Scalable Kernel Inverse Optimization

Youyuan Long
Delft Center for Systems and Control
Delft University of Technology
The Netherlands
Y.Long-2@student.tudelft.nl

Tolga Ok
Delft Center for Systems and Control
Delft University of Technology
The Netherlands
T.Ok@tudelft.nl

Pedro Zattoni Scroccaro
Delft Center for Systems and Control
Delft University of Technology
The Netherlands
P.ZattoniScroccaro@tudelft.nl

Peyman Mohajerin Esfahani
Delft Center for Systems and Control
Delft University of Technology
The Netherlands
P.MohajerinEsfahani@tudelft.nl

Abstract

Inverse Optimization (IO) is a framework to learn the unknown objective function of an expert decision-maker from a past dataset. In this paper, we extend the hypothesis class of IO objective functions to a space of reproducing kernel Hilbert space (RKHS), thereby enhancing its features to an infinite dimensional space. We show that a variant of the representer theorem holds for a specific training loss, hence reformulating the problem to a finite-dimensional convex optimization. To address the scalability issues often encountered with kernel methods, we further propose a Sequential Selection Optimization (SSO) algorithm to efficiently train the proposed Kernel Inverse Optimization (KIO) model. Finally, we demonstrate the generalization capabilities of the proposed KIO model and the effectiveness of the SSO algorithm through learning-from-demonstration tasks within the MuJoCo benchmark.

1 Introduction

Inverse Optimization (IO) is distinct from traditional optimization problems, where we typically seek the optimal decision variables by optimizing an objective function over a set of constraints. In contrast, inverse optimization works "in reverse" by determining the optimization objective given the optimal solution. The inherent assumption in IO is that an agent generates its decision by solving an optimization problem. The assumed optimization problem is called the Forward Optimization Problem (FOP), which is parametric in the exogenous signal $\hat{s}$ with the corresponding optimal solution $\hat{u}$. Therefore, IO aims to deduce the objective function of the FOP (the constraint is assumed to be known in this paper) from a dataset of exogenous signal and decision pairs $\{(\hat{s}_i, \hat{u}_i)\}_{i=1}^N$. Consequently, we can leverage the FOP derived from the expert's dataset by solving it to mimic the expert's behavior when encountering new exogenous signals. IO has garnered widespread attention within several fields, giving rise to numerous studies encompassing both theoretical and applied research. Application domains include vehicle routing [10, 27, 37], transportation system modeling [25, 7], portfolio optimization [22, 36, 21], power systems [8, 14, 28], electric vehicle charging problems [15], network design [13], healthcare problems [5] as well as controller design [2]. For a more detailed discussion on different applications of IO, we refer the readers to the recent survey paper [9].

IO can be categorized into classic IO and data-driven IO. In classic IO, only a single signal-decision pair is considered, where the decision is assumed to be the optimal solution of the FOP (i.e., there is

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

no noise), and different classes of FOPs have been studied, such as linear conic problems [1, 33, 18]. However, in real-world applications, there are usually many observations of signal-decision pairs, and due to the presence of noise, it is usually unreasonable to assume noiseless data (i.e., that all observed decisions are optimal w.r.t. a single true data-generating FOP). Additionally, for complex tasks, the chosen FOP may only approximate the task, not allowing for perfect replication of the observed behavior from the expert. Such cases are called data-driven IO problems. In such scenarios, a loss function is usually used to compute the discrepancy between observed data and the decision generated by the learned FOP. Examples of loss functions include the *2-norm distance loss* [4], *suboptimality loss* [22], *variational inequality loss* [7], *KKT loss* [19] and , *augmented suboptimality loss* [31].

In data-driven IO, the objective function of FOP is typically non-linear with respect to an exogenous signal $\hat{s}$. Hence, classical methods that learn an FOP based on linear function classes may oversimplify the problem and lead to suboptimal solutions. One effective approach to addressing the expressibility issue in data-driven IO problems is the introduction of kernel methods. These methods have been extensively studied within the context of IO [32, 7] and have shown promising results for scaling IO to address practical problems. The application of kernel methods in IO allows for the exploration of a broader class of optimization problems, thereby enhancing the model's ability to generalize from observed decisions to unseen situations. Specifically, using a kernelized approach facilitates the embedding of decision data into a richer feature space, enabling the deduction of an FOP that not only fits the training data but also exhibits strong generalization capabilities.

Contributions. We list the contributions of this work as follows:

(1) Kernelized IO Formulation We propose a novel Kernel Inverse Optimization (KIO) model based on suboptimality loss [22]. The proposed approach leverages kernel methods to enable IO models to operate on infinite-dimensional feature spaces, which allows KIO to outperform existing imitation learning (IL) algorithms on complex continuous control tasks in low-data regimes.

(2) Sequential Selection Optimization Algorithm To address the quadratic computational complexity of the proposed KIO model, we introduce the Sequential Selection Optimization (SSO) algorithm that is inspired by coordinate descent style updates. This algorithm selectively optimizes components of the decision variable, greatly enhancing the efficiency and scalability, while provably converging to the same solution of our proposed KIO model.

(3) Open Source Code To foster reproducibility and further research, we provide an open-source implementation of the proposed KIO model and the SSO algorithm, along with the source code of the experiments conducted in this study.

