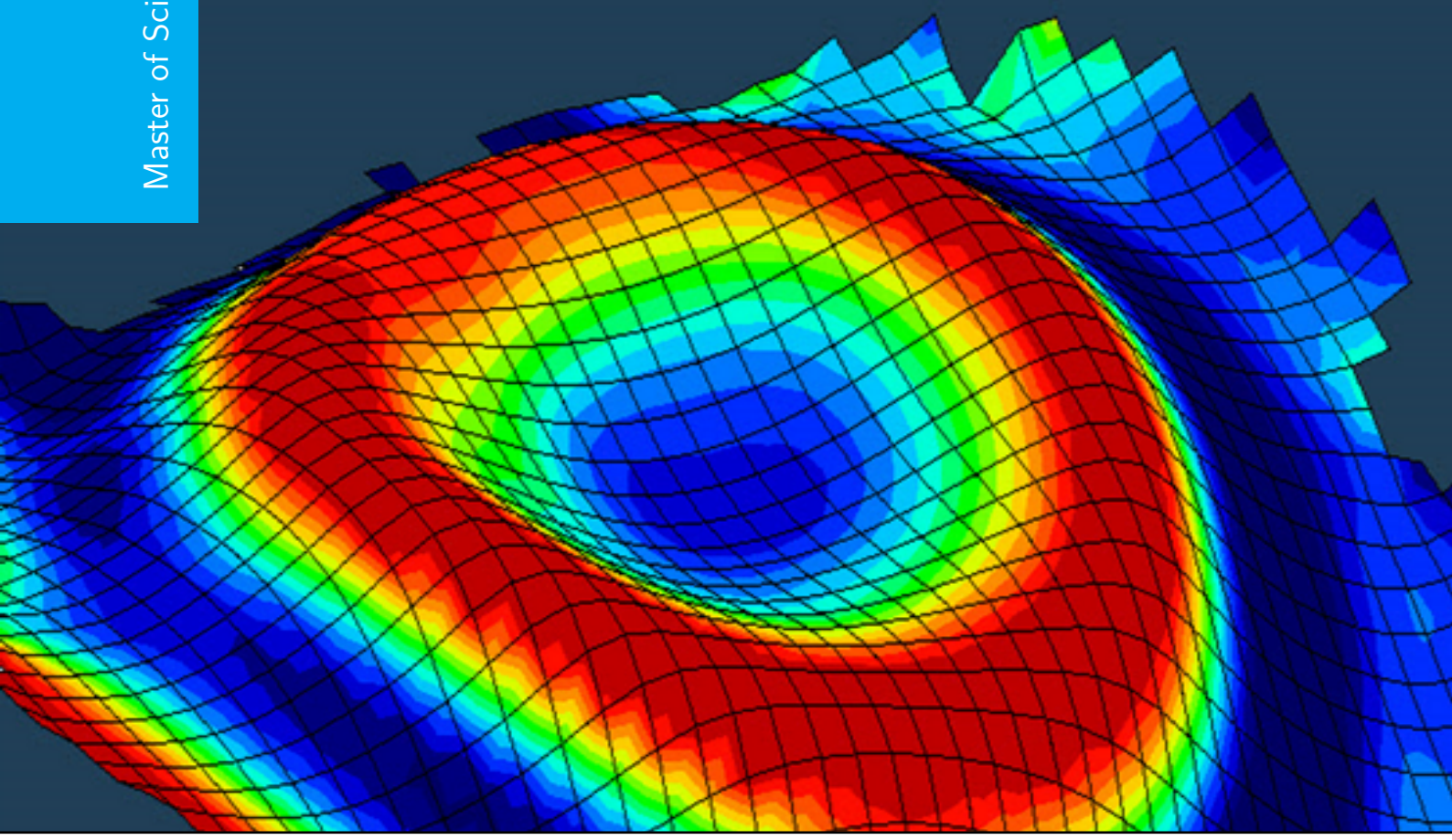
Data-Driven Optimal Control

An Inverse Optimization Model and Algorithm

LONG, YOUYUAN

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Data-Driven Optimal Control

An Inverse Optimization Model and Algorithm

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

LONG, YOUYUAN

April 20, 2024

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

In Inverse Optimization (IO), it is hypothesized that experts, when making decisions, implicitly engage in solving an optimization problem. If we can reconstruct this optimization problem using the decision data of the expert, then the behavior of the expert can be emulated. In this thesis, a novel inverse optimization model, Kernel Inverse Optimization Machine (KIOM), is proposed, utilizing kernel methods. Because its parameter space can be potentially infinite-dimensional, the model exhibits strong representation and generalization capabilities. Furthermore, empirical evidence is presented demonstrating the model's ability to learn complex MuJoCo continuous control tasks. Subsequently, an algorithm for training KIOM, Sequential Selection Optimization (SSO), is proposed to address memory issues. SSO is a coordinate descent-based algorithm, and its memory requirements are nearly equal to the memory needed for solving one of its subproblems. Experimental results demonstrate that SSO converges to the optimal solution within a small number of iterations, highlighting its efficiency.

Keywords: Imitation Learning, Inverse Optimization (IO), Kernel Method, Coordinate Descent (CD)

Table of Contents

Acknowledgements v

1 Introduction 1
1-1 Inverse Optimization . . . 1
1-1-1 Background and research status . . . 1
1-1-2 Process of data-driven inverse optimization . . . 2
1-2 Kernel Method . . . 5
1-3 Coordinate Descent Algorithm . . . 6
1-4 Organization of Chapters . . . 8
1-5 Notation . . . 8

2 Kernel Inverse Optimization Machine 9
2-1 Learning for Control: An Inverse Optimization Model . . . 9
2-1-1 A quadratic forward model . . . 9
2-1-2 Suboptimality loss function . . . 10
2-1-3 Inverse optimization problem and its tractable reformulation . . . 10
2-1-4 Analysis of the model's limitations . . . 11
2-2 An Enhanced Model: Kernel Inverse Optimization Machine . . . 12
2-2-1 Theoretical derivation . . . 12
2-2-2 Extra assumption: a simpler version . . . 15
2-2-3 An extension with augmented suboptimality loss . . . 17
2-3 Numerical Experiments . . . 18
2-3-1 Experimental environment, dataset, and solver . . . 18
2-3-2 Results . . . 20

3 Sequential Selection Optimization 23
3-1 A Distributed Algorithm: Sequential Selection Optimization . . . 24
3-1-1 Naive coordinate descent . . . 24
3-1-2 Convergence analysis . . . 25
3-1-3 Heuristics for choosing which coordinates to optimize . . . 28
3-1-4 Warm-up trick for improved initial guess . . . 29
3-2 Numerical Experiments . . . 30
3-2-1 Performance evaluation . . . 31
3-2-2 Ablation studies . . . 32

4 Conclusion and Future Study 35
4-1 Conclusion . . . 35
4-2 Future Study . . . 35

A Proofs **37**
A-1 Corollary 1 37
A-2 Corollary 3 38

B Hyperparameters **41**
B-1 Hyperparameters of KIOM 41

Bibliography **43**

Acknowledgements

I would like to express my gratitude to my supervisor, Peyman Mohajerin Esfahani, for his patience in discussing career planning and research direction with me during the proposal stage. He has granted me considerable freedom in my research and provided valuable advice and feedback throughout the research process. Additionally, I want to thank my daily supervisors, Tolga Ok and Pedro Zattoni Scroccaro, for their profound knowledge and witty humor. They have engaged in detailed discussions with me whenever I encountered research difficulties and offered their analytical insights. Their substantial assistance has been invaluable to my research. Lastly, I am grateful to my parents for their support during my academic journey abroad.

Delft, University of Technology LONG, YOUYUAN
April 20, 2024

Chapter 1

Introduction

1-1 Inverse Optimization

1-1-1 Background and research status

Inverse Optimization (IO) is distinct from traditional optimization problems where we typically seek the optimal decision variables by providing an objective function and a set of constraints. In contrast, inverse optimization works "in reverse" by determining what the optimization problem is when the optimal solution is given. This approach is based on an intriguing assumption: when individuals address a problem, we assume that there exists an underlying optimization problem which is also called Forward Optimization Problem (FOP) parametric in the exogenous signal s in their minds, and their decision u represents the optimal solution to this problem. Therefore inverse optimization involves deducing such an FOP from a dataset of exogenous signal and decision pairs $\{(\hat{s}_i, \hat{u}_i)\}_{i=1}^{N}$. Consequently, we can successfully accomplish tasks even in the absence of experts by solving the constructed FOP and executing the corresponding optimal action. However inverse optimization problems belong to the category of traditional optimization problems. That is, when we reversely infer the optimization problem, we are essentially solving a traditional optimization problem.

The field of inverse optimization has garnered widespread attention, giving rise to numerous studies encompassing both theoretical and applied research. A new survey on inverse optimization has been written by Chan et al. [18], which comprehensively introduces the theory and application part of this area. Inverse optimization is widely employed across various domains, including vehicle routing [20, 49], transportation system modeling [46, 14], Portfolio Optimization [40, 64, 37], power systems [16, 24, 50], electric vehicle charging problems [25], network design [23], healthcare problems [8] as well as controller design [2]. In the previously mentioned survey, there is a more extensive introduction to the application section. Moving forward, we will delve into a detailed discussion of the theoretical contributions within the field of inverse optimization.

Inverse optimization can be categorized into classic IO and data-driven IO. In classic IO, when an exogenous signal is provided from the training set, the corresponding decision data is required to be the optimal solution of the FOP obtained by solving the inverse optimization problem. In this branch, many researchers have engaged in theoretical studies, for instance, some scholars have conducted different research on the model of FOP including the linear forward model and conic forward model [1, 54, 34]. However, due to the presence of noise in real-world data, it may be unreasonable to force data with noise as the optimal solution of the constructed FOP in certain situations. Additionally, for complex tasks, a relatively simple convex forward model may only approximate the task and cannot precisely describe it. In such cases, the requirement that the decision data in the training set are all optimal solutions to the forward optimization problem parametric in the corresponding exogenous signals may be unattainable. Therefore, this introduces another branch: data-driven inverse optimization. In such scenarios, as decision data is not mandated to be optimal, it becomes necessary to introduce an additional loss function to penalize the discrepancy between decision data and the actual optimal solution. Many scholars have researched such loss functions, including *2-norm distance loss* [7], *absolute sub-optimality loss* [9], *relative sub-optimality loss* [32], *variational inequality loss* [14], *KKT loss* [35]. In addition to research on loss functions, there are also numerous promising research avenues. For example, Bertsimas et al. [14] consider non-parametric kernel functions to model a forward optimization objective and Mohajerin Esfahani & Kuhn [40] propose a distributionally robust inverse optimization problem where the ambiguity set controls a worst-case data distribution from the empirical distribution corresponding to the data set [18]. In this section, our main focus is on the research in the field of data-driven approaches.

1-1-2 Process of data-driven inverse optimization

This subsection will outline the core process of data-driven inverse optimization which encompasses the design of three crucial components: the forward model, loss function, and optimization algorithm. It is essential to emphasize that not all inverse optimization problems follow this process, and different considerations may apply to various problems.

Forward optimization model

As previously mentioned, when an expert agent makes decisions, we posit that the agent solves an optimization problem based on exogenous signals s, and the corresponding optimal solution, u^*, represents the decision the agent will take. Therefore, our first step is to determine the structure of this optimization problem, which is also called forward optimization model/problem. Mathematically, the forward optimization model can be described as

$$\mathbf{FOP}(s \mid \theta) := \min_{u}\{f(s,u,\theta)|u \in \mathcal{X}(s)\} \tag{1-1}$$

and the hypothesis space $\mathcal{F}$ is

$$\mathcal{F} = \{f(s,u,\theta) \mid \theta \in \Theta\}. \tag{1-2}$$

The model is parameterized by a parameter θ from a parameter class Θ that controls the objective function $f(s,u,\theta)$, where $s \subseteq \mathbb{R}^m$ and $u \subseteq \mathbb{R}^n$. It is also parametric in s which represents the environment's state or observation. The objective of the inverse optimization

is to identify the optimal θ such that given an exogenous signal s, the corresponding optimal action u^*, calculated by solving the forward optimization problem, aligns closely, if not identically, with the actual expert's action. There are some common choices for modeling the objective function [18]:

- **Linear:** $f(s,u,\theta) = \theta^T u$, where $\Theta \subseteq \mathbb{R}^n$.
- **Quadratic:** $f(s,u,\theta := (\theta_{uu}, \theta_{su}, q)) = u^T\theta_{uu}u + s^T\theta_{su}u + q^T u$, where $\Theta \subseteq \left\{(\theta_{uu}, \theta_{su}, q) \mid \theta_{uu} \in \mathbb{S}_+^{n\times n}, \theta_{su} \in \mathbb{R}^{m\times n}, q \in \mathbb{R}^n\right\}$.
- **Convex-separable bases:** $f(s,u,\theta) = \sum_{b=1}^{B} \theta_b f^{(b)}(s,u)$, where $f^{(1)}(s,u), f^{(2)}(s,u), \ldots, f^{(B)}(s,u)$ are B convex basis functions and $\Theta \subseteq \mathbb{R}^B$.

The choice of the model is important. On the one hand, the hypothesis space $\mathcal{F}$ should be rich enough to get closer to or even contain the agent's unknown true objective function f. On the other hand, $\mathcal{F}$ should be small enough to ensure tractability of the inverse optimization problem and to prevent degeneracy of its optimal solutions [41].

Loss function

The objective of inverse optimization is to find a suitable θ value such that when given a specific exogenous signal s, the optimal decision u^* obtained by solving the forward optimization problem (1-1) aligns with the corresponding expert's decision u, thereby imitating the behavior of the expert. Therefore, we need to use a loss function to penalize the discrepancy between u and u^*. We first define the optimal solution set

$$\mathcal{X}^{opt}(\theta, s) := \arg\min_u \{f(s,u,\theta) | u \in \mathcal{X}(\theta, s)\}. \tag{1-3}$$

The common loss functions include:

- **Minimum distance loss :** $\ell_{\mathrm{D}}(s,u,\mathcal{X}^{\mathrm{opt}}(\theta,s)) := \min_{u^*\in\mathcal{X}^{\mathrm{opt}}(\theta,s)} \|u^* - u\|_2$. This is an intuitive measure that directly penalizes the Euclidean distance between the optimal decision and the expert's decision, but in some cases, using this loss function may result in a non-convex inverse optimization problem.
- **Absolute sub-optimality loss:** $\ell_{\mathrm{ASO}}(s,u,\mathcal{X}^{\mathrm{opt}}(\theta,s)) := \left| f(s,u,\theta) - \min_{u'\in\mathcal{X}(\theta,s)} f(s,u',\theta) \right|$. This loss function penalizes the disparity between the objective function values associated with the observed decisions and the estimated optimal values.
- **Relative sub-optimality loss:** $\ell_{\mathrm{RSO}}(s,u,\mathcal{X}^{\mathrm{opt}}(\theta,s)) := \left| \frac{f(s,u,\theta)}{\min_{u'\in\mathcal{X}(\theta,s)} f(s,u',\theta)} - 1 \right|$. This loss function assesses the competitive ratio between the objective function values of the observed decisions and the estimated optimal values.
- **Variational inequality loss:** $\ell_{\mathrm{VI}}(s,u,\mathcal{X}^{\mathrm{opt}}(\theta,s)) := \max_{u'\in\mathcal{X}(\theta,s)} \nabla_{u'} f(s,u',\theta)^T(u-u')$. The first-order Variational Inequality (VI) is an optimality criterion for any general convex optimization problem with a differentiable objective function [18]. [14] proposed

this Variational Inequality as a loss function for data-driven estimation in equilibrium with inverse optimization.

- **KKT loss:** Keshavarz et al. [35] consider convex forward model and propose KKT loss functions that describe the degree to which each observed decision violates the KKT conditions [18].

