. Stochastic control using compressed continuous computation (C3SC). `https://github.com/goroda/c3sc`.

[56] A. Gorodetsky, S. Karaman, and Y. Marzouk. Efficient high-dimensional stochastic optimal motion control using Tensor-Train decomposition. In *Robotics: Science and Systems XI*, Rome, Italy, July 2015. Robotics: Science and Systems Foundation.

[57] A. Gorodetsky, S. Karaman, and Y. Marzouk. Function-Train: A continuous analogue of the Tensor-Train decomposition. *arXiv preprint arXiv:1510.09088*, 2015.

[58] A. Gorodetsky and Y. Marzouk. Efficient localization of discontinuities in complex computational simulations. *SIAM Journal on Scientific Computing*, 36(6):A2584–A2610, January 2014.

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

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

[61] W. Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*. Springer-Verlag, Berlin, 2012.

[62] W. Hackbusch and S. Kühn. A new scheme for the tensor representation. *Journal of Fourier Analysis and Applications*, 15(5):706–722, October 2009.

[63] R. A. Harshman. Foundations of the PARAFAC procedure: models and conditions for an "explanatory" multimodal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.

[64] J. Håstad. Tensor rank is NP-complete. *Journal of Algorithms*, 11(4):644–654, December 1990.

[65] F. L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, April 1927.

[66] F. L. Hitchcock. Multiple invariances and generalized rank of a p-way matrix or tensor. *J. Math. Phys.*, 7:39–79, 1927.

[67] M. B. Horowitz, A. Damle, and J. W. Burdick. Linear Hamilton Jacobi Bellman equations in high dimensions. In *53rd IEEE Conference on Decision and Control*, pages 5880–5887. Institute of Electrical & Electronics Engineers (IEEE), December 2014.

[68] K. Ito and B. Rozovskii. Approximation of the Kushner equation for nonlinear filtering. *SIAM Journal on Control and Optimization*, 38(3):893–915, January 2000.

[69] K. Ito and K. Xiong. Gaussian filters for nonlinear filtering problems. *IEEE Transactions on Automatic Control*, 45(5):910–927, May 2000.

[70] E. Jones, T. Oliphant, P. Peterson, et al. Open source scientific tools for python, 2001.

[71] S. J. Julier and J. K. Uhlmann. New extension of the Kalman filter to nonlinear systems. In I. Kadar, editor, *Signal Processing, Sensor Fusion, and Target Recognition VI*. SPIE-Intl Soc Optical Eng, July 1997.

[72] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, March 1960.

[73] R. E. Kalman and R. S. Bucy. New results in linear filtering and prediction theory. *Journal of Basic Engineering*, 83(1):95–108, March 1961.

[74] B. N. Khoromskij and I. V. Oseledets. QTT approximation of elliptic solution operators in higher dimensions. *Russian Journal of Numerical Analysis and Mathematical Modelling*, 26(3):303–322, January 2011.

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

[76] J. B. Kruskal. *Multiway data analyis*, chapter Rank, decomposition, and uniqueness for 3-way and N-way arrays. Amsterdam, North-Holland, 1989.

[77] H. J. Kushner. On the differential equations satisfied by conditional probablitity densities of Markov processes, with applications. *Journal of the Society for Industrial and Applied Mathematics Series A Control*, 2(1):106–119, January 1964.

[78] H. J. Kushner. Dynamical equations for optimal nonlinear filtering. *Journal of Differential Equations*, 3(2):179–190, April 1967.

[79] H. J. Kushner. *Probability methods for approximations in stochastic control and for elliptic equations*, volume 129 of *Mathematics in science and engineering*. New York: Academic Press, 1977.

[80] H. J. Kushner. Numerical methods for stochastic control problems in continuous time. *SIAM Journal on Control and Optimization*, 28(5):999–1048, September 1990.

[81] H. J. Kushner and P. G. Dupuis. *Numerical methods for stochastic control problems in continuous time.* Springer, 2001.

[82] L. D. Lathauwer, B. D. Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, January 2000.

[83] J. Lei and P. Bickel. A moment matching ensemble filter for nonlinear non-Gaussian data assimilation. *Monthly Weather Review*, 139(12):3964–3973, December 2011.

[84] P. F. J. Lermusiaux. Estimation and study of mesoscale variability in the Strait of Sicily. *Dynamics of Atmospheres and Oceans*, 29(2):255–303, 1999.

[85] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. *ICML*, pages 362–370, 1995.

[86] S. Lototsky, R. Mikulevicius, and B. L. Rozovskii. Nonlinear filtering revisited: a spectral approach. *SIAM Journal on Control and Optimization*, 35(2):435–461, March 1997.

[87] M. W. Mahoney and P. Drineas. CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, January 2009.

[88] Y. M. Marzouk, H. N. Najm, and L. A. Rahn. Stochastic spectral methods for efficient Bayesian solution of inverse problems. *Journal of Computational Physics*, 224(2):560–586, June 2007.

[89] J. Moore and R. Tedrake. Control synthesis and verification for a perching uav using lqr-trees. In *CDC*, pages 3707–3714. Citeseer, 2012.

[90] G. N. Newsam. On the asymptotic distribution of the eigenvalues of discretizations of a compact operator. In *Proc. Center for Mathematical Analysis*, volume 17, pages 92–105, 1988.

[91] B. Oksendal. *Stochastic Differential Equations: An Introduction with Applications.* Springer Verlag, 2003.

[92] S. Olver. Approxfun: Julia package for function approximation. `https://github.com/ApproxFun/ApproxFun.jl`, 2014.