Notation $\mathbb{R}^n_+$ denotes the space of n-dimensional non-negative vectors. 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). 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 $\|x\|_Q^2 := x^\top Q x$. Symmetric block matrices are 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. The Euclidean inner product of x and y is denoted as $x^\top y$.

2 Preliminaries

2.1 Inverse Optimization

In general, to solve a data-driven IO problem we need to design two crucial components: the Forward Optimization Problem (FOP) and the loss function. Specifically, the FOP corresponds to the optimization problem we will "fit" to the observed dataset $\hat{\mathcal{D}} = \{(\hat{s}_i, \hat{u}_i)\}_{i=1}^N$ (in this paper, we use the "hat" notation (e.g., $\hat{s}$) for the objects dependent on the dataset). That is, our goal is to find a parameter vector $\theta \in \mathbb{R}^p$, such that the FOP

$$\mathrm{FOP}(\theta, \hat{s}) := \min_{u \in \mathbb{U}(\hat{s})} F_\theta(\hat{s}, u), \tag{1}$$

replicates (as close as possible) the data, we do it by minimizing a loss function, akin to classical empirical risk minimization problems. In this work, we focus on quadratic FOPs with linear constraints

and continuous decision variables:

$$F_\theta(\hat{s}, u) := u^\top \theta_{uu} u + 2\phi(\hat{w})^\top \theta_{su} u \quad \text{and} \quad \mathbb{U}(\hat{s}) := \left\{ u \in \mathbb{R}^n : \hat{M}u \leq \hat{W} \right\}, \tag{2}$$

where $\theta := (\theta_{uu}, \theta_{su})$, $\theta_{uu} \in \mathbb{R}^{n\times n}$, $\theta_{su} \in \mathbb{R}^{m\times n}$, $\hat{s} := (\hat{w}, \hat{M}, \hat{W})$, $\hat{w} \in \mathbb{R}^q$, $\hat{M} \in \mathbb{R}^{d\times n}$, $\hat{W} \in \mathbb{R}^d$, and $\phi : \mathbb{R}^q \to \mathbb{R}^m$ is the feature function that maps the feature $\hat{w}$ to a higher-dimensional feature space to enhance the model's capacity.

To learn θ, we solve a regularized loss minimization problem using the Suboptimality Loss [22]

$$\min_{\theta\in\Theta} \; k\mathcal{R}(\theta) + \frac{1}{N}\sum_{i=1}^{N} \max_{u_i \in \mathbb{U}(\hat{s}_i)} \left\{ F_\theta(\hat{s}_i, \hat{u}_i) - F_\theta(\hat{s}_i, u_i) \right\} \tag{3}$$

where $\Theta := \{\theta = (\theta_{uu}, \theta_{su}) : \theta_{uu} \succeq I_n\}$, $\mathcal{R}(\theta) := \|\theta_{uu}\|_F^2 + \|\theta_{su}\|_F^2$, and k is a positive regularization parameter. Notice that the constraint $\theta_{uu} \succeq I_n$ prevents the trivial solution $\theta_{uu} = \theta_{su} = 0$ and guarantees the resulting FOP is a convex optimization problem. Moreover, since F_θ is linear in θ, the optimization program (3) is convex w.r.t. θ, and it can be reformulated from the "minimax" form to a single minimization problem, where the reformulation is based on dualizing the inner maximization problems of (3) and combining the resulting minimization problems.

Proposition 1 (LMI reformulation [2])**.** *For the hypothesis function and feasible set in* (2)*, the optimization program* (3) *is equivalent to*

$$\begin{aligned} \min_{\theta,\lambda_i,\gamma_i} \quad & k\mathcal{R}(\theta) + \frac{1}{N}\sum_{i=1}^{N} \left(F_\theta(\hat{s}_i, \hat{u}_i) + \frac{1}{4}\gamma_i + \hat{W}_i^\top \lambda_i \right) & \\ \text{s.t.} \quad & \theta = (\theta_{uu}, \theta_{su}),\ \theta_{uu} \succeq I_n,\ \lambda_i \in \mathbb{R}^d_+,\ \gamma_i \in \mathbb{R} & \forall i \leq N \\ & \begin{bmatrix} \theta_{uu} & \hat{M}_i^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{w}_i) \\ * & \gamma_i \end{bmatrix} \succeq 0 & \forall i \leq N. \end{aligned} \tag{4}$$

2.2 The Kernel Method

The kernel method is a powerful technique used in machine learning and statistics that exploits the structure of data embedded in a higher dimensional space. The kernel method has found numerous applications, including Support Vector Machines [11], Kernel Principal Component Analysis [29], and Kernel Linear Discriminant Analysis [6]. 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 [32]. The kernel method generalizes the hypothesis of the optimization problem to a nonlinear function class based on a *Reproducing Kernel Hilbert Space* (RKHS) $\mathcal{H}$. We are particularly interested in lifting the original optimization problem of the form $\min_{f\in\mathcal{F}} \mathbb{E}\big[\ell\big(f(\hat{w}), \hat{u}\big)\big]$ to $\min_{f\in\mathcal{H}} \mathbb{E}\big[\ell\big(f(\hat{w}), \hat{u}\big)\big]$, where $\mathcal{F}$ is the original function class, typically set to linear or quadratic functions in the context of IO. By lifting the function class to $\mathcal{H}$, we effectively optimize over *nonlinear* hypotheses.