Inverse optimization problem and algorithm

Once the forward model and the loss function are selected, the inverse optimization problem can be formulated. Let $\{(\hat{s}_i, \hat{u}_i)\}_{i=1}^N$ be the data set, then the data-driven inverse optimization problem can be

$$\min_{\theta} \left\{ \kappa h(\theta) + \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{s}_i, \hat{u}_i, \mathcal{X}^{\text{opt}}(\theta, \hat{s}_i)) \;\middle|\; \theta \in \Theta \right\}. \tag{1-4}$$

The $h(\theta)$ is an application-specific objective. For example, $h(\theta)$ can be a regularization term, i.e., the sum of the squares of each element of θ, which can prevent overfitting. κ is a non-negative coefficient that describes a trade-off between the application-specific objective $h(\theta)$ and the loss.

In general, if the inverse optimization problem (1-4) is a tractable convex optimization problem or can be reformulated into a tractable convex problem, then this problem is essentially considered solvable as convex optimization problems have been extensively studied, and there are numerous algorithms available for their solution. On the other hand, if this inverse optimization problem cannot be reformulated into a convex problem, it is generally challenging to guarantee the identification of the optimal solution θ^*. Therefore, a crucial issue in the field of inverse optimization is how to formulate the problem as a convex one. This requires thoughtful consideration when selecting the forward model and loss function, also ensuring that these choices are suitable for the given task. In many instances, it becomes necessary to design new forward models and loss functions tailored to the specific requirements. Additionally, the challenge often involves contemplating how to reformulate non-convex inverse optimization problems into convex ones. These issues represent valuable research directions in the field of inverse optimization, and numerous related studies have been previously introduced.

Certainly, even when inverse optimization problems are formulated as convex, considering the time complexity of optimization algorithms becomes essential, especially with large datasets. Generally, large-scale optimization problems often exhibit specific structures. Therefore, selecting suitable algorithms can significantly save computational time. Additionally, in some cases, it is necessary to design new algorithms based on these structures to maximize the speed of the solution process. This is a highly active research direction in the entire field of optimization. Commonly used optimization algorithms include Alternating Direction Method of Multipliers (ADMM) [30], Frank–Wolfe algorithm [26], Mirror descent [43] which is efficient for the IO problems with high cardinality discrete feasible sets and its variant Stochastic Approximate Mirror Descent (SAMD) [65], etc.

1-2 Kernel Method

The kernel method is a powerful technique used in machine learning and statistics for handling non-linear relationships between variables. It has been widely applied in support vector machines (SVMs) and other kernelized algorithms. The fundamental idea behind the kernel method is to implicitly map input data into a higher-dimensional space without explicitly computing the transformation, thus enabling the algorithms to capture complex patterns and non-linear relationships without heavy computational burden.

Specifically, in the field of machine learning, variables in many algorithms appear in the form of inner products, $\langle s_i, s_j \rangle$. Simultaneously, due to certain reasons (such as the necessity to capture non-linear information or dealing with non-linearly separable data in SVM, etc.), there arises a need to map these input variables into a higher-dimensional space. In this scenario, one can define a feature map function $\phi(\cdot)$ to accomplish this task. Therefore, in such situations, we need to compute the value of $\langle \phi(s_i), \phi(s_j) \rangle$. However, the function $\phi(\cdot)$, which maps features to a high-dimensional space — potentially infinite-dimensional — makes computing $\phi(s_i)$ and subsequently performing the inner product practically infeasible. In response to this issue, kernel methods ingeniously circumvent this computationally expensive process, obtaining the calculation of the aforementioned inner product through an alternative, simple, and efficient pathway. Thus, it becomes essential to define a kernel function $\kappa(s_i, s_j) = \langle \phi(s_i), \phi(s_j) \rangle$ to achieve the stated objective. Next, we introduce some fundamental concepts and theorems related to the kernel methods.

Definition 1 (Gram matrix)**.** *Let $\mathcal{X}$ be a nonempty set. Given a kernel $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ and arbitrary inputs $x_1, ...x_n \in \mathcal{X}$, then the $n \times n$ matrix K, where*

$$K_{ij} := \kappa(x_i, x_j), \tag{1-5}$$

is called the Gram matrix (or kernel matrix) of κ with respect to $x_1, ..., x_n$.

Definition 2 (Positive definite kernel)**.** *Let $\mathcal{X}$ be a nonempty set. A function $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ which for all $n \in \mathbb{N}, x_i \in \mathcal{X}, i \in [n]$ gives rise to a positive definite Gram matrix is called a positive definite kernel. A function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, which for all $n \in \mathbb{N}$ and distinct $x_i \in \mathcal{X}$ gives rise to a strictly positive definite Gram matrix, is called a strictly positive definite kernel.*

Besides the two definitions, there exists a significant theorem known as Mercer's Theorem [39] in the context of kernel methods. In simple terms, this theorem states that $\kappa(\cdot, \cdot)$ is a proper kernel (having a corresponding feature map function $\phi(\cdot)$) if and only if its Gram matrix is positive semi-definite. The theorem provides a criterion to determine the effectiveness of a kernel function, and when designing a new kernel, Mercer's theorem can be employed to assess the validity of the kernel function.

Definition 3 (RKHS [6, 12])**.** *A Reproducing Kernel Hilbert Space (RKHS) is a Hilbert space H of functions $f : \mathcal{X} \mapsto \mathbb{R}$ with a reproducing kernel $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ where $\kappa(x, \cdot) \in \mathcal{H}$ and $f(x) = \langle \kappa(x, \cdot), f \rangle$.*

Note that given a kernel, the corresponding RKHS is unique (up to isometric isomorphisms). Given an RKHS, the corresponding kernel is unique. In other words, each kernel generates a new RKHS [31].

Different kernel functions often have varying impacts on the performance of algorithms. Therefore, designing a suitable kernel function is crucial. However, creating a kernel function is not a straightforward task. Fortunately, there are many existing kernel functions to choose from. Some widely used kernel functions include:

- **Linear Kernel:** $\kappa(x_i, x_j) = x_i^T x_j$. The linear kernel is the simplest, representing the inner product of two original vectors.
- **Polynomial Kernel:** $\kappa(x_i, x_j) = (x_i^T x_j + c)^d$. The polynomial kernel introduces additional parameters d, representing the polynomial's degree and c. It can capture non-linear relationships in the data. if $c = 0, d = 1$, the polynomial kernel degenerates into a linear kernel.
- **Gaussian Kernel:** $\kappa(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$. The Gaussian kernel, or RBF kernel, is widely used for non-linear problems, which maps the input variable into infinite dimension space. It introduces a parameter σ to control the width of the Gaussian distribution.
- **Laplacian Kernel:** $\kappa(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|}{\sigma}\right)$. The Gaussian kernel, or RBF kernel, is widely used for non-linear problems. It introduces a parameter σ to control the width of the Gaussian distribution.

The kernel method has found numerous classical applications in the field of machine learning, with one of the most prominent being Support Vector Machines [21], where the kernel techniques are employed to enable the construction of rich classes of nonlinear decision surfaces. Other notable applications include Kernel Principal Component Analysis [52], Kernel Linear Discriminant Analysis [10], and Gaussian Process Kernels [61], where kernel functions are utilized to represent the covariance matrix of the process in the form of a Gram matrix, Kernel Ridge Regression [51], and Kernelized Decision Tree [58]. In recent years, kernel methods have also seen many successful applications in the field of deep neural networks. There have been numerous attempts to introduce the approach of kernelization in deep neural networks for bringing efficient feature engineering or feature enrichment [44]. Some typical results include kernel-based deep belief networks [33], kernel-based deep recurrent neural networks [56, 3] and kernel-based deep convolutional neural networks [38, 42, 19, 4].

1-3 Coordinate Descent Algorithm

In the current era of big data, numerous problems are accompanied by vast amounts of data. As a result, the size of optimization problems is increasingly growing. Many traditional optimization algorithms can efficiently handle medium or small-scale optimization problems. However, when the scale of the problem becomes too large, these algorithms are no longer viable in terms of time and memory constraints [48]. In this context, the *coordinate descent* (CD) method has garnered increasing attention and application. These methods are generally applicable to a variety of problems involving large or high-dimensional datasets as they naturally break down complicated optimization problems into simpler subproblems, which can be efficiently parallelized or distributed [55].

Coordinate descent methods are iterative algorithms where each iteration involves fixing the

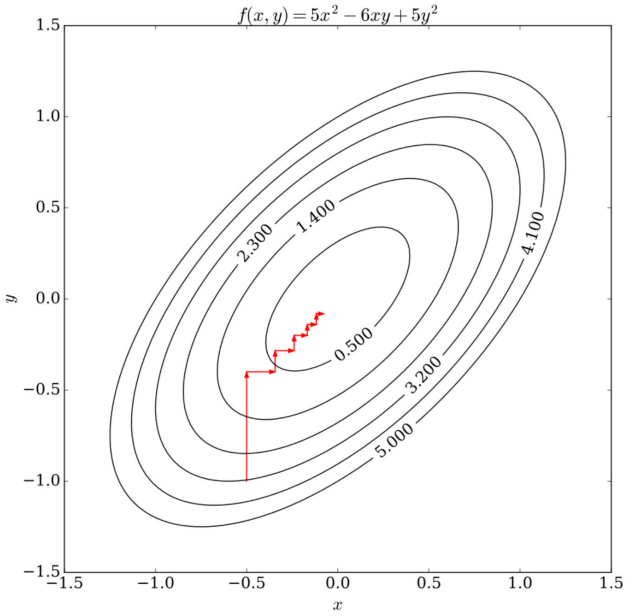


Figure 1-1: An illustration of coordinate descent [60].

majority of components in the variable vector x at their current values and approximately minimizing the objective with respect to the remaining components. Each subproblem is a lower-dimensional, even scalar, minimization problem, making it typically more manageable than the full problem [62]. Figure 1-1 shows an illustration of this method.

In this thesis, we consider the block coordinate descent approach which is a variant of coordinate descent. We focus on the problem

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in X, \end{array}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ is a differentiable convex function and X is a Cartesian product of closed convex sets

$$X = X_1 \times X_2 \times \cdots \times X_m,$$

where X_i is a subset of $\mathbb{R}^{n_i}$. The vector x is partitioned as

$$x = (x^1, x^2, \cdots, x^m),$$

where each x^i is a "block component" of x that is constrained to be in X_i. The block coordinate descent method is defined as follow (same with naive coordinate descent in Subsection 3-1-1): given the current iterate $x_k = (x_k^1, x_k^2, \cdots, x_k^m)$, we generate the next iterate $x_{k+1} = (x_{k+1}^1, x_{k+1}^2, \cdots, x_{k+1}^m)$, according to

$$x_{k+1}^i \in \arg\min_{\xi \in X_i} f\left(x_{k+1}^1, \ldots, x_{k+1}^{i-1}, \xi, x_k^{i+1}, \ldots, x_k^m\right), \quad i = 1, \ldots, m; \tag{1-6}$$

where we assume that the preceding minimization has at least one optimal solution [13].

Generally, the coordinate descent method updates decision variables on each coordinate sequentially in a cyclic manner. However, besides block coordinate descent, this method has

various other variants, including Randomized Variants, which randomly select a coordinate for updating, and Greedy Variants, which choose the index that minimizes the objective function the most or selects the direction where the gradient or subgradient has the largest size [55]. Coordinate descent methods find applications in various fields. For instance, in the context of the Lasso problem, Wu et al. [63] demonstrated that the CD algorithm exhibits faster convergence. In the case of solving Support Vector Machine (SVM) problems, Platt et al. [47] proposed an efficient algorithm based on the CD method called Sequential Minimal Optimization (SMO), which boasts reduced solving time and memory usage compared to previously introduced methods.

1-4 Organization of Chapters

The rest of the thesis is organized as follows: Chapter 2 introduces an inverse optimization model and proposes an enhanced model, Kernel Inverse Optimization Machine (KIOM), along with its two variants. Experimental results are presented to demonstrate the model's efficacy in effectively learning complex continuous control tasks. Chapter 3 introduces the Sequential Selection Optimization (SSO) algorithm to address the memory challenges encountered during the training of the KIOM model and provides pertinent experiments to evaluate its performance. Finally, Chapter 4 presents the conclusion and future study.

1-5 Notation

For a non-negative integer $n, \mathbb{R}^n$ and $\mathbb{R}^n_+$ denote the spaces of n-dimensional reals and non-negative reals, respectively. The identity square matrix with dimension n is denoted by I_n. For a symmetric matrix Q, the inequality $Q \succeq 0$ (respectively, $Q \succ 0$) means that Q is positive semi-definite (respectively, positive definite). For a vector v, the inequality $v \geq 0$ (respectively, $v > 0$) means that every single entry of v is nonnegative (respectively, positive). The trace of a matrix Q is denoted as $\mathrm{Tr}(Q)$. Given a vector $x \in \mathbb{R}^n$, we use the shorthand notation $\|x\|_Q^2 := x^\top Q x$. A symmetric matrix is often described by the upper diagonal elements while the lower diagonal elements are replaced by "*". The Frobenius norm of matrix Q is denoted as $\|Q\|_F$ and the Kronecker product is denoted as $\otimes$. The notation Q_{ij} represents the element in the i-th row and j-th column of the matrix Q. $\langle \cdot, \cdot \rangle$ denotes the inner product. Throughout this study, we also reserve the hat notation (e.g., $\hat{s}$) for the objects dependent on data.

Chapter 2

Kernel Inverse Optimization Machine

In this chapter, we introduce an imitation learning model based on inverse optimization and analyze its shortcomings. Subsequently, we propose an improved version called Kernel Inverse Optimization Machine (KIOM) and present two related variants: a simplified version and an extended version. Finally, we conduct a numerical evaluation of the KIOM model and compare its performance with other behavior cloning algorithms.

2-1 Learning for Control: An Inverse Optimization Model

Akhtar et al. [2] propose a learning approach based on inverse optimization to learn the mapping from the input space to the action space. They employ a simple quadratic forward model along with the suboptimality loss to formulate an inverse optimization problem and present a tractable convex reformulation. Finally, the effectiveness of the method is demonstrated through an experiment that mimicked the behavior of a Model Predictive Control (MPC) controller. The proposed method follows the process introduced in Chapter 1.

2-1-1 A quadratic forward model

Akhtar et al. [2] utilize a quadratic objective function for the FOP and assume that its constraints are linear and only parameterized by s:

$$f(s,u,\theta) := \begin{bmatrix} s \\ u \end{bmatrix}^T \theta \begin{bmatrix} s \\ u \end{bmatrix} \tag{2-1}$$

$$\begin{aligned} \mathbf{FOP}(s \mid \theta) := \min_{u} \; & f(s,u,\theta) \\ \text{s.t. } & M(s)u \leq W(s), \end{aligned} \tag{2-2}$$

where $s \in \mathbb{R}^m$, $u \in \mathbb{R}^n$, $M(s) \in \mathbb{R}^{d \times m}$ and $W(s) \in \mathbb{R}^d$. Therefore, it is easy to infer that $\theta \in \mathbb{R}^{(m+n)\times(m+n)}$. However, to ensure the convexity of the forward optimization model, it is

necessary to impose constraints on the range of values for θ:

$$\begin{aligned} &\theta \in \Theta \ , \\ &where\ \Theta = \left\{ \theta = \begin{bmatrix} 0 & \theta_{su} \\ * & \theta_{uu} \end{bmatrix} \,\middle|\, \theta_{uu} \succeq I_n \right\}. \end{aligned} \tag{2-3}$$

2-1-2 Suboptimality loss function

Let $\{(\hat{s}_i, \hat{u}_i)\}_{i=1}^N$ be the dataset and N represents the size of the dataset. When given an exogenous signal $\hat{s}_i$, the optimal solution of the forward optimization problem is

$$\begin{aligned} u_i^* = \arg\min_u \ & f(\hat{s}_i, u, \theta) \\ \text{s.t. } & M(\hat{s}_i)u \leq W(\hat{s}_i). \end{aligned} \tag{2-4}$$

The loss function employed in the model is referred to as the suboptimality loss [2], which is defined as

$$\begin{aligned} \ell_{\text{sub}}(\hat{s}_i, \hat{u}_i) &:= f(\hat{s}_i, \hat{u}_i, \theta) - f(\hat{s}_i, u_i^*, \theta) \\ &= f(\hat{s}_i, \hat{u}_i, \theta) - \min_{u \in \mathbb{U}(\hat{s}_i)} f(\hat{s}_i, u, \theta), \end{aligned} \tag{2-5}$$

where

$$\mathbb{U}(\hat{s}_i) = \{u \in \mathbb{R}^n \mid M(\hat{s}_i)u \leq W(\hat{s}_i)\}. \tag{2-6}$$

The suboptimality loss penalizes the mismatch between the expert and learning agent's actions "nonuniformly" [2]. If $\hat{u}_i \in \mathbb{U}(\hat{s}_i),\ \forall i \in \{1, ..., N\}$, then the suboptimality loss is the same as the absolute sub-optimality loss.

