Multi-convex Programming for Discrete Latent Factor Models Prototyping

Hao Zhu, Shengchao Yan, Jasper Hoffmann, and Joschka Boedecker

Department of Computer Science and IMBIT//BrainLinks-BrainTools
University of Freiburg

June 26, 2025

Abstract

Discrete latent factor models (DLFMs) are widely used in various domains such as machine learning, economics, neuroscience, psychology, *etc.* Currently, fitting a DLFM to some dataset relies on a customized solver for individual models, which requires lots of effort to implement and is limited to the targeted specific instance of DLFMs. In this paper, we propose a generic framework based on `CVXPY`, which allows users to specify and solve the fitting problem of a wide range of DLFMs, including both regression and classification models, within a very short script. Our framework is flexible and inherently supports the integration of regularization terms and constraints on the DLFM parameters and latent factors, such that the users can easily prototype the DLFM structure according to their dataset and application scenario. We introduce our open-source Python implementation and illustrate the framework in several examples.

Contents

1 Introduction

Discrete latent factor models (DLFMs) represent a class of models in which some variables are always hidden. In this paper, we consider DLFMs expressed as,

$$\begin{array}{rcl} z & \sim & \mathbf{prob}(z) \\ y & \sim & \mathbf{prob}(y \mid x, z, \theta), \end{array} \tag{1.1}$$

where the *latent factor* $z \in \{1, \ldots, K\}$ is a discrete random variable, x and y are the *feature* and *observation*, respectively, and θ denotes the *model parameters*. For example, a *mixture of linear regressions* in standard form (1.1) is

$$\begin{array}{rcl} z & \sim & \mathrm{Cat}(p) \\ y & \sim & \mathcal{N}(x^T\theta_z, \sigma^2) \end{array}$$

with latent factor $z \in \{1, \ldots, K\}$, feature $x \in \mathbf{R}^n$, observation $y \in \mathbf{R}$, and parameters $\theta_z \in \mathbf{R}^n$ for all $z = 1, \ldots, K$, where $\mathrm{Cat}(p)$ denotes the categorical distribution with category probability vector $p \in \mathbf{R}^K_+$ satisfying $\mathbf{1}^T p = 1$, and $\mathcal{N}(x^T\theta_z, \sigma^2)$ is the Gaussian distribution with mean $x^T\theta_z$ and variance σ^2. DLFMs appear in domains such as machine learning, economics, signal processing, and cognitive science, with a vast range of applications [Mur23, chapter 28]. Specifically, in neuroscience and psychology, DLFMs have provided interpretable characterizations of both neural population activities [JMP21] and subject behavior [ARS+22, ZCK+24]. Note that the references listed here are by no means thorough; interested readers may refer to Jha *et al.* [JAP24] for a detailed review. In §6, we also provide some specific examples applied in these domains.

DLFM fitting problems. We consider the problem of fitting a DLFM model given the set of m observations $\{y_1, \ldots, y_m\}$ and the corresponding features $\{x_1, \ldots, x_m\}$. Informally, the DLFM fitting problem consists in finding the parameters θ_z for all $z = 1, \ldots, K$, and the latent factors z_i for all $i = 1, \ldots, m$, such that some metric indicating the model fitting error (*e.g.*, the negative log-likelihood of the data) is minimized. (We provide a formal definition of the DLFM fitting problem in §3.1.) The DLFM fitting problem is in general very hard to solve (actually, NP-hard, as we will show in §3.1). Since the latent factors $\{z_1, \ldots, z_m\}$ dominating the generation of the observations $\{y_1, \ldots, y_m\}$ under features $\{x_1, \ldots, x_m\}$ were not observed, in most cases, fitting a DLFM requires solving a nonconvex combinatorial optimization problem where the number of variables increases linearly with the number of samples in the dataset.

Current methods for fitting DLFMs. Various approaches have been introduced to fit different classes of DLFMs. The most commonly used method for fitting DLFMs is the *expectation-maximization* (EM) algorithm [DLR77]. EM algorithm is an iterative method with each iteration alternating between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E-step. The estimated parameters are then used to determine the distribution of the latent factors in the next E-step. The major limitation with EM is that only convergence to a local minimum is guaranteed. Thus, it is common to apply repeated EM iterations with random initializations and select the best-performing estimation, which turns out to work quite well in practice [ARS+22, ZCK+24]. Another class of methods for fitting

DLFMs is the *Monte Carlo* methods [Mur23, chapters 11–13], which instead of directly solving the fitting problem, but try to estimate the posterior distribution of the unknown parameters given the problem data and some predefined prior, by generating a large number of instances from the desired distribution. Monte Carlo methods avoid the local optimality limitation of EM methods and are hence more accurate than EM, but normally require a very long time to generate enough instances for an accurate estimation of the targeted posterior distribution. A compromise between EM and Monte Carlo is the *variational inference* methods [Mur23, chapter 10], which also try to (approximately) estimate a posterior distribution of the unknown parameters as in Monte Carlo methods, but via solving an alternative optimization problem. A more detailed discussion about Monte Carlo and variational inference methods for fitting DLFMs can be found in Jha *et al.* [JAP24].