Given a proper kernel function κ, that is symmetric and positive definite, Moore-Aronszajn's reproducing kernels theory implies that there exists a unique RKHS and a feature map ϕ with the reproducing properties induced by κ, such that $\forall f \in \mathcal{H} : f(w) = \langle f, \kappa(\cdot, w)\rangle_\mathcal{H} = \sum_{i=1}^{\infty} \alpha_i \kappa(w_i, w)$. We therefore define $\phi : \mathbb{X} \to \mathcal{H}$, where $\phi(w) = \kappa(\cdot, w)$ and $w \in \mathbb{X}$, as the feature map and $\kappa : \mathbb{X} \times \mathbb{X} \to \mathbb{R}_+$ as the kernel function. Hence, it follows from the definitions:

$$\kappa(\hat{w}_i, \hat{w}_j) := \langle \phi(\hat{w}_i), \phi(\hat{w}_j)\rangle_\mathcal{H}. \tag{5}$$

Furthermore, the Riesz representation theorem states that there exists a unique $\phi(x) \in \mathcal{H}$ for all $x \in \mathbb{X}$. Recall that the lifted optimization problem is formed over all *nonlinear* hypotheses via $f \in \mathcal{H}$. However, as a result of the reproducing property [30], we can write the optimization problem in the infinite-dimensional inner product space by substituting $x \in \mathbb{X}$ with $\phi(x)$. Hence, the resulting optimization problems have the form shown in (6).

$$\min_{f\in\mathcal{H}} \mathbb{E}\Big[\ell\big(\langle f, \phi(\hat{w})\rangle, \hat{u}\big)\Big] \tag{6}$$

In what follows, we show that the solution of the optimization problem (6) exists and it is finite when the problem is built over a finite dataset $\mathcal{D}$ of size N. We leverage Representer theorem [32] to state the the solution to the lifted problem (6) admits the kernel representation of the form $f^\star(w) = \sum_{i=1}^N \alpha_i \kappa(w_i, w)$. Effectively, this result suggests optimizing over an infinite-dimensional RKHS has a sparse solution over the linear hypotheses.

3 Kernel Inverse Optimization

In this section, we extend the Inverse Optimization (IO) model proposed by [2] to incorporate kernel methods. We consider a hypothesis class of the form in Equation (1). However, kernelizing such hypotheses is not straightforward. Instead, we argue that by kernelizing the optimization problem associated with the loss function described in Equation (3), we can obtain a forward optimization problem (FOP) that minimizes the kernelized objective $F(s, u)$.

We propose to dualize the problem in (3) to obtain the dot product terms $\phi(\hat{w}_i)^\top \phi(\hat{w}_j)$ that we can substitute with the kernel function $\kappa(\hat{w}_i, \hat{w}_j)$. Theorem 1 shows that the optimal solution to θ_{su} of the proposed dual problem is a linear function of N coefficients. Thereby, obtaining a kernelized objective that we can plugin within the FOP.

Theorem 1 (Kernel reformulation)**.** *The Lagrangian dual of the optimization program* (4) *is*

$$\begin{aligned}
\min_{P,\Lambda_i,\Gamma_i} \quad & \frac{1}{4k} \left\| \left(\sum_{i=1}^N \frac{\hat{u}_i \hat{u}_i^\top}{N} - \Lambda_i \right) - P \right\|_F^2 - \operatorname{Tr}(P) \\
& + \frac{1}{k} \sum_{i=1}^N \sum_{j=1}^N \kappa(\hat{w}_i, \hat{w}_j) \left(\frac{\hat{u}_i}{N} - 2\Gamma_i \right)^\top \left(\frac{\hat{u}_j}{N} - 2\Gamma_j \right) \\
\text{s.t.} \quad & P \succeq 0,\ \Lambda_i \succeq 0,\ \Gamma_i \in \mathbb{R}^n && \forall i \leq N \\
& \frac{\hat{W}_i}{N} - 2\hat{M}_i \Gamma_i \geq 0 && \forall i \leq N \\
& \begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \frac{1}{4N} \end{bmatrix} \succeq 0 && \forall i \leq N,
\end{aligned} \tag{7}$$

where $\kappa(\hat{w}_i, \hat{w}_j) = \phi(\hat{w}_i)^\top \phi(\hat{w}_j)$ is the kernel function. The primal variables θ_{uu} and θ_{su} can be recovered using

$$\theta_{uu} = \frac{P - \sum_{i=1}^N \left(\Lambda_i - \frac{\hat{u}_i \hat{u}_i^\top}{N} \right)}{2k} \quad \text{and} \quad \theta_{su} = \frac{\sum_{i=1}^N \phi(\hat{w}_i) \left(2\Gamma_i^\top - \frac{\hat{u}_i^\top}{N} \right)}{k}. \tag{8}$$