2-1-3 Inverse optimization problem and its tractable reformulation

Then the inverse optimization problem is formulated as

$$\begin{aligned} \min_{\theta} \ & \frac{1}{N} \sum_{i=1}^{N} \left\{ f(\hat{s}_i, \hat{u}_i, \theta) - \min_{u \in \mathbb{U}(\hat{s}_i)} f(\hat{s}_i, u, \theta) \right\} \\ \text{s.t. } & \theta \in \Theta \text{ in (2-3)}. \end{aligned} \tag{2-7}$$

In fact, the inverse optimization problem (2-7) is trying to find an optimal θ^* in the set Θ that minimizes the sum of the loss functions. The inverse optimization problem (2-7) is convex as the objective function is a pointwise maximum of infinitely many linear functions and the constraint is a Linear Matrix Inequality (LMI). To make (2-7) can be solved efficiently, Akhtar et al. [2] reformulate it into a tractable LMI optimization problem.

Theorem 1 (LMI Reformulation [2])**.** *For the feasible set* (2-6) *and hypothesis function* (2-1)*, the inverse optimization problem* (2-7) *is equivalent to*

$$\begin{aligned} \min_{\theta, \lambda_i, \gamma_i} \quad & \frac{1}{N} \sum_{i=1}^{N} \left(f(\hat{s}_i, \hat{u}_i, \theta) + \tfrac{1}{4}\gamma_i + W\left(\hat{s}_i\right)^\top \lambda_i \right) \\ s.t. \quad & \theta \in \Theta \ in\ (2\text{-}3),\ \lambda_i \in \mathbb{R}_+^d,\ \gamma_i \in \mathbb{R}, \quad \forall i \leq N \\ & \begin{bmatrix} \theta_{uu} & M\left(\hat{s}_i\right)^\top \lambda_i + 2\theta_{su}^\top \hat{s}_i \\ * & \gamma_i \end{bmatrix} \succeq 0, \quad \forall i \leq N, \end{aligned} \tag{2-8}$$

where λ_i is the Lagrange multiplier and γ_i is the slack variable.

For more details of this model and its derivation, we refer the reader to the original paper [2] and references therein.

2-1-4 Analysis of the model's limitations

The inverse optimization problem (2-8) poses as a tractable convex optimization problem, for which we can conveniently employ tailored, efficient, off-the-shelf solvers such as MOSEK [5] and SCS [45] for resolution. In the realm of machine learning, when a task is relatively intricate, there is a desire to enhance the model's capacity for better task learning, as otherwise, the risk of underfitting arises. A common approach to achieve it involves employing a mapping function, $\phi(\cdot) : \mathbb{R}^m \mapsto \mathbb{R}^l$, to map the state s into a higher-dimensional feature space, thereby augmenting the model's capacity. Note that as a result, the dimension of θ_{su} becomes $\mathbb{R}^{l\times n}$, while the dimension of θ_{uu} remains unchanged. In numerical experiments, we employed this approach; however, two primary issues were identified.

Increasing computational burden An increased output dimension of the mapping function $\phi(\cdot)$ can enhance the model's capacity. However, this comes at the price of a heavier computational burden, as the model parameters also increase accordingly. Figure 2-1 illustrates the relationship between computation time and the dimensionality of the augmented state $\phi(s)$. In this instance, we selected the first 100 data points from the MuJoCo hopper-expert task [57] within the D4RL dataset [27] as training data (details about the D4RL dataset and MuJoCo environment are elaborated in Section 2-3). The original dimensionality of the state s in the training data is 11. We progressively augmented it to 1000 dimensions, and it is noticeable that as the dimensionality of the mapped state $\phi(s)$ increases, the computational time experiences almost linear growth. Therefore, when employing this inverse optimization model, it is often necessary to make a trade-off between the model capacity and the computational time.

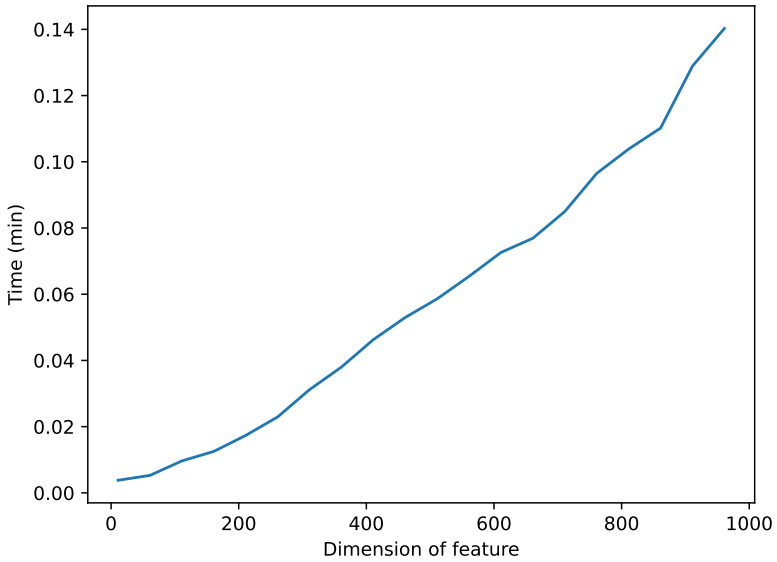


Figure 2-1: The relationship between the solution time of the inverse optimization problem (2-8) and the number of feature dimensions (data size $N = 100$).

Challenges of feature engineering Devising an efficient mapping function $\phi(\cdot)$ and establishing its output dimensionality, referred to as feature engineering, can also be challenging. This process can be time-consuming and labor-intensive. It often requires domain expertise and extensive trial and error to identify and create meaningful features for a given task.

In the next section, we introduce an enhanced model using the kernel method to address these issues.

2-2 An Enhanced Model: Kernel Inverse Optimization Machine

In this section, we propose an enhanced model: Kernel Inverse Optimization Machine (KIOM). Firstly, we present the derivation process of KIOM, which essentially involves modifying the original model and simplifying its Lagrangian dual problem, and compare the KIOM model with the original model. Subsequently, based on prior knowledge of θ_{uu}, we propose a simplified version of the KIOM model. Lastly, we introduce an extended version based on augmented suboptimality loss which offers better geometric and robust interpretations in comparison to suboptimality loss.

2-2-1 Theoretical derivation

To introduce the improved model, certain modifications need to be applied to the original model. Initially, all states $\hat{s}_i$ are replaced by $\phi(\hat{s}_i)$, since during training, augmented features are utilized, except for $\mathbb{U}(\hat{s}_i)$, as the constraints are only associated with the original states $\hat{s}_i$. Secondly, in the inverse optimization problem (2-7), an additional regularization term has been incorporated

$$\begin{aligned} &\min_{\theta} k\|\theta_{uu}\|_F^2 + k\|\theta_{su}\|_F^2 + \tfrac{1}{N}\sum_{i=1}^{N}\left\{ f(\phi(\hat{s}_i),\hat{u}_i,\theta) - \min_{u\in\mathbb{U}(\hat{s}_i)} f(\phi(\hat{s}_i),u,\theta)\right\} \\ &\text{s.t. } \theta\in\Theta \text{ in (2-3).} \end{aligned} \tag{2-9}$$

The hyperparameter k should be positive, so the regularization term here penalizes the magnitude of the parameters θ, which can effectively prevent overfitting when the model's capacity is excessively large (Indeed, a more crucial rationale is to ensure the emergence of inner products of the augmented states in the dual problem, and we will elaborate on this later). Then the equivalent tractable LMI reformulation of the modified inverse optimization problem (2-9) becomes

$$\begin{aligned} \min_{\theta,\lambda_i,\gamma_i} \quad & k\|\theta_{uu}\|_F^2 + k\|\theta_{su}\|_F^2 + \tfrac{1}{N}\sum_{i=1}^{N}\left(f(\phi(\hat{s}_i),\hat{u}_i,\theta) + \tfrac{1}{4}\gamma_i + W\left(\hat{s}_i\right)^\top\lambda_i\right) \\ \text{s.t.} \quad & \theta\in\left\{\begin{bmatrix} 0 & \theta_{su}\\ * & \theta_{uu}\end{bmatrix}\,\middle|\, \theta_{uu}\succeq I_n\right\}, \\ & \lambda_i\in\mathbb{R}_+^d,\ \gamma_i\in\mathbb{R},\quad \forall i\le N \\ & \begin{bmatrix} \theta_{uu} & M\left(\hat{s}_i\right)^\top\lambda_i + 2\theta_{su}^\top\phi(\hat{s}_i)\\ * & \gamma_i\end{bmatrix}\succeq 0,\quad \forall i\le N. \end{aligned} \tag{2-10}$$

The derivation of the LMI reformulation (2-10) follows the same procedure as that of (2-8) and omitted here. Next, we present a more efficient reformulation of (2-10), which remains tractable even when the augmented state is of infinite dimension.

Theorem 2 (Tractable reformulation with kernel method). *For the quadratic hypothesis function* (2-1)*, the modified inverse optimization problem* (2-10) *can be reformulated as*

$$\begin{aligned}\min_{P,\Lambda_i,\Gamma_i} \quad & \tfrac{1}{4k}\left\|\left(\sum_{i=1}^{N}\tfrac{\hat{u}_i\hat{u}_i^T}{N}-\Lambda_i\right)-P\right\|_F^2 \\ & +\tfrac{1}{k}\begin{bmatrix}m_1 & m_2 & ... & m_N\end{bmatrix}(K\otimes I_n)\begin{bmatrix}m_1 & m_2 & ... & m_N\end{bmatrix}^T-\mathrm{Tr}(P) \\ s.t. \quad & P\succeq 0,\ \tfrac{W(\hat{s}_i)}{N}-2M(\hat{s}_i)\Gamma_i\geq 0, \quad \forall i\leq N \\ & \begin{bmatrix}\Lambda_i & \Gamma_i\\ * & \frac{1}{4N}\end{bmatrix}\succeq 0, \quad \forall i\leq N,\end{aligned} \tag{2-11}$$

where $m_i:=\frac{\hat{u}_i^T}{N}-2\Gamma_i^T$ *for* $i\in\{1,...,N\}$*.* K *is the Gram matrix with respect to* $\hat{s}_1,...,\hat{s}_N$*, so* $K\in\mathbb{R}^{N\times N}$ *and* $K_{ij}=\kappa(\hat{s}_i,\hat{s}_j)=\phi(\hat{s}_i)^T\phi(\hat{s}_j)$ *which is the inner product of the augmented states, and decision variables* $P,\Lambda_i\in\mathbb{R}^{n\times n}$*,* $\Gamma_i\in\mathbb{R}^n$ *for* $i\in\{1,...,N\}$*. The expressions of matrix* θ_{uu} *and* θ_{su} *are*

$$\theta_{uu}=-\frac{\left(\sum_{i=1}^{N}\frac{\hat{u}_i\hat{u}_i^T}{N}-\Lambda_i\right)-P}{2k} \tag{2-12}$$

$$\theta_{su}=-\frac{\sum_{i=1}^{N}\phi(\hat{s}_i)m_i}{k}, \tag{2-13}$$

where the weight θ_{su} *is a linear combination of the augmented states.*

Proof. First, let P, $\tilde{\lambda}_i$ and $\begin{bmatrix}\Lambda_i & \Gamma_i\\ * & \alpha_i\end{bmatrix}$ be the Lagrange multiplier associated with the constraints $\theta_{uu}\succeq I_m$, $\lambda_i\in\mathbb{R}_+^d$ and $\begin{bmatrix}\theta_{uu} & M^\top\lambda_i+2\theta_{su}^\top\phi(\hat{s}_i)\\ * & \gamma_i\end{bmatrix}\succeq 0$ respectively in (2-10), where $P,\Lambda_i\in\mathbb{R}^{n\times n}$, $\Gamma_i\in\mathbb{R}^n$, $\tilde{\lambda}_i\in\mathbb{R}^d$ and $\gamma_i\in\mathbb{R}$. Note that for ease of notation, we omit writing the dependency of the matrices M and W on $\hat{s}_i$. Then define the Lagrangian function

$$\begin{aligned}L(\theta_{uu},\theta_{su},\lambda_i,\gamma_i,P,\tilde{\lambda}_i,\Lambda_i,\Gamma_i,\alpha_i)=k\|\theta_{uu}\|_F^2+k\|\theta_{su}\|_F^2+\frac{1}{N}\sum_{i=1}^{N}\Bigg(\overbrace{\hat{u}_i^T\theta_{uu}\hat{u}_i+2\hat{u}_i^T\theta_{su}^T\phi(\hat{s}_i)}^{f(\phi(\hat{s}_i),\hat{u}_i,\theta)}+\frac{1}{4}\gamma_i \\ +W^\top\lambda_i\Bigg)-\mathrm{Tr}(P(\theta_{uu}-I_n))+\sum_{i=1}^{N}\tilde{\lambda}_i^T(-\lambda_i)+\sum_{i=1}^{N}-\mathrm{Tr}\left(\begin{bmatrix}\Lambda_i & \Gamma_i\\ * & \alpha_i\end{bmatrix}\begin{bmatrix}\theta_{uu} & M^\top\lambda_i+2\theta_{su}^\top\phi(\hat{s}_i)\\ * & \gamma_i\end{bmatrix}\right),\end{aligned} \tag{2-14}$$

and the Lagrange dual problem

$$\begin{aligned}\max_{P,\tilde{\lambda}_i,\Lambda_i,\Gamma_i,\alpha_i} \quad & \inf_{\theta_{uu},\theta_{su},\lambda_i,\gamma_i} L(\theta_{uu},\theta_{su},\lambda_i,\gamma_i,P,\tilde{\lambda}_i,\Lambda_i,\Gamma_i,\alpha_i) \\ \text{s.t.} \quad & P\succeq 0,\ \tilde{\lambda}_i\in R_+^d, \quad \forall i\leq N \\ & \begin{bmatrix}\Lambda_i & \Gamma_i\\ * & \alpha_i\end{bmatrix}\succeq 0, \quad \forall i\leq N.\end{aligned} \tag{2-15}$$

When the Lagrangian function (2-14) is at the point of the infimum with respect to $\theta_{uu},\theta_{su},\lambda_i,\gamma_i$, one obtains (we omit writing the dependency of the function L for ease of notation)

$$\frac{\partial L}{\partial\theta_{uu}}=2k\theta_{uu}+\frac{1}{N}\sum_{i=1}^{N}\hat{u}_i\hat{u}_i^T-P+\sum_{i=1}^{N}-\Lambda_i=0\Rightarrow\theta_{uu}=-\frac{(\sum_{i=1}^{N}\frac{\hat{u}_i\hat{u}_i^T}{N}-\Lambda_i)-P}{2k}, \tag{2-16}$$

$$\frac{\partial L}{\partial \theta_{su}} = 2k\theta_{su} + 2\sum_{i=1}^{N}\phi(\hat{s}_i)(\frac{\hat{u}_i^T}{N} - 2\Gamma_i^T) = 0 \Rightarrow \theta_{su} = -\frac{\sum_{i=1}^{N}\phi(\hat{s}_i)(\frac{\hat{u}_i^T}{N} - 2\Gamma_i^T)}{k}, \tag{2-17}$$

$$\frac{\partial L}{\partial \lambda_i} = \frac{W}{N} - \tilde{\lambda}_i - 2M\Gamma_i = 0 \Rightarrow \tilde{\lambda}_i = \frac{W}{N} - 2M\Gamma_i, \tag{2-18}$$

$$\frac{\partial L}{\partial \gamma_i} = \frac{1}{4N} - \alpha_i = 0 \Rightarrow \alpha_i = \frac{1}{4N}. \tag{2-19}$$

By substituting the expressions for $\theta_{uu}, \theta_{su}, \tilde{\lambda}_i$ and α_i into the Lagrange dual problem (2-15) and simplifying it, we obtain

$$\begin{aligned} \min_{P,\Lambda_i,\Gamma_i} \quad & \tfrac{1}{4k}\left\|\left(\sum_{i=1}^{N}\tfrac{\hat{u}_i\hat{u}_i^T}{N} - \Lambda_i\right) - P\right\|_F^2 + \tfrac{1}{k}\sum_{i=1}^{N}\sum_{j=1}^{N}\kappa(\hat{s}_i,\hat{s}_j)m_i m_j^T - \mathrm{Tr}(P) \\ \text{s.t.} \quad & P \succeq 0,\ \tfrac{W(\hat{s}_i)}{N} - 2M(\hat{s}_i)\Gamma_i \geq 0, \quad \forall i \leq N \\ & \begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \frac{1}{4N} \end{bmatrix} \succeq 0, \quad \forall i \leq N. \end{aligned} \tag{2-20}$$

Finally, by proving

$$\begin{aligned} \sum_{i=1}^{N}\sum_{j=1}^{N}\kappa(\hat{s}_i,\hat{s}_j)m_i m_j^T &= \sum_{i=1}^{N}\sum_{j=1}^{N} m_i \left(K_{ij}\otimes I_n\right) m_j^T \\ &= \begin{bmatrix} m_1 & m_2 & ... & m_N \end{bmatrix}\left(K\otimes I_n\right)\begin{bmatrix} m_1 & m_2 & ... & m_N \end{bmatrix}^T \end{aligned}$$

we obtain (2-11). □

From the proof of Theorem 2, it can be observed that problem (2-11) is essentially the dual problem of (2-10). The augmented states $\phi(\hat{s}_i)$ in (2-11) appear in the form of inner products, allowing for a reduction in computational complexity through the use of kernel methods. Additionally, the complexity of this dual problem (2-11) is no longer influenced by the dimensionality of augmented states. Consequently, even a Gaussian kernel that maps the original state into an infinite-dimensional feature space can be employed now, while this is impracticable in the primal problem (2-10).