[93] S. Olver and A. Townsend. A practical framework for infinite-dimensional linear algebra. In *Proceedings of the 1st First Workshop for High Performance Technical Computing in Dynamic Languages*, pages 57–62. Institute of Electrical & Electronics Engineers (IEEE), November 2014.

[94] I. V. Oseledets. DMRG approach to fast linear algebra in the TT-format. *Computational Methods in Applied Mathematics*, 11(3):382–393, 2011.

[95] I. V. Oseledets. Tensor-Train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, January 2011.

[96] I. V. Oseledets. Constructive representation of functions in low-rank tensor formats. *Constructive Approximation*, 37(1):1–18, December 2012.

[97] I. V. Oseledets and S. V. Dolgov. Solution of linear systems and matrix inversion in the TT-format. *SIAM Journal on Scientific Computing*, 34(5):A2718–A2739, January 2012.

[98] I. V. Oseledets and E. Tyrtyshnikov. TT-cross approximation for multidimensional arrays. *Linear Algebra and its Applications*, 432(1):70–88, January 2010.

[99] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, August 1987.

[100] M. K. Pitt and N. Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, 94(446):590–599, June 1999.

[101] R. B. Platte and L. N. Trefethen. Chebfun: A new kind of numerical computing. In *Progress in Industrial Mathematics at ECMI 2008*, pages 69–87. Springer Science + Business Media, 2010.

[102] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, volume 703. John Wiley & Sons, 2007.

[103] P. Rai. *Sparse low-rank approximation of multivariate functions - applications in uncertainty quantification.* PhD thesis, Ecole Centrale Nantes, 2014.

[104] J. W. Roberts, R. Cory, and R. Tedrake. On the controllability of fixed-wing perching. In *2009 American Control Conference*, pages 2018–2023. IEEE, Institute of Electrical & Electronics Engineers (IEEE), 2009.

[105] J. Rust. Using randomization to break the curse of dimensionality. *Econometrica*, 65(3):487–516, May 1997.

[106] S. Sarkka. On unscented Kalman filtering for state estimation of continuous-time nonlinear systems. *IEEE Transactions on Automatic Control*, 52(9):1631–1641, September 2007.

[107] D. V. Savostyanov. Quasioptimality of maximum-volume cross interpolation of tensors. *Linear Algebra and its Applications*, 458:217–244, October 2014.

[108] E. Schmidt. Zur theorie der linearen und nicht linearen integralgleichungen zweite abhandlung. *Mathematische Annalen*, 64(2):161–174, June 1907.

[109] C. Snyder. Particle filters, the "optimal" proposal and high-dimensional systems. In *Proceedings of the ECMWF Seminar on Data Assimilation for Atmosphere and Ocean*, 2011.

[110] C. Snyder, T. Bengtsson, P. Bickel, and J. Anderson. Obstacles to high-dimensional particle filtering. *Monthly Weather Review*, 136(12):4629–4640, December 2008.

[111] A. Sparkes, W. Aubrey, E. Byrne, A. Clare, M. N. Khan, M. Liakata, M. Markham, J. Rowland, L. N. Soldatova, K. E. Whelan, M. Young, and R. D. King. Towards robot scientists for autonomous scientific discovery. *Automated Experimentation*, 2(1):1–11, 2010.

[112] G. W. Stewart. Fredholm, Hilbert, Schmidt: three fundamental papers on integral equations. `www.cs.umd.edu/~stewart/FHS.pdf`, 2011.

[113] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: test functions and datasets. Retrieved September 4, 2015, from `http://www.sfu.ca/~ssurjano`.

[114] S. Thomke and T. Fujimoto. The effect of "front-loading" problem-solving on product development performance. *Journal of Product Innovation Management*, 17(2):128–142, March 2000.

[115] S. Thrun. Monte Carlo POMDPs. In S. Solla, T. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 1064–1070. MIT Press, 1999.

[116] C. Tobler. *Low-rank tensor methods for linear systems and eigenvalue problems.* PhD thesis, ETH Zürich, 2012.

[117] A. Townsend and L. N. Trefethen. An extension of Chebfun to two dimensions. *SIAM Journal on Scientific Computing*, 35(6):C495–C518, January 2013.

[118] A. Townsend and L. N. Trefethen. Continuous analogues of matrix factorizations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2173):20140585–20140585, November 2014.

[119] L. N. Trefethen. Householder triangularization of a quasimatrix. *IMA Journal of Numerical Analysis*, 30(4):887–897, August 2009.

[120] U. Trottenberg, C. W. Oosterlee, and A. Schuller. *Multigrid.* Academic press, 2000.

[121] J. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, 1995.

[122] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[123] A. Uschmajew. Local convergence of the alternating least squares algorithm for canonical tensor approximation. *SIAM Journal on Matrix Analysis and Applications*, 33(2):639–652, January 2012.

[124] P. J. van Leeuwen. Particle filtering in geophysical systems. *Monthly Weather Review*, 137(12):4089–4114, December 2009.

[125] Y. Wu, D. Hu, M. Wu, and X. Hu. A numerical-integration perspective on Gaussian filters. *IEEE Transactions o Signal Processing*, 54(8):2910–2921, 2006.

[126] M. Xue, D. Wang, J. Gao, K. Brewster, and K. K. Droegemeier. The Advanced Regional Prediction System (ARPS), storm-scale numerical weather prediction and data assimilation. *Meteorology and Atmospheric Physics*, 82(1):139–170, 2003.

[127] I. Yang, M. Morzfeld, C. J. Tomlin, and A. J. Chorin. Path integral formulation of stochastic optimal control with generalized costs. *IFAC Proceedings Volumes*, 47(3):6994–7000, 2014.

[128] M. Zakai. On the optimal filtering of diffusion processes. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 11(3):230–243, 1969.