Proof. See Appendix A.1. □

Notice that the complexity of the optimization program (7) does not depend on the dimensionality of the feature vector $\phi(\hat{w}_i)$. Consequently, this allows us to use kernels generated even from infinite-dimensional feature spaces, e.g., the Gaussian (a.k.a. radial basis function) kernel $\kappa(\hat{w}_i, \hat{w}_j) = \exp(-\gamma \|\hat{w}_i - \hat{w}_j\|_2^2)$. Program (7) is a convex optimization problem, and can be solved using off-the-shelf solvers, such as MOSEK [3]. Once solved, we can recover the optimal primal variables $\theta_{uu}^\star$ and $\theta_{su}^\star$ from the optimal dual variables $\Lambda_i^\star, \Gamma_i^\star$ and $P^\star$ using (8). Notice that the dimensionality of $\theta_{su}^\star$ depends on ϕ, thus, it can be an infinite dimensional matrix. However, recall that our ultimate goal is to learn an FOP, and to use it to replicate the behavior observed in the data, and combining (8) into (1), we have that, for the signal $\hat{s}_{\text{new}} = (\hat{w}_{\text{new}}, \hat{M}_{\text{new}}, \hat{W}_{\text{new}})$, the resulting FOP is

$$\min_{\hat{M}_{\text{new}} u \leq \hat{W}_{\text{new}}} \frac{1}{2k} u^\top \left(P^\star - \sum_{i=1}^N \left(\Lambda_i^\star - \frac{\hat{u}_i \hat{u}_i^\top}{N} \right) \right) u + \frac{2}{k} \left(\sum_{i=1}^N \kappa(\hat{w}_{\text{new}}, \hat{w}_i) \left(2\Gamma_i^\star - \frac{\hat{u}_i}{N} \right) \right)^\top u \tag{9}$$

which again does not depend on the dimensionality of ϕ, but only depends on the kernel function κ. However, a major difference between solving the IO problem using the kernel reformulation of Theorem 1 and other classical IO approaches (e.g., [22, 2, 31]) is that the resulting FOP (9)

explicitly depends on the entire training dataset $\{(\hat{s}_i, \hat{u}_i)\}_{i=1}^N$. These models are sometimes called *nonparametric models* [23], which means that the number of parameters of the model (in our case, $P^\star$, $\Lambda_i^\star$, and $\Gamma_i^\star$ for all $i \leq N$) depends on the size of the training dataset.

Remark 1 (A potential variant of representer theorem)**.** *In the primal problem* (4)*, a regularized empirical risk loss is optimized over a set of constraints. In the learned objective function of the FOP problem* (9)*, the term related to the features $\hat{w}_i$ (the linear coefficient of the optimizer u) can be represented as a finite linear combination of kernel products evaluated on the input points in the training set data, i.e., $f^*(\cdot) = \sum_{i=1}^N \alpha_i \kappa(\cdot, \hat{w}_i)$. This shows that the trained objective function possesses the properties described by the representer theorem and also indicates the possibility of a variant of the representer theorem.*

In the class of cost function studied in this paper (2), θ_{uu} can be interpreted as a matrix that penalizes the components of the decision vector u. However, for many problems, it turns out that the expert that generates the data equally penalizes each dimension of u, or equivalently, uses $\theta_{uu} = I_n$. This is the case, for instance, in the Gymnasium MuJoCo environments [35], where the reward settings for all tasks have the same penalty on each dimension of the decision vector. Intuitively, this means that the expert trained under such reward settings tries to reduce the magnitude of the decision vector for each dimension uniformly, rather than deliberately decreasing it for a specific dimension. Therefore, assuming $\theta_{uu} = I_n$ as prior knowledge can reduce the model complexity and lead to faster training, without deteriorating the performance of the learned model.

Corollary 1 (Kernel reformulation for $\theta_{uu} = I_n$)**.** *The Lagrangian dual of the optimization program* (4) *with $\theta_{uu} = I_n$ is*

$$\begin{aligned}
\min_{\Lambda_i, \Gamma_i:\, \forall i \in S} \quad & \frac{1}{k} \sum_{i=1}^N \sum_{j=1}^N \kappa(\hat{w}_i, \hat{w}_j) \left(\frac{\hat{u}_i}{N} - 2\Gamma_i\right)^\top \left(\frac{\hat{u}_j}{N} - 2\Gamma_j\right) + \sum_{i=1}^N \mathrm{Tr}(\Lambda_i) & \\
\text{s.t.} \quad & \Lambda_i \succeq 0,\ \Gamma_i \in \mathbb{R}^n & \forall i \in S \\
& \frac{\hat{W}_i}{N} - 2\hat{M}_i \Gamma_i \geq 0 & \forall i \in S \\
& \begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \frac{1}{4N} \end{bmatrix} \succeq 0 & \forall i \in S,
\end{aligned} \tag{10}$$

where $\kappa(\hat{w}_i, \hat{w}_j) = \phi(\hat{w}_i)^\top \phi(\hat{w}_j)$ is the kernel function and $S = \{1, 2, 3, \ldots, N\}$. The primal variable θ_{su} can be recovered using (8)*.*

The proof of Corollary 1 is the same as the proof of Theorem 1 with the assumption that $\theta_{uu} = I_n$, and is therefore omitted here. S represents an index set, where all decision variables whose indices belong to S will be optimized, while decision variables whose indices do not belong to S will retain their original values and be treated as constants. In Corollary 1, all variables will be optimized, hence S is the set comprising natural numbers from 1 to N. The concept of index set S introduced in Problem (10) is to align with the sub-optimization problems based on the coordinate descent method outlined in Section 4.