Up to this point, we can conveniently utilize CVXPY [22] to model the problem (2-11) and employ off-the-shelf solvers to obtain the optimal decision variables Λ_i^*, Γ_i^* and P^*, and then use these optimal variables to recover θ_{uu}^* and θ_{su}^*. Subsequently, when confronted with a previously unseen state s_{new}, solving such a forward optimization problem (2-21) allows the agent to mimic the behavior of the expert:

$$\begin{aligned} u^* = \arg\min_{u} \ & u^T\theta_{uu}^* u + 2\phi^T(s_{new})\theta_{su}^* u \\ \text{s.t.} \ & M(s_{new})u \leq W(s_{new}). \end{aligned} \tag{2-21}$$

It is worth noting that, since θ_{su}^* can be an infinite-dimensional matrix, we may cannot explicitly compute its result. However, in practice, explicit computation of θ_{su}^* is unnecessary. We only need to compute the result of $\phi^T(s_{new})\theta_{su}^*$, and this is feasible:

$$\phi^T(s_{new})\theta_{su}^* = -\tfrac{1}{k}\textstyle\sum_{i=1}^{N}\kappa(s_{new},\hat{s}_i)(\frac{\hat{u}_i}{N} - 2\Gamma_i^*)^T. \tag{2-22}$$

Does the KIOM model address the two deficiencies mentioned in Section 2-1? The first issue arises when increasing the dimensionality of the augmented state, resulting in higher model complexity and, consequently, longer solution times. However, as indicated in Figure 2-2, the improved model (2-11) has effectively addressed the first problem, with solution times showing little variation with changes in the dimensionality of the feature space. Earlier analysis has already highlighted the reason behind this improvement. In the improved model (2-11), augmented states appear in the form of inner products. We can efficiently compute these inner products using the kernel trick, and the number of parameters in the model is no longer dependent on the dimensionality of the feature space. Consequently, the solution time does not increase with an increase in the dimensionality of the feature space.

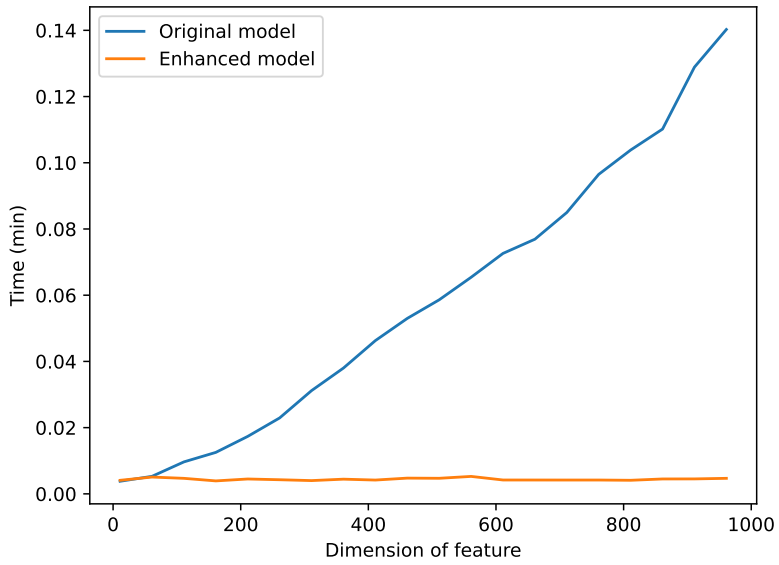


Figure 2-2: The relationship between the solution time of two optimization problems (2-8), (2-11) and the number of feature dimensions (data size $N = 100$).

The second issue is that designing the mapping function $\phi(\cdot)$ often requires domain knowledge relevant to the task, along with a considerable amount of tuning. In the original model, regardless of how efficient the mapping function is, the output space is always finite-dimensional. However, in the improved model, we can directly employ a Gaussian kernel mapping function to map states into an infinite-dimensional space. Generally speaking, the infinite-dimensional features are more representative than the finite-dimensional ones. Although there is no guarantee that the Gaussian kernel function is the most suitable, the Gaussian kernel is an adequate general-purpose kernel function. Additionally, subsequent experiments have demonstrated the effectiveness of the Gaussian kernel.

2-2-2 Extra assumption: a simpler version

In Model (2-11), θ_{uu} is an $n \times n$ decision variable. However, in many experimental tests, we observed that the ultimately solved θ_{uu} is an identity matrix. This is because many experts, when executing their control policy, apply the same penalty coefficient to each dimension of the action or do not penalize the action effort at all. For example, in the Gymnasium MuJoCo

environments, the reward settings for all tasks impose the same penalty on each dimension of the action: $ctrl\_cost\_weight \times sum(action^2)$, where $ctrl\_cost\_weight$ is a parameter set for the control and has a default value of 1e-4 [59]. Consequently, experts trained based on such reward settings tend to uniformly reduce the control effort for each dimension, rather than deliberately decreasing the control effort for a specific dimension. Therefore, assuming $\theta_{uu} = I_n$ as prior knowledge can reduce the model complexity and hypothetically expedite the solution speed.

Corollary 1 (A simpler model)**.** *With the assumption $\theta_{uu} = I_n$, problem* (2-10) *can be simplified to*

$$\begin{array}{ll} \min\limits_{\theta_{su},\gamma_i,\lambda_i} & k\|\theta_{su}\|_F^2 + \frac{1}{N}\sum_{i=1}^N \left(2\hat{u}_i^T\theta_{su}^T\phi(\hat{s}_i) + \frac{1}{4}\gamma_i + W\left(\hat{s}_i\right)^\top \lambda_i\right) \\ s.t. & \lambda_i \in \mathbb{R}_+^d,\ \gamma_i \in \mathbb{R}, \quad \forall i \leq N \\ & \begin{bmatrix} I_n & M\left(\hat{s}_i\right)^\top \lambda_i + 2\theta_{su}^\top\phi(\hat{s}_i) \\ * & \gamma_i \end{bmatrix} \succeq 0, \quad \forall i \leq N, \end{array} \tag{2-23}$$

then, its corresponding Lagrangian dual problem can be simplified to

$$\begin{array}{ll} \min\limits_{\Lambda_i,\Gamma_i} & \begin{bmatrix} m_1 & m_2 & ... & m_N \end{bmatrix} \frac{K\otimes I}{k} \begin{bmatrix} m_1 & m_2 & ... & m_N \end{bmatrix}^T + \sum_{i=1}^N \mathrm{Tr}(\Lambda_i) \\ s.t. & \frac{W(\hat{s}_i)}{N} - 2M(\hat{s}_i)\Gamma_i \geq 0, \quad \forall i \leq N \\ & \begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \frac{1}{4N} \end{bmatrix} \succeq 0, \quad \forall i \leq N, \end{array} \tag{2-24}$$

where $m_i = \frac{\hat{u}_i^T}{N} - 2\Gamma_i^T$ for $i \in \{1, ..., N\}$, and K is the Gram matrix with respect to $\hat{s}_1, ..., \hat{s}_N$, so $K \in \mathbb{R}^{N\times N}$ and $K_{ij} = \kappa(\hat{s}_i, \hat{s}_j)$. The value of θ_{su} can still be represented by (2-13)*.*

Proof. See Appendix A-1. □

It can be observed that model (2-24) is considerably simpler than model (2-11). In the experimental phase of the next section, we utilize the simplified Model (2-24) based on the assumption $\theta_{uu} = I_n$.

Remark 1 (Potential numerical issues and their solutions)**.** *During the experimental phase, we discovered potential numerical issues when directly solving problem* (2-24) *(or problem* (2-11)*). This is because, in the Gymnasium MuJoCo environments, the range of each action dimension is limited to between -1 and 1. However, in problem* (2-24)*, there is a term $\hat{u}_i^T/N$, where N is the dataset size, typically ranging from a few thousand to tens of thousands. This can cause $\hat{u}_i^T/N$ to become a very small quantity, posing potential numerical issues when it is added to or subtracted from other values. Conversely, K/k is often large because the coefficient k is generally set around 10^{-6}, further impacting numerical stability (the same analysis applies to problem* (2-11)*). To address this issue, we can multiply the objective function of the primal problem of problem* (2-24) *(or problem* (2-11)*) by an extra large hyperparameter,* ***scalar****, before deriving the dual problem, which we refer to as the* ***scalar trick****. Based on experience, this* ***scalar*** *is typically chosen in the range of $50N$ to $100N$. It is imperative to underscore the significance of the* ***scalar trick****, as it not only profoundly accelerates the solving speed but also enhances the precision of the results.*

2-2-3 An extension with augmented suboptimality loss

Inspired by the geometry of the consistent cost vectors' inverse optimization set, Zattoni Scroccaro et al. [65] introduce the concept of "incenter", a novel notion similar to the recently proposed circumcenter [15]. Exploring the geometric and robust interpretation of the incenter cost vector, Zattoni Scroccaro et al. [65] formulate corresponding tractable convex representations and, furthermore, suggest a new loss function named Augmented Suboptimality Loss (ASL) as a relaxation of the incenter concept, specifically designed for problems involving inconsistent data. For more details, we refer the reader to the paper [65].

Considering the quadratic objective function (2-1), the Augmented Suboptimality Loss is defined as

$$\ell_{\mathrm{ASL}}(\phi(\hat{s}_i), \hat{u}_i) = f(\phi(\hat{s}_i), \hat{u}_i, \theta) - \min_{u \in \mathbb{U}(\hat{s}_i)} \{f(\phi(\hat{s}_i), u, \theta) - \|\hat{u}_i - u\|_\infty\}, \tag{2-25}$$

where $\mathbb{U}(\hat{s}_i)$ is defined in (2-6). Note that in Augmented Suboptimality Loss, an additional penalty term, $\|\hat{u}_i - u\|_\infty$, for action divergence has been introduced. Using the new loss function, the regularized inverse optimization problem is

$$\begin{aligned} &\min_{\theta} k\|\theta_{uu}\|_F^2 + k\|\theta_{su}\|_F^2 + \tfrac{1}{N}\sum_{i=1}^{N} \ell_{\mathrm{ASL}}(\phi(\hat{s}_i), \hat{u}_i) \\ &\text{s.t. } \theta \in \left\{ \begin{bmatrix} 0 & \theta_{su} \\ * & \theta_{uu} \end{bmatrix} \,\middle|\, \theta_{uu} \succeq 0 \right\}. \end{aligned} \tag{2-26}$$

In the problem (2-26), θ_{uu} is constrained to be positive semidefinite, unlike in equation (2-3), where it is required to be greater or equal to the identity matrix. This modification is made because the infinity norm term in the augmented suboptimality loss (2-25) prevents θ from being trained as a completely zero matrix.

Corollary 2 (LMI Reformulation)**.** *For the feasible set* (2-6)*, hypothesis function* (2-1)*, and loss function* (2-25)*, the inverse optimization problem* (2-26) *is equivalent to*

$$\begin{aligned} \min_{\theta_{uu}, \theta_{su}, t_i, g_{ij}, a_{ij}} \quad & k\,\|\theta_{uu}\|_F^2 + k\|\theta_{su}\|_F^2 + \tfrac{1}{N}\textstyle\sum_{i=1}^{N} t_i & \\ s.t. \quad & g_{ij} \in \mathbb{R}_+^d, \quad t_i \in \mathbb{R}_+ & \forall (i,j) \in [N] \times [2n] \\ & f(\phi(\hat{s}_i), \hat{u}_i, \theta) + a_{ij} + \langle g_{ij}, W(\hat{s}_i)\rangle + \langle \hat{y}_j, \hat{u}_i\rangle \le t_i & \forall (i,j) \in [N] \times [2n] \\ & \begin{bmatrix} \theta_{uu} & 2\theta_{su}^T \phi(\hat{s}_i) + \hat{y}_j + M(\hat{s}_i)^\top g_{ij} \\ * & 4a_{ij} \end{bmatrix} \succcurlyeq 0 & \forall (i,j) \in [N] \times [2n], \end{aligned} \tag{2-27}$$

where, if $j \le n$, $\hat{y}_j$ is a vector of zeros except for the j-th element, which is equal to 1. If $j > n$, $\hat{y}_j$ is a vector of zeros except for the $(j-n)$-th element, which is equal to -1. n is the dimensionality of the action space.

For the proof, we refer to the paper [65], Theorem 4.5, for a more general case. Similarly, by solving the dual problem of (2-27), the augmented states appear in the form of inner products.

Corollary 3 (Tractable reformulation with kernel method)**.** *For the quadratic hypothesis*

function (2-1)*, the extended inverse optimization problem* (2) *can be reformulated as*

$$\begin{aligned}
\min_{\lambda_{ij},\Lambda_{ij},\Gamma_{ij}} \quad & \tfrac{1}{4k}\mathrm{Tr}\left(\sum_{i=1}^{N}\sum_{j=1}^{2n}(\lambda_{ij}\hat{u}_i\hat{u}_i^T-\Lambda_{ij})\sum_{i=1}^{N}\sum_{j=1}^{2n}(\lambda_{ij}\hat{u}_i\hat{u}_i^T-\Lambda_{ij})\right) \\
& +\tfrac{1}{2k}\mathrm{Tr}\left(\sum_{i=1}^{N}\sum_{j=1}^{2n}\sum_{k=1}^{N}\sum_{l=1}^{2n}\kappa(\hat{s}_i,\hat{s}_k)(\lambda_{ij}\hat{u}_i-2\Gamma_{ij})(\lambda_{kl}\hat{u}_k-2\Gamma_{kl})^T\right) \\
& -\sum_{i=1}^{N}\sum_{j=1}^{2n}\lambda_{ij}\hat{y}_j^T\hat{u}_i+\sum_{i=1}^{N}\sum_{j=1}^{2n}2\Gamma_{ij}^T\hat{y}_j \\
s.t. \quad & \lambda_{ij}W(\hat{s}_i)-2M(\hat{s}_i)\Gamma_{ij}\geq 0, \quad \tfrac{1}{N}-\sum_{j=1}^{2n}\lambda_{ij}\geq 0, \quad \lambda_{ij}\in\mathbb{R}_+ \qquad \forall(i,j)\in[N]\times[2n] \\
& \begin{bmatrix}\Lambda_{ij} & \Gamma_{ij}\\ * & \lambda_{ij}/4\end{bmatrix}\succeq 0 \qquad \forall(i,j)\in[N]\times[2n],
\end{aligned} \tag{2-28}$$

where $\kappa(\hat{s}_i,\hat{s}_j)=\phi(\hat{s}_i)^T\phi(\hat{s}_j)$*, which is the inner product of the augmented states, and* $\Lambda_{ij}\in\mathbb{R}^{n\times n}$*,* $\Gamma_{ij}\in\mathbb{R}^n$*, and* $\lambda_{ij}\in\mathbb{R}$ *are the decision variables. The matrices* θ_{uu} *and* θ_{su} *can be written as*

$$\theta_{uu}=-\frac{\sum_{i=1}^{N}\sum_{j=1}^{2n}(\lambda_{ij}\hat{u}_i\hat{u}_i^T-\Lambda_{ij})}{2k} \tag{2-29}$$

$$\theta_{su}=-\frac{\sum_{i=1}^{N}\sum_{j=1}^{2n}\phi(\hat{s}_i)(\lambda_{ij}\hat{u}_i^T-2\Gamma_{ij}^T)}{k}, \tag{2-30}$$

where the weight θ_{su} *is a linear combination of the augmented states.*

Proof. See Appendix A-2. □

It can be observed that, although Augmented Suboptimality Loss provides a better interpretation of robustness, problem (2-28) becomes more complex. Its parameters are not only related to the size of the dataset N but also to the dimensionality of the action space n. Therefore, we did not use this model in the experimental phase, but it is still meaningful to mention this potential direction.

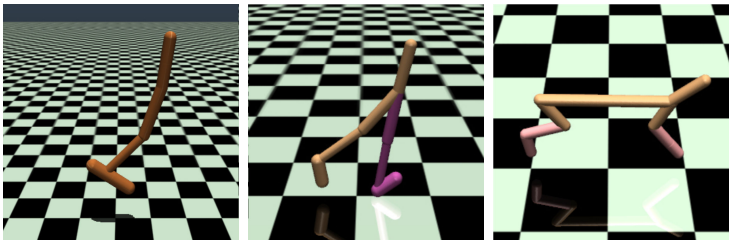
2-3 Numerical Experiments

In this section, we numerically evaluate the simplified version of the KIOM model (2-24). We begin by introducing the experimental task, dataset, and solver. Subsequently, we compare the simulation results with those from related behavior cloning algorithms. Finally, the model parameters are provided in Appendix B-1.

2-3-1 Experimental environment, dataset, and solver

Experimental environment

Gymnasium [59] is a toolkit developed by OpenAI for developing and comparing reinforcement learning algorithms. It provides a variety of environments, from simple toy examples to complex simulations, making it a valuable resource for researchers and developers working on reinforcement learning tasks. To evaluate our model, we measure its performance on the suite of MuJoCo [57] continuous control tasks, interfaced through Gymnasium [59]. In this experiment, we specifically utilize the Hopper-v2, Walker2d-v2, and HalfCheetah-v2 tasks within the Mujoco environments (Figure 2-3).



(a) Hopper (b)Walker2d (c)HalfCheetah

Figure 2-3: Example MuJoCo environments [29]

- **Hopper-v2:** The hopper is a two-dimensional, one-legged figure consisting of four main body parts: the torso at the top, the thigh in the middle, the leg in the bottom, and a single foot on which the entire body rests. The goal is to make hops that move in the forward (right) direction by applying torques on the three hinges connecting the four body parts [59].
- **Walker2d-v2:** This environment builds on the hopper environment by adding another set of legs making it possible for the robot to walk forward instead of hop. The walker is a two-dimensional, two-legged figure that consists of seven main body parts: a single torso at the top (with the two legs splitting after the torso), two thighs in the middle below the torso, two legs in the bottom below the thighs, and two feet attached to the legs on which the entire body rests. The goal is to walk in the forward (right) direction by applying torques on the six hinges connecting the seven body parts [59].
- **HalfCheeta-v2:** The HalfCheetah is a 2-dimensional robot consisting of 9 body parts and 8 joints connecting them (including two paws). The goal is to apply torque on the joints to make the cheetah run forward (right) as fast as possible. The torso and head of the cheetah are fixed, and the torque can only be applied on the other 6 joints over the front and back thighs (connecting to the torso), shins (connecting to the thighs), and feet (connecting to the shins) [59].

Dataset

D4RL

D4RL [27], or Datasets for Deep Data-Driven Reinforcement Learning, is a collection of datasets designed to facilitate research and development in the field of deep reinforcement learning. D4RL aims to provide standardized and diverse datasets that researchers can use to

benchmark and evaluate their algorithms. We tested the algorithm on each task using both **Medium** and **Expert** datasets, resulting in a total of six datasets.

- **Medium:** Medium has 1M samples derived from a policy that is trained to achieve approximately 1/3 the performance of the expert [27].
- **Expert:** Expert has 1M samples from a policy trained to completion with Soft Actor-Critic [27].

Solver

Splitting Conic Solver (SCS) [45] is an efficient open-source optimization solver developed by the Optimization and Systems Theory group at UCLA. Designed to tackle large-scale convex cone problems, particularly those involving linear, quadratic, and second-order cone programming, SCS is known for its speed and scalability. With support for parallel processing, it efficiently handles problems with a high number of variables and constraints. Its open-source nature allows users to modify the source code, making it a flexible choice for researchers and practitioners working on convex optimization problems in diverse fields such as machine learning, control systems, and scientific applications.

While MOSEK [5] is a versatile optimization software package renowned for its efficiency and reliability. Developed by MOSEK ApS, it addresses a broad spectrum of mathematical optimization problems, including linear programming, quadratic programming, conic programming, and semidefinite programming. Widely used in engineering, finance, and data analysis, MOSEK excels in handling large-scale and complex optimization challenges. Its advanced algorithms and interface support for multiple programming languages make it a preferred choice for researchers, engineers, and professionals seeking effective solutions to diverse optimization problems.

In our experiments, we predominantly utilized the SCS solver, as we observed a significantly faster solving speed compared to Mosek. Additionally, we employed CVXPY as the modeling language interface, and the compilation time with the SCS solver was considerably shorter than that with Mosek. Despite SCS's efficiency, Mosek, being a reputable commercial solver, offers higher precision. Therefore, for a small number of small-scale datasets, we also employed Mosek as an alternative solver.

2-3-2 Results

In this evaluation, KIOM is utilized in the simplified version (2-24), incorporating a Gaussian kernel. In each task, the model's performance is assessed through 100 episodes of testing, and the score[1] for KIOM is the average obtained over these 100 episodes. The parentheses following KIOM scores represent the amount of data used.

[1]Regarding the definition of the score for one episode, we recommend readers to consult the official documentation of Gymnasium [59] and the D4RL paper [27].

For comparison, three additional agents are selected in this experiment. The scores of agents **BC(TD3+BC)**[28] and **BC(CQL)**[36] are obtained from two offline reinforcement learning papers. These papers have implemented their respective behavior cloning agents using the corresponding datasets in D4RL, serving as baselines to compare against their proposed offline reinforcement learning algorithms. From the original paper, **BC(TD3+BC)** and **BC(CQL)** are evaluated over 10 seeds and 3 seeds respectively. The **Teacher agent** is the agent responsible for generating the dataset and serves as the target for our imitation learning.

Task	KIOM	BC(TD3+BC)[28]	BC(CQL)[36]	Teacher agent
Hopper-expert	**109.9**(5k)	**111.5**	**109.0**	**108.5**
Hopper-medium	**50.2**(5k)	30.0	29.0	44.3
Walker2d-expert	108.5(10k)	56.0	**125.7**	107.1
Walker2d-medium	**74.6**(5k)	11.4	6.6	62.1
Halfcheetah-expert	84.4(10k)	105.2	**107.0**	88.1
Halfcheetah-medium	**39.0**(5k)	36.6	36.1	**40.7**

Table 2-1: Performance of KIOM, others Behaviour Cloning (BC) implementations and teacher agent on gym domains from D4RL, on the normalized return metric. The numbers inside the parentheses represent the amount of data used, and the score for KIOM in every task is the average score over 100 episodes.

Table 2-1 displays the final experimental results, where KIOM achieves competitive results in four out of the six tasks. In these six tasks, except for a slightly lower score in the Halfcheetah-expert task compared to the teacher agent, KIOM's scores are either close to or higher than those of the teacher agent. This indicates that the enhanced model exhibits strong learning capabilities in complex control tasks. All hyperparameters used in this experiment for KIOM are listed in Appendix B-1.

Chapter 3

Sequential Selection Optimization

This chapter proposes a new algorithm for training kernel inverse optimization machine: *Sequential Selection Optimization*, or *SSO*. Training a kernel inverse optimization machine involves solving a significantly large-scale Semidefinite Programming (SDP) optimization problem. We observed that as the dataset size increases, the memory requirements for solving this problem grow significantly. Table 3-1 below illustrates the relationship between data volume and required memory. When the data volume exceeds ten thousand, most personal computers will encounter insufficient memory issues. Moreover, when the data volume reaches twenty thousand, a staggering 256GB of memory is required. This represents a formidable challenge.

Data size	**Memory (RAM)**
5K	16GB
10K	64GB
15K	128GB
20K	256GB

Table 3-1: The relationship between the dataset size and the required memory on the Halfcheetah-expert task.

Consequently, when dealing with excessively large datasets, the problem can be intractable in terms of memory. To address this issue, SSO breaks this large SDP problem into a series of smaller SDP problems. These smaller SDP problems can be rapidly solved without consuming excessive memory. The memory space needed for SSO is nearly equivalent to the memory required to solve one of these smaller SDP problems. This feature empowers SSO to efficiently address larger-scale optimization problems. The final experiments demonstrate that SSO can quickly converge to the optimum with lower memory overhead.

3-1 A Distributed Algorithm: Sequential Selection Optimization

This section presents the proposed algorithm, SSO. We initially introduce the naive coordinate descent, which is essentially a block coordinate descent method. Subsequently, we prove the convergence of the algorithm. Building upon the naive coordinate descent, we then introduce two heuristic techniques, forming the basis of SSO. In the following section, we experimentally demonstrate how these two techniques can significantly accelerate the convergence speed of the algorithm.

3-1-1 Naive coordinate descent

Model (2-24) exhibits a proportional relationship between the number of parameters and the volume of data. With the addition of each data point $(\hat{s}_i, \hat{u}_i)$, a corresponding parameter set $\{\Lambda_i, \Gamma_i\}$ is introduced. For convenience in expression, we collectively refer to a pair of parameters, Λ_i and Γ_i, as $coordinate\_i := \{\Lambda_i, \Gamma_i\}$.

In this subsection, we introduce a straightforward distributed algorithm, Coordinate Descent (CD), and in the next subsection, we establish its convergence. The CD algorithm is selected here because, in model (2-24), the objective function is quadratic, and each coordinate is coupled with others. However, in the constraints, each coordinate is decoupled. Coordinate descent (CD) can effectively leverage this structure. To facilitate the convergence proof, we first make some modifications to model (2-24):

$$\begin{aligned} \min_{\Lambda_i,\Gamma_i,\Lambda_i^{'}} \quad & \tfrac{1}{k}\sum_{i=1}^{N}\sum_{j=1}^{N}\kappa_{ij}(\tfrac{\hat{u}_i^T}{N}-2\Gamma_i^T)(\tfrac{\hat{u}_j}{N}-2\Gamma_j)+\sum_{i=1}^{N}\mathrm{Tr}(\Lambda_i)+c\sum_{i=1}^{N}\|\Lambda_i-\Lambda_i^{'}\|_F^2 \\ \text{s.t.} \quad & \tfrac{W(\hat{s}_i)}{N}-2M(\hat{s}_i)\Gamma_i\geq 0, \quad \forall i\leq N \\ & \begin{bmatrix}\Lambda_i & \Gamma_i\\ * & \frac{1}{4N}\end{bmatrix}\succeq 0, \quad \forall i\leq N, \end{aligned} \tag{3-1}$$

where a new decision variable $\Lambda_i^{'}$ is introduced, along with an extra proximal term $c\sum_{i=1}^{N}\|\Lambda_i-\Lambda_i^{'}\|_F^2$ $(c>0)$ in the objective function. It is noteworthy that, for ease of observation, we have decomposed the quadratic form matrix term into a summation form and omitted the use of m_i notation. It is straightforward to demonstrate the equivalence between problem (3-1) and problem (2-24), i.e., the optimal solutions for the two problems are identical. In (3-1), the optimal value of $\Lambda_i^{'}$ should always be equal to that of Λ_i, resulting in the proximal term of zero. Consequently, problem (3-1) degenerates into problem (2-24).

Due to the introduction of the new decision variable $\Lambda_i^{'}$, we need to redefine the coordinates in (3-1)

$$coordinate\_i := \begin{cases}\{\Lambda_i,\ \Gamma_i\}, & i=1,...,N\\ \{\Lambda_{i-N}^{'}\}, & i=N+1,...,2N.\end{cases} \tag{3-2}$$

Here, we employ the block coordinate descent method as introduced in Chapter 1, Section 1-3 to optimize the problem (3-1). Specifically, we sequentially group every p coordinates into a block coordinate (assuming N is divisible by p for simplifying analysis), and then iteratively update each block coordinate in a cyclic manner.

Update of the first half block coordinates When updating the first half of the block coordinates, each subproblem shares a similar structure with problem (2-24) but with fewer

parameters. For simplicity, we provide the subproblem corresponding to the first block coordinate (subproblems for other first half block coordinates have a similar form):

$$\begin{aligned}\min_{\Lambda_i,\Gamma_i} \quad & \tfrac{1}{k}\sum_{i=1}^{p}\sum_{j=1}^{p}\kappa_{ij}(\tfrac{\hat{u}_i^T}{N}-2\Gamma_i{}^T)(\tfrac{\hat{u}_j}{N}-2\Gamma_j)+\tfrac{2}{k}\sum_{i=1}^{p}\sum_{j=p+1}^{N}\kappa_{ij}(\tfrac{\hat{u}_i^T}{N}-2\Gamma_i{}^T)(\tfrac{\hat{u}_j}{N}-2\Gamma_j)\\ & +\textstyle\sum_{i=1}^{p}\mathrm{Tr}(\Lambda_i)+c\sum_{i=1}^{p}\|\Lambda_i-\Lambda_i^{'}\|_F^2\\ \text{s.t.} \quad & \tfrac{W(\hat{s}_i)}{N}-2M(\hat{s}_i)\Gamma_i\succeq 0, \quad \forall i\le p\\ & \begin{bmatrix}\Lambda_i & \Gamma_i\\ * & \frac{1}{4N}\end{bmatrix}\succeq 0, \quad \forall i\le p,\end{aligned} \tag{3-3}$$

where, to better distinguish decision variables from constants, we highlight the decision variables.

Update of the second half block coordinates The second half of the block coordinates only include parameters $\Lambda_i^{'}$. When updating these $\Lambda_i^{'}$, as they are unconstrained, it can be straightforwardly proven that their optimal solution is Λ_i, i.e., $\Lambda_i^{'*}=\Lambda_i$. Therefore, when updating these block coordinates, there is no need to solve any optimization problems; we simply assign the values of Λ_i to $\Lambda_i^{'}$ for all i.

Note that this updating approach is essentially a standard block coordinate descent method. In this thesis, we also refer to it as naive coordinate descent.

3-1-2 Convergence analysis

Before proving the convergence of naive coordinate descent, it is necessary to introduce two crucial lemmas and one proposition.