In the following section, we will focus on algorithms to solve problem (10). However, all ideas also apply to the general problem (7).

4 Sequential Selection Optimization

Solving the kernel IO problem (10) involves optimizing a Semidefinite program (SDP), which may become prohibitively costly if the number of semidefinite constraints and optimization variables becomes too large. In our case, the size of the SPD grows linearly with the size of the training dataset N. For instance, in our experiments, solving (10) with $N = 20000$ using CVXPY [12] requires up to 256GB of memory. Therefore, in this section, we propose a coordinate descent-type algorithm to find an approximate solution to (10) by iteratively optimizing only a subset of the coordinates at each iteration of the algorithm, keeping all other coordinates fixed. We define a pair of variables Λ_i and Γ_i as the i-th coordinate, denoted as $\{\Lambda_i, \Gamma_i\}$. Note that in problem (10), each coordinate is decoupled in the constraints, which is the reason the coordinate descent framework is applicable here. We call this method *Sequential Selection Optimization* (SSO), and we show it in Algorithm 1.

Algorithm 1 Sequential Selection Optimization (SSO)

1: Initialize $\{\Lambda_i, \Gamma_i\}_{i=1}^N$
2: **for** $t = 1, \ldots, T$ **do**
3: Select a batch of p coordinates $S = \{a_i\}_{i=1}^p$, where $a_i \in [1, \ldots, N]$
4: Update $\{\Lambda_{a_i}, \Gamma_{a_i}\}_{i=1}^p$ based on (10) with S
5: **end for**

Here, we explain each step of Algorithm 1: (i) Initialization of the optimization variables. In general, the variables are usually initialized randomly or set to 0 or 1. These methods are simple but can not provide a good initial guess. (ii) Selection of a batch of p coordinates. The most natural approach to choosing p coordinates is to select them cyclically. Alternatively, we can select the coordinates at random at each iteration (not necessarily with equal probability). Lastly, we can choose coordinates greedily, choosing the components corresponding to the greatest descent or the ones with the largest gradient or subgradient at the current iteration [34]. (iii) Solving the KIO subproblem to update the selected coordinates. The mathematical expression of the subproblem is problem (10) with S, where S is a set containing the indices of the coordinates that need to be updated. Note that the coordinates whose indices $\notin S$ are fixed. Therefore, the number of quadratic terms in (10) scales with $|S|^2$ but not N^2.

Next, we propose two heuristics to accelerate the convergence rate of the SSO algorithm. Namely, we present a heuristic method for choosing which coordinates (line 3 of Algorithm 1) to optimize and a warm-up trick to improve the initialization of the optimization variables (line 1 of Algorithm 1).

4.1 Heuristic for choosing coordinates

At each iteration of Algorithm 1, intuitively, the largest improvement will be made by updating the "least optimal" set of p variables. One way to evaluate their degree of suboptimality is to choose the variables with the most significant violation of the Karush-Kuhn-Tucker (KKT) conditions of the primal version of the program (10) (i.e., (4) with $\theta_{uu} = I_n$), which is inspired by the Sequential Minimal Optimization (SMO) method from [26].

Proposition 2. *For optimal decision variables of problem (10), the coordinate, $\{\Lambda_i, \Gamma_i\}$, that satisfies*

$$\frac{\hat{W}_i}{N} - 2\hat{M}_i\Gamma_i > 0 \tag{11}$$

should also satisfy

$$\mathrm{Tr}\left(\begin{bmatrix}\Lambda_i & \Gamma_i \\ * & \frac{1}{4N}\end{bmatrix}\begin{bmatrix}I_n & 2\theta_{su}^\top\phi(\hat{w}_i) \\ * & \|2\theta_{su}^\top\phi(\hat{w}_i)\|_2^2\end{bmatrix}\right) = 0. \tag{12}$$

Proof. See Appendix A.2. □

The Proposition 2 is based on KKT conditions. Based on condition (12), we can define the KKT violation condition

$$\text{kkt\_violator}(i) := \left|\mathrm{Tr}\left(\begin{bmatrix}\Lambda_i & \Gamma_i \\ * & \frac{1}{4N}\end{bmatrix}\begin{bmatrix}I_n & 2\theta_{su}^\top\phi(\hat{w}_i) \\ * & \|2\theta_{su}^\top\phi(\hat{w}_i)\|_2^2\end{bmatrix}\right)\right|. \tag{13}$$

Finally, from the violation condition (13), we can establish the following heuristic method to construct the set S in Algorithm 1: given the current values of $\{\Lambda_i, \Gamma_i\}_{i=1}^N$ at iteration t, we choose the p coordinates that satisfy condition (11) with the maximum KKT violation (13). However, in practice, at each iteration, we additionally select some random coordinates to update. This ensures that all coordinates have the chance to be updated, including those that initially do not meet the criteria specified in condition (11), which would otherwise never be updated.