Proposition 1 (Convergence of block coordinate descent [13]). *Let $f:\mathbb{R}^n\mapsto\mathbb{R}$ be convex and differentiable, and let $X=X_1\times X_2\times\cdots\times X_m$, where X_i are closed and convex. Assume further that for each $x=(x^1,\ldots,x^m)\in X$ and i,*

$$f\left(x^1,\ldots,x^{i-1},\xi,x^{i+1},\ldots,x^m\right)$$

viewed as a function of ξ, attains a unique minimum over X_i. Let $\{x_k\}$ be the sequence generated by the block coordinate descent method (1-6)*. Then, every limit point of $\{x_k\}$ minimizes f over X.*

Proposition 1 serves as the cornerstone for proving the convergence of naive coordinate descent. We will verify that problem (3-1) satisfies all its assumptions to demonstrate that the naive coordinate descent method introduced in Subsection 3-1-1 can find its optimal solution.

Lemma 1 (Strict convexity and uniqueness of optimal solutions [11]). *Consider an optimization problem*

$$\begin{aligned}&\min f(x)\\ &s.t.\ x\in\Omega,\end{aligned}$$

where $f:\mathbb{R}^n\to\mathbb{R}$ is strictly convex on Ω and Ω is a convex set. Then the optimal solution (assuming it exists) must be unique.

Proposition 1 imposes the crucial requirement that all subproblems have a unique solution. This constitutes the most important and challenging aspect of the entire proof. Note that when updating the second half of the block coordinates, we have already established the uniqueness of the optimal solution. Therefore, the only thing we need to prove is the uniqueness of the optimal solution for subproblem (3-3). With Lemma 1, we only need to prove the strict convexity of the objective function of the subproblems to demonstrate the uniqueness of the optimal solution.

Lemma 2 (Property of Kronecker product [17])**.** *Let $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times n}$. If λ is an eigenvalue of A with corresponding eigenvector $x \in \mathbb{F}^n$ and if μ is an eigenvalue of B with corresponding eigenvector $y \in \mathbb{F}^m$, then $\lambda\mu$ is an eigenvalue of $A \otimes B$ with corresponding eigenvector $x \otimes y \in \mathbb{F}^{mn}$. The set of eigenvalues of $A \otimes B$ is $\{\lambda_i\mu_j : i = 1, 2, \cdots, n$ and $j = 1, 2, \cdots, m\}$ where the set of eigenvalues of A is $\{\lambda_i : i = 1, 2, \cdots, n\}$ and the set of eigenvalues of B is $\{\mu_j : j = 1, 2, \cdots, m\}$ (including algebraic multiplicities in all three cases). In particular, the set of eigenvalues of $A \otimes B$ is the same as the set of eigenvalues of $B \otimes A$.*

Lemma 1 requires the objective function to be strictly convex, whereas the objective function in subproblem (3-3) is quadratic. Therefore, we utilize Lemma 2 to prove that the quadratic form matrix associated with the objective function is positive definite to prove the strict convexity of the objective function.

We show below that the sequence of iterates generated by the naive coordinate descent introduced in the previous Subsection 3-1-1 is in some sense convergent.

Theorem 3 (Convergence for problem (3-1))**.** *Let x represent all decision variables of problem* (3-1)*, i.e., $x := \{\Lambda_i, \Gamma_i, \Lambda_i^{'}\}_{i=1,\dots N}$, and let $\{x_k\}$ be a sequence of iterates generated by the Naive Coordinate Descent algorithm. Then, if the Gram matrix of the chosen kernel function is positive definite, every limit point of $\{x_k\}$ is an optimal solution of* (3-1)*.*

Proof. We first prove that the objective function of each subproblem, when updated using the naive coordinate descent method, is strictly convex. In fact, as discussed earlier, it suffices to prove that the objective function in problem (3-3) is strictly convex. First, we reformulate problem (3-3) into

$$\begin{aligned}
\min_{\Lambda_i,\Gamma_i} \quad & f(\Lambda_i,\Gamma_i) = \tfrac{4}{k}\begin{bmatrix}\Gamma_1^T & \Gamma_2^T & \dots & \Gamma_p^T\end{bmatrix}(K_{pp}\otimes I_n)\begin{bmatrix}\Gamma_1^T & \Gamma_2^T & \dots & \Gamma_p^T\end{bmatrix}^T + \sum_{i=1}^{p} a_i^T\Gamma_i \\
& + \sum_{i=1}^{p}\mathrm{Tr}(\Lambda_i) + c\sum_{i=1}^{p}\|\Lambda_i - \Lambda_i^{'}\|_F^2 \\
\text{s.t.} \quad & \tfrac{W(\hat{s}_i)}{N} - 2M(\hat{s}_i)\Gamma_i \geq 0, \quad \forall i \leq p \\
& \begin{bmatrix}\Lambda_i & \Gamma_i \\ * & \frac{1}{4N}\end{bmatrix} \succeq 0, \quad \forall i \leq p,
\end{aligned} \tag{3-4}$$

where a_i is a constant vector, and its expression does not matter, so not provided here. K_{pp} is the Gram matrix with respect to $\hat{s}_1, ..., \hat{s}_p$. It can be observed that the decision variables $\{\Lambda_i\}_{i=1,\dots N}$ and $\{\Gamma_i\}_{i=1,\dots N}$ are decoupled from each other. Therefore, to prove the strict convexity of $f(\Lambda_i, \Gamma_i)$, it suffices to prove that

$$f_1(\Gamma_i) = \tfrac{4}{k}\begin{bmatrix}\Gamma_1^T & \Gamma_2^T & \dots & \Gamma_p^T\end{bmatrix}(K_{pp}\otimes I_n)\begin{bmatrix}\Gamma_1^T & \Gamma_2^T & \dots & \Gamma_p^T\end{bmatrix}^T + \sum_{i=1}^{p} a_i\Gamma_i \tag{3-5}$$

and

$$f_2(\Lambda_i) = \sum_{i=1}^{p} \mathrm{Tr}(\Lambda_i) + c\sum_{i=1}^{p} \|\Lambda_i - \Lambda_i^{'}\|_F^2 \tag{3-6}$$

are both strictly convex.

For $f_1(\Gamma_i)$, as assumed, K_{pp} is positive definite. Then based on lemma 2, $K_{pp} \otimes I_n$ is also positive definite. Therefore, we can easily prove $\nabla^2 f_1(\Gamma_i) = \frac{4K_{pp}\otimes I_n}{k} \succ 0$, so $f_1(\Gamma_i)$ is strictly convex. As to $f_2(\Lambda_i)$, each element in matrices Λ_i is mutually decoupled, appearing in the form of a quadratic function (with quadratic coefficients all equal to c). Consequently, it can be easily proved that $f_2(\Lambda_i)$ exhibits strict convexity.

Therefore, the objective function $f(\Lambda_i, \Gamma_i)$ is strictly convex, and the constraints of the subproblems consist of linear inequalities and Linear Matrix Inequalities (LMI). Thereby, their feasible set is convex, and, according to Lemma 1, we can prove that the optimal solution of each subproblem is unique.

The objective function of problem (3-1) is quadratic, and its Gram matrix K is positive semi-definite. Therefore, it is straightforward to prove that the objective function is convex and differentiable. Additionally, each block coordinate is decoupled in the constraints, and its constraints are solely comprised of linear inequalities and Linear Matrix Inequalities (LMIs). Consequently, it is also straightforward to prove that the constraint set corresponding to each block coordinate is convex and closed. Therefore, according to Proposition 1, we have established the convergence. □

In the selection of the kernel function, we can opt for the Gaussian kernel, as Lemma 3 below elucidates its positive definiteness of the Gram matrix.

Lemma 3 (Positive definiteness of Gram matrix for Gaussian kernel [53])**.** *Given N arbitrary distinct instances $\{\mathbf{x}_i \mid \mathbf{x}_i \in \mathbf{X}\}_{i=1}^{N}$, and an arbitrary non-zero real value $\sigma \neq 0$, where $\mathbf{X}$ is a nonempty set. The kernel matrix $\mathbf{K} = (\kappa_{i,j})_{N\times N}$ given by*

$$\kappa_{i,j} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

has full rank.

A little discussion

In proving the convergence, we introduced an additional term $c\sum_{i=1}^{N} \|\Lambda_i - \Lambda_i^{'}\|_F^2$ to ensure the strict convexity of the subproblems' objective function. However, this approach can be applied to any differentiable objective function. Even if the objective function is non-convex or concave, we can ensure the modified objective function is strictly convex by increasing the value of c, thereby proving the convergence of the problem.

The conclusion that there is convergence guarantee for any differentiable objective function may seem counterintuitive at first glance, however, this is a specific characteristic of coordinate descent. Other algorithms, such as interior-point methods, require the objective function to be convex and satisfy additional conditions to guarantee a decrease in the objective function value at each iteration, or the value may increase. However, for coordinate descent, regardless

of the form of the objective function, updating a block coordinate never results in a worse solution, meaning the objective function never increases after an update. This property makes the convergence assumptions for coordinate descent more "lenient" compared to other algorithms.

Since there is no strong requirement imposed on the objective function, even a "bad" objective function exhibits convergence. However, compared to a "good" objective function, what is the price for being "bad"? In this thesis, our analysis has only delved into convergence but not convergence rate, and indeed, the price lies within the convergence speed. For these "bad" objective functions, there is often a need for a larger c for correction. This c will also emerge in the convergence rate formula, where a larger c results in a slower convergence speed.

3-1-3 Heuristics for choosing which coordinates to optimize

In the previous algorithm, we updated the parameters of each block coordinate, which contains p coordinates, in a cyclical fashion. However, this approach may not be optimal. In this subsection, we propose a heuristic method: we assess the Karush-Kuhn-Tucker (KKT) conditions for each coordinate and select the p coordinates with the most significant violations to form a new block coordinate to update. Now, let's reconsider problem (2-24). Note that problem (2-24) is the dual problem of problem (2-23), and the Karush-Kuhn-Tucker (KKT) conditions are sufficient and necessary conditions for the optimal solutions of both problems. Here, we enumerate the five KKT conditions that will be employed (For ease of notation, we omit writing the dependency of the matrices M and W on $\hat{s}_i$.):

$$\theta_{su} = -\frac{\sum_{i=1}^{N} \phi(\hat{s}_i)(\frac{\hat{u}_i^T}{N} - 2\Gamma_i^T)}{k} \ (stationarity), \tag{3-7}$$

$$\tilde{\lambda}_i = \frac{W}{N} - 2M\Gamma_i, \ \forall i \leq N \ (stationarity), \tag{3-8}$$

$$\tilde{\lambda_i}^T \lambda_i = 0, \ \forall i \leq N \ (complementary \ slackness), \tag{3-9}$$

$$\mathrm{Tr}\left(\begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \frac{1}{4N} \end{bmatrix} \begin{bmatrix} I_n & M^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{s}_i) \\ * & \gamma_i \end{bmatrix}\right) = 0, \ \forall i \leq N \ (complementary \ slackness), \tag{3-10}$$

$$\lambda_i \in R_+^d, \ \forall i \leq N \ (primal \ feasibility). \tag{3-11}$$

First, we choose coordinate i such that

$$\frac{W}{N} - 2M\Gamma_i > 0. \tag{3-12}$$

Based on KKT condition (3-8), we have

$$\tilde{\lambda}_i > 0. \tag{3-13}$$

Then, based on conditions (3-9) and (3-11), one can obtain

$$\lambda_i = 0. \tag{3-14}$$

Substituting the result (3-14) into condition (3-10) yields

$$\mathrm{Tr}\left(\begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \frac{1}{4N} \end{bmatrix} \begin{bmatrix} I_n & 2\theta_{su}^\top \phi(\hat{s}_i) \\ * & \gamma_i \end{bmatrix}\right) = 0. \tag{3-15}$$

γ_i is the decision variable of problem (2-23). Next, let's solve for its expression. By utilizing the Schur complement, we can prove the following two constraints are equivalent

$$\begin{bmatrix} I_n & M^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{s}_i) \\ * & \gamma_i \end{bmatrix} \succeq 0 \Leftrightarrow \gamma_i \geq \|M^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{s}_i)\|_2^2.$$

Therefore, problem (2-23) can be equivalently expressed as

$$\begin{aligned} \min_{\theta_{su},\gamma_i,\lambda_i} \quad & k\|\theta_{su}\|_F^2 + \tfrac{1}{N}\textstyle\sum_{i=1}^N \left(2\hat{u}_i^T \theta_{su}^T \phi(\hat{s}_i) + \tfrac{1}{4}\gamma_i + W\left(\hat{s}_i\right)^\top \lambda_i\right) \\ \text{s.t.} \quad & \lambda_i \in \mathbb{R}_+^d, \ \gamma_i \in \mathbb{R}, \quad \forall i \leq N \\ & \gamma_i \geq \|M^\top \lambda_i + 2\theta_{su}^\top \hat{s}_i\|_2^2, \quad \forall i \leq N, \end{aligned} \tag{3-16}$$

Here, the variable γ_i is highlighted. It can be easily proven that when γ_i attain its optimal values, the equality in the last constraint should hold: $\gamma_i = \|M^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{s}_i)\|_F^2$. Note that $\lambda_i = 0$ (3-14), then the expression of γ_i is

$$\gamma_i = \|2\theta_{su}^\top \phi(\hat{s}_i)\|_2^2. \tag{3-17}$$

Substituting (3-17) into (3-15):

$$\mathrm{Tr}\left(\begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \frac{1}{4N} \end{bmatrix} \begin{bmatrix} I_n & 2\theta_{su}^\top \phi(\hat{s}_i) \\ * & \|2\theta_{su}^\top \phi(\hat{s}_i)\|_2^2 \end{bmatrix}\right) = 0. \tag{3-18}$$

Note that the expression for θ_{su} is given by (3-7), and it is evident that $\theta_{su}^\top \phi(\hat{s}_i)$ can be computed. Therefore, if the decision variables Λ_i, Γ_i of problem (2-24) attain their optimal values, then for those coordinates satisfying condition (3-12), they should also satisfy condition (3-18). Furthermore, we define

$$\mathrm{kkt\_violator}(i) = \left|\mathrm{Tr}\left(\begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \frac{1}{4N} \end{bmatrix} \begin{bmatrix} I_n & 2\theta_{su}^\top \phi(\hat{s}_i) \\ * & \|2\theta_{su}^\top \phi(\hat{s}_i)\|_2^2 \end{bmatrix}\right)\right|. \tag{3-19}$$

Finally, a heuristic method for selecting which coordinates to update has been established: Given the current values of Λ_i, Γ_i (not necessarily optimal), we choose p coordinates that satisfy condition (3-12) with the maximum KKT violators determined by the function (3-19). We can optimize problem (2-24) directly without introducing the proximal term $c\sum_{i=1}^N \|\Lambda_i - \Lambda_i^{'}\|_F^2$, as using this heuristic has violated the previously established convergence conditions. Of course, we can still optimize problem (3-1) with this heuristic method because it shares the same solution as (2-24). The only additional step is to assign the values of Λ_i to $\Lambda_i^{'}$ after each update.

Certainly, this approach introduces a new issue: coordinates that initially do not satisfy condition (3-12) will never be updated. Therefore, in practical implementation, we additionally randomly select a subset of coordinates for updating to ensure that each coordinate has the opportunity to be updated.

3-1-4 Warm-up trick for improved initial guess

In addressing problem (2-24) (or (3-1)), we typically assume the initial values of its decision variables Λ_i, Γ_i (or $\Lambda_i, \Gamma_i, \Lambda_i^{'}$) to be 0. However, a poor initial guess can lead to slow solver

convergence for the subproblems or even result in numerical instability. Indeed, in our experiments, we observed that setting all decision variables initially to 0 resulted in the solver issuing warnings about inaccurate optimal solutions and significantly prolonged solving times for several subproblems at the beginning (almost ten times longer than usual). Therefore, in this subsection, we propose a method for setting appropriate initial values.