4.2 Warm-up trick for improved initialization

Another component of Algorithm 1 that may have a great practical impact is how the optimization variables $\{\Lambda_i, \Gamma_i\}_{i=1}^N$ are initialized. A poor initial guess (e.g., $\Lambda_i = \Gamma_i = 0$) can lead to slow solver convergence or even result in numerical instability. In this part, we propose a simple warm-up trick

that leads to a better initialization of the optimization variables. First, we divide the original dataset $\hat{\mathcal{D}}$ into n sub datasets $\hat{\mathcal{D}}_1, \ldots, \hat{\mathcal{D}}_n$ and solve the n small problems (10) with respect to these sub datasets ($N = |\hat{\mathcal{D}}_i|$ and S is the set of indices of all the data in $\hat{\mathcal{D}}_i$). Then, we concatenate the optimal solutions of these n solved small problems to form an initial guess. Here we emphasize that even when the original problem (10) is intractable due to a large training dataset (i.e., large N), each subproblem is tractable for a small enough batch size $|\hat{\mathcal{D}}_i|$, and its solutions are still feasible w.r.t. (10).

5 Numerical Experiments

5.1 Performance evaluation

In this evaluation, KIO is utilized in the simplified version (10), incorporating a Gaussian kernel, tested on continuous control datasets from the D4RL benchmark [16], and we use SSO Algorithm 1 to train the model. In each task, the model's performance is assessed through 100 episodes of testing, and the score[1] for KIO is the average obtained over these 100 episodes. The parentheses following KIO scores represent the amount of data used.

For comparison, four additional agents are selected in this experiment. **IO** is the inverse optimization model without kernel method, introduced in Proposition 1. To demonstrate the effect of the kernel method, both the **KIO** and **IO** models utilize identical datasets. The scores of two behavior cloning agents **BC(TD3+BC)**[17] and **BC(CQL)**[20] 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** is the agent responsible for generating the dataset and serves as the target for our imitation learning.

Table 1: Performance of KIO, IO, two Behavior Cloning (BC) agents, and the Teacher agent on MuJoCo tasks from the D4RL benchmark on the normalized return metric. The numbers inside the parentheses represent the amount of data used, and the score for KIO in every task is the average score over 100 episodes.

Task	KIO	IO	BC(TD3+BC)[17]	BC(CQL)[20]	Teacher
Hopper-expert	**109.9**(5k)	31.8	**111.5**	**109.0**	**108.5**
Hopper-medium	**50.2**(5k)	20.6	30.0	29.0	47.2
Walker2d-expert	108.5(10k)	0.9	56.0	**125.7**	107.1
Walker2d-medium	**74.6**(5k)	0.0	11.4	6.6	68.1
Halfcheetah-expert	84.4(10k)	-1.7	105.2	**107.0**	88.1
Halfcheetah-medium	**39.0**(5k)	-3.1	36.6	36.1	**40.7**

Evaluation for KIO. Table 1 displays the final experimental results, where KIO 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, KIO's scores are either close to or higher than those of the teacher agent. This indicates that the KIO model exhibits strong learning capabilities in complex control tasks. However, without the kernel method, the IO model demonstrates weak learning capabilities, achieving only low scores in the hopper task and failing to learn anything in the other two more challenging tasks. All hyperparameters used in this experiment for KIO are listed in Appendix B.

Evaluation for SSO. Table 2 presents the optimal objective function values and task scores obtained by the centralized algorithm, Splitting Conic Solver (SCS) [24], and the distributed algorithm, Sequential Selection Optimization (SSO), for solving the problem (10). In this experiment, SCS is employed to directly address this large-scale problem (10) with $|S| = N$. The corresponding solution is evaluated 100 times, and the average is taken as the final score. Meanwhile, SSO addresses the large-scale problem 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

[1]Regarding the definition of the score for one episode, we recommend readers to consult the official documentation of Gymnasium [35] and the D4RL paper [16].

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 2: Final Objective Function Value and Score (average return over 100 evaluations) for SCS [24] and SSO (20 iterations for all the tasks) algorithms. 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.

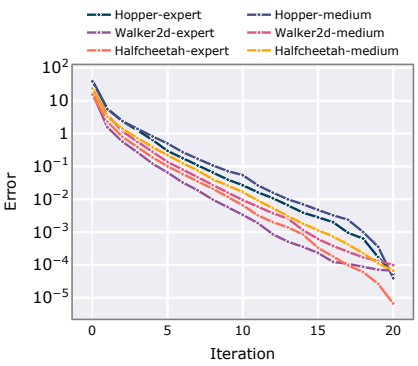

Figure 1: Convergence curves for SSO.

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. Figure 1 displays the convergence performance of the SSO algorithm across six distinct tasks, with the horizontal axis representing the number of iterations and the vertical axis representing the error between the current objective function value and the optimal objective function value (calculated by SCS) in problem (10). 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.

In the previous evaluation of the SSO algorithm, we limited the maximum training data to 10k to ensure that we could directly solve problem (10) without SSO. Thereby, we were able to compare the results with and without the SSO algorithm. However, to further verify the effectiveness of the SSO algorithm, we tested it on a new task (the medium-expert dataset) using 100k data points. At this scale, due to memory limitations, we were unable to directly solve the problem (10) without the SSO algorithm, and thus could only infer the effectiveness of the SSO algorithm from its experimental performance. Table 3 shows the results of the KIO model optimized by the SSO algorithm. The results indicate that the KIO model achieves competitive results and is able to scale to larger data sizes. Similarly, we provide the hyperparameters used in this experiment in Appendix B.

Table 3: Performance of KIO, two Behavior Cloning (BC) agents and the Teacher agent on MuJoCo tasks from D4RL benchmark, on the normalized return metric. The numbers inside the parentheses represent the amount of data used, and the score for KIO in every task is the average score over 100 episodes.

Task	KIO	BC(TD3+BC)	BC(CQL)	Teacher
Hopper-medium-expert	79.6(100k)	89.6	**111.9**	64.8
Walker2d-medium-expert	**100.1**(100k)	12.0	11.3	82.7
Halfcheetah-medium-expert	46.4(100k)	**67.6**	35.8	64.4

5.2 Ablation studies

We perform ablation studies to understand the contribution of each individual component: Heuristic Coordinates Selection, abbreviated as HeuristicSelection (Section 4.1), and Warm-Up Trick (Section 4.2). We present our results in Figure 2 in which we compare the performance of removing each component from SSO (All model hyperparameters remain unchanged as shown in Appendix B).

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 ($|S| = 2500$). When employing the Warm-Up Trick, we partition the data into two equal parts, solving two subproblems (10) each with $|S| = 2500$. 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 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 in problem (10).

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 curve decreases to the initial values of the SSO curve after approximately 10 iterations, whereas using the trick requires only about 2 iterations. With Heuristic Coordinates Selection, the error curves descend rapidly. It's worth noting the WarmUp curve, 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.

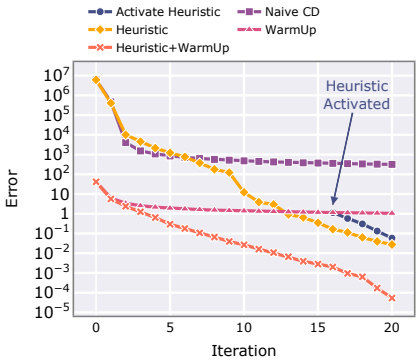

Figure 2: Convergence curves on the MuJoCo Hopper task with the first 5k data points from the D4RL Hopper-expert dataset. The vertical axis represents the difference between the current objective function value and the optimal value. Sequential Selection Optimization (orange) exhibits the fastest convergence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

6 Conclusion and Limitations

We introduce Kernel Inverse Optimization (KIO), an inverse optimization model utilizing kernel methods, its simple variant, along with the theoretical derivations for them. Subsequently, we introduce the Sequential Selection Optimization (SSO) algorithm for training the KIO model, which can address the memory issues by decomposing the original problem into a series of subproblems. Finally, we empirically show that KIO has strong learning capabilities in complex control tasks, and the SSO algorithm can achieve rapid convergence to the optimal solution within a small number of iterations.

One of the limitations of this model is the computational cost of adding a new data point. In that case, we need to use all the training data to calculate the coefficients for the FOP problem (see FOP (9)) for the given point. Therefore, as the amount of training data increases, the computational cost also increases accordingly. Another limitation is the lack of theoretical analysis of the convergence rate of the SSO algorithm, which we leave for future research. Finally, in our numerical experiments in Section 5, the proposed KIO model, even with the SSO algorithm, requires a large amount of memory in the order of 256 GB. However, we believe that the implementation can be improved to reduce the memory requirements.

References

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

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

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

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

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

[8] 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.

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

[10] 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.

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

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

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

[14] 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.

[15] 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.

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

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

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

[19] 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.

[20] 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.

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

[22] 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.

[23] Kevin P. Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.

[24] Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, June 2016.

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

[26] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Microsoft, 1998.

[27] 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.

[28] 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.

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

[30] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[31] Pedro Zattoni Scroccaro, Bilge Atasoy, and Peyman Mohajerin Esfahani. Learning in inverse optimization: Incenter cost, augmented suboptimality loss, and algorithms. *arXiv preprint arXiv:2305.07730*, 2023.

[32] Soroosh Shafieezadeh-Abadeh, Daniel Kuhn, and Peyman Mohajerin Esfahani. Regularization via mass transportation. *Journal of Machine Learning Research*, 20(103):1–68, 2019.

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

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

[35] 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.

[36] 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.

[37] Pedro Zattoni Scroccaro, Piet van Beek, Peyman Mohajerin Esfahani, and Bilge Atasoy. Inverse optimization for routing problems. *arXiv preprint arXiv:2307.07357*, 2023.

A Technical Proofs

A.1 Proof of Theorem 1

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 $\begin{bmatrix} \theta_{uu} & \hat{M}_i^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{w}_i) \\ * & \gamma_i \end{bmatrix} \succeq 0$ respectively, where $P, \Lambda_i \in \mathbb{R}^{n\times n}$, $\Gamma_i \in \mathbb{R}^n$, $\tilde{\lambda}_i \in \mathbb{R}^d$ and $\gamma_i \in \mathbb{R}$. Then define the Lagrangian function

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

and the Lagrange dual problem

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

When the Lagrangian function (14) is at the point of the infimum with respect to $\theta_{uu}, \theta_{su}, \lambda_i, \gamma_i$, we have

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

$$\frac{\partial L}{\partial \theta_{su}} = 2k\theta_{su} + 2\sum_{i=1}^N \phi(\hat{w}_i) \left(\frac{\hat{u}_i^\top}{N} - 2\Gamma_i^\top \right) = 0 \Rightarrow \theta_{su} = \frac{\sum_{i=1}^N \phi(\hat{w}_i) \left(2\Gamma_i^\top - \frac{\hat{u}_i^\top}{N} \right)}{k}$$

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

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

Finally, by substituting the expressions for $\theta_{uu}, \theta_{su}, \tilde{\lambda}_i$ and α_i into the Lagrange dual problem (15) and simplifying it ends the proof.