Currently, we possess N data points with the assumption that they can be evenly divided into n subsets, each comprising p data points. Directly solving the large-scale problem (2-24) involving N data points is unfeasible. However, we can rapidly solve a moderate-scale subproblem (3-20), restricted to only p data points. By solving n instances of such problems ($l = 1, ..., n$), we systematically traverse all N data points. Ultimately, we concatenate the optimal solutions of these n subproblems to form an initial estimate for problem (2-24).

$$\begin{aligned} \text{subproblem}(l) := \min_{\Lambda_i, \Gamma_i} \quad & \tfrac{1}{k} \textstyle\sum_{i=(l-1)p+1}^{lp} \sum_{j=(l-1)p+1}^{lp} \kappa_{ij} (\tfrac{\hat{u}_i^T}{N} - 2\Gamma_i^T)(\tfrac{\hat{u}_j}{N} - 2\Gamma_j) \\ & + \textstyle\sum_{i=(l-1)p+1}^{lp} \operatorname{Tr}(\Lambda_i) \\ \text{s.t.} \quad & \tfrac{W(\hat{s}_i)}{N} - 2M(\hat{s}_i)\Gamma_i \geq 0 \\ & \begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \frac{1}{4N} \end{bmatrix} \succeq 0, \quad \forall i \in \{(l-1)p+1, ..., lp\}, \end{aligned} \tag{3-20}$$

It is crucial to emphasize that in subproblem (3-20), we only utilize p data points. However, we do not substitute N with p in (3-20). This is to ensure that the ultimately obtained initial guess is feasible for the original problem (2-24). If we attempt to solve problem (3-1), we can use the same method to get the initial value of Λ_i, Γ_i and assign the obtained values of Λ_i to $\Lambda_i^{'}$.

SSO Algorithm

Finally, SSO is summarized in Algorithm 1, where the function 'WarmUp()' returns the initial guess of the decision variables as described in Subsection 3-1-4 and the function 'HeuristicSelection()' returns the p coordinates that violate the KKT condition the most as derived in Subsection 3-1-3.

Algorithm 1 SSO

1: Initialize variable: $\{\Lambda_i, \Gamma_i\}_{i=1,...,N} \leftarrow \text{WarmUp}(\{\hat{s}_i, \hat{u}_i\}_{i=1,...N})$
2: **for** iteration = 1 to T **do**
3: $\quad \{\Lambda_{a_i}, \Gamma_{a_i}\}_{i=1,...,p} \leftarrow \text{HeuristicSelection}(\{\hat{s}_i, \hat{u}_i\}_{i=1,...N}, \{\Lambda_i, \Gamma_i\}_{i=1,...,N})$
4: $\quad \text{Update}(\{\Lambda_{a_i}, \Gamma_{a_i}\}_{i=1,...,p})$ ▷ Update selected coordinates
5: **end for**

3-2 Numerical Experiments

We present the Sequential Selection Optimization (SSO) algorithm, which builds on the block coordinate descent by incorporating two crucial techniques, Heuristic selection and Warm-up tricks, described in Subsections 3-1-3 and 3-1-4 to increase convergence speed and identify more optimal parameter initializations. In the experiments of this section, we exclusively focus on problem (2-24), that is, without introducing the proximal term.

3-2-1 Performance evaluation

To assess the effectiveness of the SSO algorithm, we conduct a reevaluation of the previous six experiments using the SSO algorithm. All experimental parameters are held constant. In the SSO algorithm, $\frac{N}{2}$ coordinates are updated at each iteration, meaning that each subproblem encompasses half of the dataset.

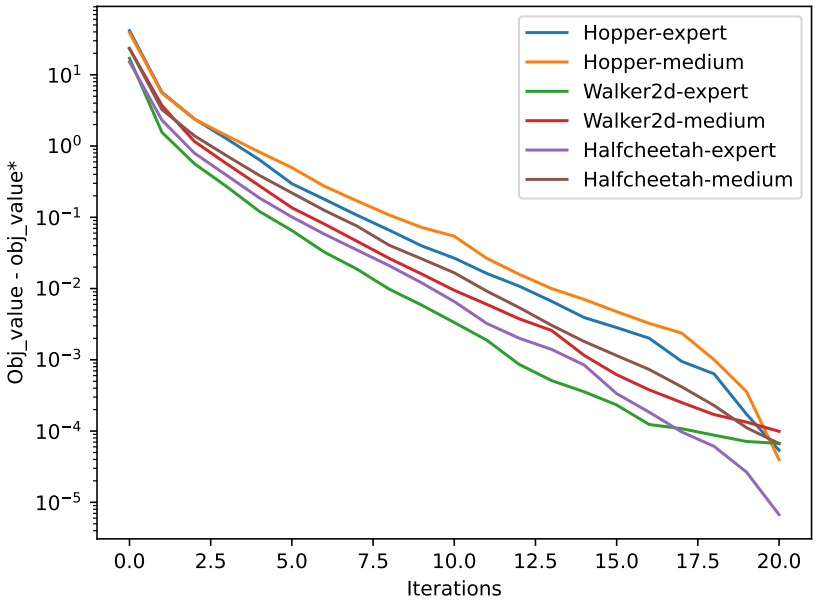


Figure 3-1: Convergence curves on the OpenAI gymnasium continuous control tasks. The horizontal axis represents the number of iterations, while the vertical axis denotes the difference between the current objective function value and the optimal objective function value. By the 20th iteration, these problems have converged closely to their optimal values.

Task	SCS		SSO	
	Obj Value	Score	Obj Value	Score
Hopper-expert	185.219	109.9	185.220	110.2
Hopper-medium	218.761	50.2	218.761	51.8
Walker2d-expert	140.121	108.5	140.121	109.2
Walker2d-medium	151.117	74.6	151.117	74.9
Halfcheetah-expert	165.041	84.4	165.041	83.8
Halfcheetah-medium	188.184	39.0	188.184	39.7

Table 3-2: Final Objective Function Value and Score (Average Return over 100 evaluations) for Splitting Conic Solver (SCS) and Sequential Selection Optimization (SSO). The ultimate Objective Function Values of the two algorithms are nearly identical, yet across the majority of tasks, the SSO exhibits a slightly higher Score compared to the SCS.

Figure 3-1 illustrates the convergence performance of the SSO algorithm across six distinct tasks, with the vertical axis representing the error between the current objective function value and the optimal objective function value in optimization problem (2-24). The SSO algorithm

demonstrates fast convergence speed. By the 10th iteration, the errors for all tasks are below 0.1, and by the 20th iteration, the errors have further diminished to approximately 1e-4 for all tasks. Table 3-2 presents the optimal objective function values and task scores obtained by the centralized algorithm, Splitting Conic Solver (SCS), and the distributed algorithm, Sequential Selection Optimization (SSO), for solving the problem (2-24). In this experiment, SCS is employed to directly address the large-scale problem (2-24). The corresponding solution is evaluated 100 times, and the average is taken as the final score. Meanwhile, SSO addresses problem (2-24) by solving a series of subproblems. After each iteration, we also assess the current solution 100 times, taking the average as the current score. After 20 iterations, there are 20 corresponding scores, and we select the highest score along with the objective function value from the last iteration as the output. It is evident that SSO and SCS yield nearly identical optimal objective function values. However, except for the Halfcheetah-medium task, SSO achieved higher scores across all other tasks.

3-2-2 Ablation studies

We perform ablation studies to understand the contribution of each individual component: Heuristic Coordinates Selection, abbreviated as Heuristic (Subsection 3-1-3), and Warm-Up Trick (Subsection 3-1-4). We present our results in Figure 3-2 in which we compare the performance of removing each component from SSO.

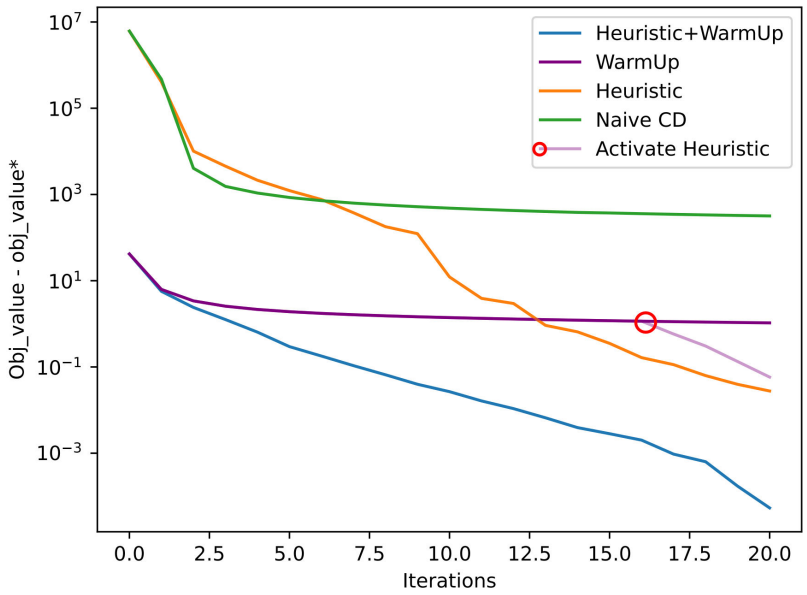


Figure 3-2: Convergence curves on the MuJoCo Hopper task with the first 5k data points from the D4RL Hopper-expert dataset. The vertical axis represents the difference between the current objective function value and the optimal value. Sequential Selection Optimization (blue) exhibits the fastest convergence rate.

We use the Hopper task as the testing task and employ the first 5000 data points from the D4RL Hopper-expert dataset as the training data. The block coordinate for each iteration consists of 2500 coordinates ($p = 2500$). When employing the Warm-Up Trick, we partition the data into two equal parts, solving two subproblems (3-20) each with a dataset size of

2500. Therefore, the computational time required for the Warm-Up Trick is approximately equivalent to the time needed for two iterations of the SSO algorithm. In Figure 3-2, we present the results of 20 iterations, with the vertical axis representing the difference between the current objective function value and the optimal value.

It is evident that both Heuristic Coordinates Selection and the Warm-Up Trick have significantly accelerated the algorithm. When using the warm-up trick, the initial value of the objective function is markedly reduced. Note that without employing the Warm-Up Trick, the Heuristic (orange) curve decreases to the initial values of the SSO (blue) curve after approximately 10 iterations, whereas using this trick consumes only about the time for two iterations. With Heuristic Coordinates Selection, the error curves descend rapidly. It's worth noting the warm-up curve (purple), which becomes nearly flat after a few iterations. However, at the 17th iteration, when we activate the Heuristic Coordinates Selection method, the curve starts to decrease rapidly.

Chapter 4

Conclusion and Future Study

4-1 Conclusion

The objective of inverse optimization is to infer expert behavior given an observed state. In this thesis, we introduce Kernel Inverse Optimization Machine (KIOM), an inverse optimization model utilizing kernel methods. Additionally, we propose two variants: a simplified version and an extended version, and also provide theoretical derivations for them. In subsequent experiments, we demonstrate that the model performs well in learning complex continuous control tasks. KIOM, in contrast to traditional regression models, incorporates input-action constraints during the training process. Furthermore, KIOM is a convex optimization model, hence it is a white-box model. This characteristic allows for a mature theoretical understanding of its operational mechanisms, distinguishing it from behavior cloning algorithms based on deep learning, which lack such interpretability.

Subsequently, we introduce the Sequential Selection Optimization (SSO) algorithm for training the KIOM model. While using off-the-shelf solvers to directly train KIOM is an alternative choice, the associated memory requirements escalate rapidly with increasing dataset size. SSO addresses this challenge by decomposing the original problem into a series of subproblems, thereby mitigating memory demands. The memory required for the SSO algorithm is nearly the same as the memory needed for solving a single subproblem. Experimental results demonstrate that the SSO algorithm achieves rapid convergence to the optimal solution within a small number of iterations. Notably, SSO is an anytime algorithm, capable of providing a suboptimal solution even during the training process.

4-2 Future Study

In the exposition of the SSO algorithm, we elucidated the naive coordinate descent algorithm in Subsection 3-1-1. To establish its convergence, an additional proximal term was introduced. However, this approach appeared somewhat "cumbersome", and empirical observations re-

vealed a notably slow convergence rate. Subsequently, we introduced two heuristic techniques to expedite the convergence speed. However, the heuristic selection technique among them rendered the previous convergence-proof methodology inapplicable. This is because the new algorithm, SSO, no longer updates block coordinates in a cyclical manner; indeed, in each iteration, we choose p coordinates to form a new block coordinate to update. Under this circumstance, the convergence analysis becomes exceedingly challenging, as it necessitates considerations of the employed Karush-Kuhn-Tucker (KKT) conditions. Consequently, we resorted to a demonstration of the effectiveness of the SSO algorithm through experimental means. Nevertheless, theoretically establishing the convergence, and even the convergence rate, of the SSO algorithm is not unattainable. Hence, this constitutes a promising avenue for future research.

Appendix A

Proofs

A-1 Corollary 1

Proof. When $\theta_{uu} = I_n$, the quadratic objective function (2-1) becomes

$$f_2(s, u, \theta_{su}) := 2u^T \theta_{su}^T s, \tag{A-1}$$

where the term $u^T u$ is omitted as it is a constant term. The inverse optimization problem (2-7) becomes

$$\min_{\theta_{su}} \frac{1}{N} \sum_{i=1}^{N} \left\{ f_2\left(\phi(\hat{s}_i), \hat{u}_i, \theta_{su}\right) - \min_{u \in \mathbb{U}(\hat{s}_i)} f_2\left(\phi(\hat{s}_i), u, \theta_{su}\right) \right\}, \tag{A-2}$$

which is an unconstrained problem, and its equivalent tractable LMI reformulation becomes (2-23). The derivation of the LMI reformulation (2-23) follows the same procedure as that of (2-8) and omitted here. For more details about the derivation, we refer the reader to the paper [2].

Similarly, let $\tilde{\lambda}_i$ and $\begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \alpha_i \end{bmatrix}$ be the Lagrange multiplier. Then the Lagrangian function is

$$\begin{aligned} L(\theta_{su}, \lambda_i, \gamma_i, \tilde{\lambda}_i, \Lambda_i, \Gamma_i, \alpha_i) =& k\|\theta_{su}\|_F^2 + \frac{1}{N} \sum_{i=1}^{N} \left(2\hat{u}_i^T \theta_{su}^T \phi(\hat{s}_i) + \frac{1}{4}\gamma_i + W\left(\hat{s}_i\right)^\top \lambda_i \right) \\ &+ \sum_{i=1}^{N} \tilde{\lambda}_i^T (-\lambda_i) + \sum_{i=1}^{N} -\mathrm{Tr}\left(\begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \alpha_i \end{bmatrix} \begin{bmatrix} I_n & M\left(\hat{s}_i\right)^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{s}_i) \\ * & \gamma_i \end{bmatrix} \right) \end{aligned} \tag{A-3}$$

and the Lagrange dual problem is

$$\begin{array}{ll} \max\limits_{\tilde{\lambda}_i,\Lambda_i,\Gamma_i,\alpha_i} & \inf\limits_{\theta_{su},\lambda_i,\gamma_i} L(\theta_{su},\lambda_i,\gamma_i,\tilde{\lambda}_i,\Lambda_i,\Gamma_i,\alpha_i) \\ \text{s.t.} & \tilde{\lambda}_i \in R_+^{2n}, \quad \forall i \leq N \\ & \begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \alpha_i \end{bmatrix} \succeq 0, \quad \forall i \leq N. \end{array} \tag{A-4}$$

When the Lagrangian function (A-3) is at the point of the infimum with respect to $\theta_{su}, \lambda_i, \gamma_i$, one obtains (we omit writing the dependency of the function L for ease of notation):

$$\frac{\partial L}{\partial \theta_{su}} = 2k\theta_{su} + 2\sum_{i=1}^{N} \phi(\hat{s}_i)(\frac{\hat{u}_i^T}{N} - 2\Gamma_i^T) = 0 \Rightarrow \theta_{su} = -\frac{\sum_{i=1}^{N} \phi(\hat{s}_i)(\frac{\hat{u}_i^T}{N} - 2\Gamma_i^T)}{k} \tag{A-5}$$

$$\frac{\partial L}{\partial \lambda_i} = \frac{W}{N} - \tilde{\lambda}_i - 2M\Gamma_i = 0 \Rightarrow \tilde{\lambda}_i = \frac{W}{N} - 2M\Gamma_i \tag{A-6}$$

$$\frac{\partial L}{\partial \gamma_i} = \frac{1}{4N} - \alpha_i = 0 \Rightarrow \alpha_i = \frac{1}{4N} \tag{A-7}$$

Finally, by substituting the expressions for $\theta_{su}, \tilde{\lambda}_i$ and α_i into the Lagrange dual problem (A-4) and simplifying, we can attain (2-24). □

A-2 Corollary 3

Proof. Let $\tilde{g}_{ij}, \tilde{t}_i, \lambda_{ij}$ and $\begin{bmatrix} \Lambda_{ij} & \Gamma_{ij} \\ * & \alpha_{ij} \end{bmatrix}$ be the Lagrange multiplier. Then the Lagrangian function is

$$\begin{aligned} L(\theta_{uu},\theta_{su},t_i,g_{ij},a_{ij},\tilde{g}_{ij},\tilde{t}_i,\lambda_{ij},\Lambda_{ij},\Gamma_{ij},\alpha_{ij}) &= k\|\theta_{uu}\|_F^2 + k\|\theta_{su}\|_F^2 + \frac{1}{N}\sum_{i=1}^{N} t_i + \sum_{i=1}^{N}\sum_{j=1}^{2n} \tilde{g}_{ij}^T(-g_{ij}) \\ + \sum_{i=1}^{N} \tilde{t}_i(-t_i) + \sum_{i=1}^{N}\sum_{j=1}^{2n} \lambda_{ij} \left(\hat{u}_i^T \theta_{uu} \hat{u}_i + 2\hat{u}_i^T \theta_{su}^T \phi(\hat{s}_i) + a_{ij} + \langle g_{ij}, W(\hat{s}_i)\rangle + \langle \hat{y}_j, \hat{u}_i\rangle - t_i \right) \\ + \sum_{i=1}^{N}\sum_{j=1}^{2n} -\mathbf{Tr}\left(\begin{bmatrix} \Lambda_{ij} & \Gamma_{ij} \\ * & \alpha_{ij} \end{bmatrix} \begin{bmatrix} \theta_{uu} & 2\theta_{su}^T\phi(\hat{s}_i) + \hat{y}_j + M(\hat{s}_i)^\top g_{ij} \\ * & 4a_{ij} \end{bmatrix} \right) \end{aligned} \tag{A-8}$$

and the Lagrange dual problem is

$$\begin{array}{ll} \max\limits_{\tilde{g}_{ij},\tilde{t}_i,\lambda_{ij},\Lambda_{ij},\Gamma_{ij},\alpha_{ij}} & \inf\limits_{\theta_{uu},\theta_{su},t_i,g_{ij},a_{ij}} L(\theta_{uu},\theta_{su},t_i,g_{ij},a_{ij},\tilde{g}_{ij},\tilde{t}_i,\lambda_{ij},\Lambda_{ij},\Gamma_{ij},\alpha_{ij}) \\ \text{s.t.} & \tilde{g}_{ij} \geq 0, \ \tilde{t}_i \geq 0, \quad \forall (i,j) \in [N] \times [2n] \\ & \lambda_{ij} \geq 0, \ \begin{bmatrix} \Lambda_{ij} & \Gamma_{ij} \\ * & \alpha_{ij} \end{bmatrix} \succeq 0, \quad \forall (i,j) \in [N] \times [2n]. \end{array} \tag{A-9}$$

When the Lagrangian function (A-8) is at the point of the infimum with respect to $\theta_{uu}, \theta_{su}, t_i, g_{ij}, a_{ij}$,

one obtains (we omit writing the dependency of the function L for ease of notation):

$$\frac{\partial L}{\partial \theta_{uu}} = 2k\theta_{uu} + \sum_{i=1}^{N}\sum_{j=1}^{2n} \lambda_{ij}\hat{u}_i\hat{u}_i^T + \sum_{i=1}^{N}\sum_{j=1}^{2n} -\Lambda_{ij} = 0 \Rightarrow \theta_{uu} = -\frac{\sum_{i=1}^{N}\sum_{j=1}^{2n}(\lambda_{ij}\hat{u}_i\hat{u}_i^T - \Lambda_{ij})}{2k} \tag{A-10}$$

$$\begin{aligned} \frac{\partial L}{\partial \theta_{su}} &= 2k\theta_{su} + \sum_{i=1}^{N}\sum_{j=1}^{2n} 2\lambda_{ij}\phi(\hat{s}_i)\hat{u}_i^T + \sum_{i=1}^{N}\sum_{j=1}^{2n} -4\phi(\hat{s}_i)\Gamma_{ij}^T = 0 \\ &\Rightarrow \theta_{su} = -\frac{\sum_{i=1}^{N}\sum_{j=1}^{2n}\phi(\hat{s}_i)(\lambda_{ij}\hat{u}_i^T - 2\Gamma_{ij}^T)}{k} \end{aligned} \tag{A-11}$$

$$\frac{\partial L}{\partial t_i} = \frac{1}{N} - \tilde{t}_i - \sum_{j=1}^{2n}\lambda_{ij} = 0 \Rightarrow \tilde{t}_i = \frac{1}{N} - \sum_{j=1}^{2n}\lambda_{ij} \tag{A-12}$$

$$\frac{\partial L}{\partial g_{ij}} = -\tilde{g}_{ij} + \lambda_{ij}W(\hat{s}_i) - 2M(\hat{s}_i)\Gamma_{ij} = 0 \Rightarrow \tilde{g}_{ij} = \lambda_{ij}W(\hat{s}_i) - 2M(\hat{s}_i)\Gamma_{ij} \tag{A-13}$$

$$\frac{\partial L}{\partial a_{ij}} = \lambda_{ij} - 4\alpha_{ij} = 0 \Rightarrow \alpha_{ij} = \frac{\lambda_{ij}}{4} \tag{A-14}$$

Finally, by substituting the expressions for $\theta_{uu}, \theta_{su}, \tilde{t}_i, \tilde{g}_{ij}$ and α_{ij} into the Lagrange dual problem (A-9) and simplifying, we can attain (2-28). □

Appendix B

Hyperparameters

B-1 Hyperparameters of KIOM

Environment	Dataset	k	scalar
Hopper-expert	0:5k	1e-6	200N
Hopper-medium	0:5k	1e-6	200N
walker2d-expert	0:5k+10k:15k	1e-5	50N
walker2d-medium	15k:20k	1e-5	50N
Halfcheetah-expert	325k:330k+345k:350k	5e-6	50N
Halfcheetah-medium	10k:15k	5e-6	50N

Table B-1: KIOM Environment Specific Parameters (N: the size of the dataset)

Bibliography

[1] Ravindra K Ahuja and James B Orlin. Inverse optimization. *Operations research*, 49(5):771–783, 2001.

[2] Syed Adnan Akhtar, Arman Sharifi Kolarijani, and Peyman Mohajerin Esfahani. Learning for control: An inverse optimization approach. *IEEE Control Systems Letters*, 6:187–192, 2021.

[3] Maruan Al-Shedivat, Andrew Gordon Wilson, Yunus Saatchi, Zhiting Hu, and Eric P Xing. Learning scalable deep kernels with recurrent structure. *The Journal of Machine Learning Research*, 18(1):2850–2886, 2017.

[4] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Learning separable fixed-point kernels for deep convolutional neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1065–1069. IEEE, 2016.

[5] Mosek ApS. Mosek optimization toolbox for matlab. *User's Guide and Reference Manual, Version*, 4:1, 2019.

[6] Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950.

[7] Anil Aswani, Zuo-Jun Shen, and Auyon Siddiq. Inverse optimization with noisy data. *Operations Research*, 66(3):870–892, 2018.

[8] Turgay Ayer. Inverse optimization for assessing emerging technologies in breast cancer screening. *Annals of operations research*, 230:57–85, 2015.

[9] Aaron Babier, Timothy CY Chan, Taewoo Lee, Rafid Mahmood, and Daria Terekhov. An ensemble learning framework for model fitting and evaluation in inverse linear optimization. *Informs Journal on Optimization*, 3(2):119–138, 2021.

[10] Gaston Baudat and Fatiha Anouar. Generalized discriminant analysis using a kernel approach. *Neural computation*, 12(10):2385–2404, 2000.

[11] Aharon Ben-Tal and Arkadi Nemirovski. Lectures on modern convex optimization (2020). *SIAM, Philadelphia, PA. Google Scholar Google Scholar Digital Library Digital Library*, 2021.

[12] Clifford Bergman. *Universal algebra: Fundamentals and selected topics*. CRC Press, 2011.

[13] Dimitri Bertsekas. *Convex optimization algorithms*. Athena Scientific, 2015.

[14] Dimitris Bertsimas, Vishal Gupta, and Ioannis Ch Paschalidis. Data-driven estimation in equilibrium using inverse optimization. *Mathematical Programming*, 153:595–633, 2015.

[15] Omar Besbes, Yuri Fonseca, and Ilan Lobel. Contextual inverse optimization: Offline and online learning. *Operations Research*, 2023.

[16] John R Birge, Ali Hortaçsu, and J Michael Pavlin. Inverse optimization for the recovery of market structure from market outcomes: An application to the miso electricity market. *Operations Research*, 65(4):837–855, 2017.

[17] Bobbi Jo Broxson. The kronecker product. 2006.

[18] Timothy CY Chan, Rafid Mahmood, and Ian Yihang Zhu. Inverse optimization: Theory and applications. *Operations Research*, 2023.

[19] Dexiong Chen, Laurent Jacob, and Julien Mairal. Biological sequence modeling with convolutional kernel networks. *Bioinformatics*, 35(18):3294–3302, 2019.

[20] Lu Chen, Yuyi Chen, and André Langevin. An inverse optimization approach for a capacitated vehicle routing problem. *European Journal of Operational Research*, 295(3):1087–1098, 2021.

[21] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

[22] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[23] András Faragó, Áron Szentesi, and Balázs Szviatovszki. Inverse optimization in high-speed networks. *Discrete Applied Mathematics*, 129(1):83–98, 2003.

[24] Ricardo Fernández-Blanco, Juan Miguel Morales, and Salvador Pineda. Forecasting the price-response of a pool of buildings via homothetic inverse optimization. *Applied Energy*, 290:116791, 2021.

[25] Ricardo Fernández-Blanco, Juan Miguel Morales, Salvador Pineda, and Álvaro Porras. Inverse optimization with kernel regression: Application to the power forecasting and bidding of a fleet of electric vehicles. *Computers & Operations Research*, 134:105405, 2021.

[26] Marguerite Frank, Philip Wolfe, et al. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

[27] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

[28] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

[29] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[30] Daniel Gabay and Bertrand Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & mathematics with applications*, 2(1):17–40, 1976.

[31] Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, and Mark Crowley. Reproducing kernel hilbert space, mercer's theorem, eigenfunctions, nystr\" om method, and use of kernels in machine learning: Tutorial and survey. *arXiv preprint arXiv:2106.08443*, 2021.

[32] Nicolas Hadjisavvas, Sándor Komlósi, and Siegfried S Schaible. *Handbook of generalized convexity and generalized monotonicity*, volume 76. Springer Science & Business Media, 2006.

[33] Geoffrey E Hinton and Russ R Salakhutdinov. Using deep belief nets to learn covariance kernels for gaussian processes. *Advances in neural information processing systems*, 20, 2007.

[34] Garud Iyengar and Wanmo Kang. Inverse conic programming with applications. *Operations Research Letters*, 33(3):319–330, 2005.

[35] Arezou Keshavarz, Yang Wang, and Stephen Boyd. Imputing a convex objective function. In *2011 IEEE international symposium on intelligent control*, pages 613–619. IEEE, 2011.

[36] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[37] Jonathan Yu-Meng Li. Inverse optimization of convex risk functions. *Management Science*, 67(11):7113–7141, 2021.

[38] Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. *Advances in neural information processing systems*, 27, 2014.

[39] James Mercer. Xvi. functions of positive and negative type, and their connection the theory of integral equations. *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, 209(441-458):415–446, 1909.

[40] Peyman Mohajerin Esfahani and Daniel Kuhn. Data-driven distributionally robust optimization using the wasserstein metric: performance guarantees and tractable reformulations. *Mathematical Programming*, 171(1-2):115–166, 2018.

[41] Peyman Mohajerin Esfahani, Soroosh Shafieezadeh-Abadeh, Grani A Hanasusanto, and Daniel Kuhn. Data-driven inverse optimization with imperfect information. *Mathematical Programming*, 167:191–234, 2018.

[42] Mohammad Reza Mohammadnia-Qaraei, Reza Monsefi, and Kamaledin Ghiasi-Shirazi. Convolutional kernel networks based on a convex combination of cosine kernels. *Pattern Recognition Letters*, 116:127–134, 2018.

[43] Arkadij Semenovič Nemirovskij and David Borisovich Yudin. Problem complexity and method efficiency in optimization. 1983.

[44] Nair K Nikhitha, AL Afzal, and S Asharaf. Deep kernel machines: a survey. *Pattern Analysis and Applications*, 24:537–556, 2021.

[45] Brendan O'Donoghue. Operator splitting for a homogeneous embedding of the linear complementarity problem. *SIAM Journal on Optimization*, 31:1999–2023, August 2021.

[46] Michael Patriksson. *The traffic assignment problem: models and methods*. Courier Dover Publications, 2015.

[47] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[48] Zheng Qu and Peter Richtárik. Coordinate descent with arbitrary sampling i: Algorithms and complexity. *Optimization Methods and Software*, 31(5):829–857, 2016.

[49] Mikael Rönnqvist, Gunnar Svenson, Patrik Flisberg, and Lars-Erik Jönsson. Calibrated route finder: Improving the safety, environmental consciousness, and cost effectiveness of truck routing in sweden. *Interfaces*, 47(5):372–395, 2017.

[50] Javier Saez-Gallego, Juan M Morales, Marco Zugno, and Henrik Madsen. A data-driven bidding model for a cluster of price-responsive consumers of electricity. *IEEE Transactions on Power Systems*, 31(6):5001–5011, 2016.

[51] Craig Saunders, Alexander Gammerman, and Volodya Vovk. Ridge regression learning algorithm in dual variables. 1998.

[52] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International conference on artificial neural networks*, pages 583–588. Springer, 1997.

[53] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[54] Zahed Shahmoradi and Taewoo Lee. Quantile inverse optimization: Improving stability in inverse linear programming. *Operations research*, 70(4):2538–2562, 2022.

[55] Hao-Jun Michael Shi, Shenyinying Tu, Yangyang Xu, and Wotao Yin. A primer on coordinate descent algorithms. *arXiv preprint arXiv:1610.00040*, 2016.

[56] Ilya Sutskever and Geoffrey Hinton. Temporal-kernel recurrent neural networks. *Neural Networks*, 23(2):239–243, 2010.

[57] E Todorov, T Erez, and YT MuJoCo. A physics engine for model-based control. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033.

[58] Luís Torgo. Kernel regression trees. In *Poster papers of the 9th European conference on machine learning (ECML 97)*, pages 118–127. Prague, Czech Republic, 1997.

[59] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

[60] Wikipedia contributors. Coordinate descent — Wikipedia, the free encyclopedia, 2023. [Online; accessed 3-March-2024].

[61] Andrew Wilson and Ryan Adams. Gaussian process kernels for pattern discovery and extrapolation. In *International conference on machine learning*, pages 1067–1075. PMLR, 2013.

[62] Stephen J Wright. Coordinate descent algorithms. *Mathematical programming*, 151(1):3–34, 2015.

[63] Tong Tong Wu and Kenneth Lange. Coordinate descent algorithms for lasso penalized regression. 2008.

[64] Shi Yu, Haoran Wang, and Chaosheng Dong. Learning risk preferences from investment portfolios using inverse optimization. *Research in International Business and Finance*, 64:101879, 2023.

[65] Pedro Zattoni Scroccaro, Bilge Atasoy, and Peyman Mohajerin Esfahani. Learning in inverse optimization: Incenter cost, augmented suboptimality loss, and algorithms. *arXiv e-prints*, pages arXiv–2305, 2023.