A.2 Proof of Proposition 2

Note that the problem (10) is the dual problem of the problem (4) with $\theta_{uu} = I_n$. Here, we enumerate the five KKT conditions that will be employed

$$\theta_{su} = \frac{\sum_{i=1}^N \phi(\hat{w}_i) \left(2\Gamma_i^\top - \frac{\hat{u}_i^\top}{N} \right)}{k} \qquad \text{(stationarity)}, \tag{16a}$$

$$\tilde{\lambda}_i = \frac{\hat{W}_i}{N} - 2\hat{M}_i \Gamma_i,\ \forall i \le N \qquad \text{(stationarity)}, \tag{16b}$$

$$\tilde{\lambda}_i{}^\top \lambda_i = 0,\ \forall i \le N \qquad \text{(complementary slackness)}, \tag{16c}$$

$$\mathrm{Tr}\left(\begin{bmatrix} \Lambda_i & \Gamma_i \\ * & \frac{1}{4N} \end{bmatrix} \begin{bmatrix} I_n & \hat{M}_i^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{w}_i) \\ * & \gamma_i \end{bmatrix} \right) = 0,\ \forall i \le N \quad \text{(complementary slackness)}, \tag{16d}$$

$$\lambda_i \in R_+^d,\ \forall i \le N \qquad \text{(primal feasibility)}. \tag{16e}$$

First, we choose coordinate i that satisfy condition (11). Then based on KKT condition (16b), we have

$$\tilde{\lambda}_i > 0. \tag{17}$$

Next, based on conditions (16c) and (16e), one can obtain

$$\lambda_i = 0. \tag{18}$$

Substituting the result (18) into condition (16d) yields

$$\mathrm{Tr}\left(\begin{bmatrix}\Lambda_i & \Gamma_i \\ * & \frac{1}{4N}\end{bmatrix}\begin{bmatrix}I_n & 2\theta_{su}^\top \phi(\hat{w}_i) \\ * & \gamma_i\end{bmatrix}\right) = 0. \tag{19}$$

γ_i is the decision variable of problem (4) with $\theta_{uu} = I_n$. Next, let's solve for its expression. By utilizing the Schur complement, we can prove the following two constraints are equivalent

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

Therefore, problem (4) with $\theta_{uu} = I_n$ can be equivalently expressed as

$$\begin{aligned}\min_{\theta_{su},\gamma_i,\lambda_i} \quad & k\|\theta_{su}\|_F^2 + \tfrac{1}{N}\textstyle\sum_{i=1}^N \left(2\hat{u}_i^T \theta_{su}^T \phi(\hat{w}_i) + \tfrac{1}{4}\gamma_i + \hat{W}_i^\top \lambda_i\right) \\ \text{s.t.} \quad & \lambda_i \in \mathbb{R}_+^d,\ \gamma_i \in \mathbb{R}, \quad \forall i \leq N \\ & \gamma_i \geq \left\|\hat{M}_i^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{w}_i)\right\|_2^2, \quad \forall i \leq N,\end{aligned} \tag{20}$$

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: $\gamma_i = \left\|\hat{M}_i^\top \lambda_i + 2\theta_{su}^\top \phi(\hat{w}_i)\right\|_2^2$. Note that $\lambda_i = 0$ (18), then the expression of γ_i is

$$\gamma_i = \left\|2\theta_{su}^\top \phi(\hat{w}_i)\right\|_2^2. \tag{21}$$

Substituting (21) into (19), one can obtain condition (12).

B Hyperparameters of KIO

Table 4 provides the hyperparameters used in the experiments in Section 5.

Table 4: KIO Environment Specific Parameters. (N is the size of the dataset and p is the number of coordinates that are updated in one iteration of SSO)

Environment	Dataset	k	$scalar$	p
Hopper-expert	0:5k	1e-6	$200N$	$0.5N$
Hopper-medium	0:5k	1e-6	$200N$	$0.5N$
Hopper-medium-expert	First 50k + Last 50k	1e-6	$200N$	$0.1N$
walker2d-expert	0:5k+10k:15k	1e-5	$50N$	$0.5N$
walker2d-medium	15k:20k	1e-5	$50N$	$0.5N$
walker2d-medium-expert	First 50k + Last 50k	1e-5	$50N$	$0.1N$
Halfcheetah-expert	325k:330k+345k:350k	5e-6	$50N$	$0.5N$
Halfcheetah-medium	10k:15k	5e-6	$50N$	$0.5N$
Halfcheetah-medium-expert	First 50k + Last 50k	1e-6	$50N$	$0.1N$