Limitations. The aforementioned methods only provide an outline for designing algorithms, specifically, each for a different DLFM. To fully implement an algorithm instance that can be directly applied to the given data to fit a DLFM, one would need expert knowledge in lots of domains, such as probability, statistics, linear algebra, optimization, *etc.*, as well as vast minor but essential coding tricks. Although stable implementation of classic DLFMs (*e.g.*, hidden Markov models) is provided by open-source packages such as `scikit-learn` [PVG+11], the more recently developed DLFMs are published only (and hopefully) accompanied with source code closely dependent on the respective practical problems. In the former case, it would be challenging to integrate even minor adaptations to the provided DLFMs, such as constraints on model parameters. In the latter case, understanding the open-source implementation for the DLFM would already require a lot of effort, and at least medium level knowledge in the related domains. For the users, this blocks them from applying the powerful class of DLFMs, especially the more recently developed ones, to practical problems, since before launching the machinery designed for very fast and/or accurate DLFM fitting, the users might first just want to try different models and select the most suitable one for their specific application scenario, and sometimes incorporate minor adaptations. For the developers, upon proposing a new DLFM or some modifications to the existing DLFMs, lots of effort needs to be devoted to modifying or redesigning the algorithm for solving a different fitting problem, before even knowing whether the new idea will work or not. The current status strongly limits both further development and broader application of DLFMs.

Focus of the paper and contributions. The focus of this paper is *not* on methods for fitting specific DLFMs, but on a *framework* for DLFMs prototyping, which aims at avoiding the limitations of fitting different DLFMs to some given dataset via respective custom solvers. Our framework allows users to specify and solve the fitting problem of a wide range of DLFMs easily in a high level, human readable language, as well as introduce custom constraints and regularization terms to the model parameters and latent factors, based on their respective application scenarios. Our framework supports a wide range of regression and classification DLFMs, whose loss functions and parameter constraints are representable via *disciplined convex programming* (DCP), even though the fitting problem itself is not convex. Our work can be considered as an extension of the widely used *domain specific language* (DSL), `CVXPY` [DB16], for specifying, canonicalizing, and solving convex optimization problems, to deal with nonconvex DLFM fitting problems. Our implementation is fully open-source and can be found at

`https://github.com/nrgrp/dlfm`.

Paper structure. We introduce some background knowledge about convex optimization and multi-convex programming in §2, which is referred to in the sequel. The standard form of the DLFM fitting problem is introduced in §3. In this section, we also discuss the properties of the DLFM fitting problem, as well as some examples. We propose a generic method for (approximately) solving the DLFM fitting problem in §4, by introducing convex relaxations and problem transformation. In §5, we provide the implementation of our framework for DLFM prototyping. Finally, in §6, we list a number of numerical examples. The objective of showing these examples is *not* on competitive results in terms of solving time, quality, *etc.*, but rather to show the simplicity of specifying and fitting different DLFMs via our framework, along with results that are at least comparable to those obtained via specialized solvers for the individual problems.

2 Preliminaries

2.1 Disciplined convex programming

Disciplined convex programming is a framework for modeling convex optimization problems introduced by Grant *et al.* [Gra04, GBY06]. DCP imposes a ruleset, by following which the specified mathematical optimization problem can be easily verified as convex, and then canonicalized to a cone program which is eventually processed by some conic solver. The conforming problems from DCP are called *disciplined convex programs*.

The DCP ruleset restricts the set of functions that can appear in a problem and the way functions can be composed. Functions that appear in a disciplined convex program are restricted to those that are some composition of a set of atomic functions with known curvature and graph implementation, or can be represented as partial optimization over a cone program [NN92, GB08]. Suppose we are given some function $f = h(g_1(x), \ldots, g_p(x))$ where $h\colon \mathbf{R}^p \to \mathbf{R}$ is a convex function and $g_1, \ldots, g_p\colon \mathbf{R}^n \to \mathbf{R}$, and let $\tilde{h}\colon \mathbf{R}^p \to \mathbf{R}\cup\{\infty\}$ be the extended-value extension of h [BV04, §3.1]. The function f is convex if and only if one of the following conditions is satisfied for all $i = 1, \ldots, p$:

- g_i is convex and $\tilde{h}$ is nondecreasing in the ith argument;
- g_i is concave and $\tilde{h}$ is nonincreasing in the ith argument;
- g_i is affine.

(The composition rule for concave functions is analogous; see [BV04, §3.2].)

A mathematical optimization program has the general form

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \sim h_i(x), \quad i = 1, \ldots, m, \end{array} \tag{2.1}$$

where $x \in \mathbf{R}^n$ is the variable and the relational symbol $\sim$ denotes one of the relational operators $=$, $\leq$, or $\geq$. Problem (2.1) is a (DCP-supported) convex optimization problem if the functions f, g_i, and h_i are expressions with curvature verified by the DCP ruleset and the following curvature restrictions on these expressions are satisfied:

- f is convex (concave if (2.1) is a maximization problem);
- when the relational operator is $=$, g_i and h_i are both affine;

- when the relational operator is $\leq$, g_i is convex, and h_i is concave;
- when the relational operator is $\geq$, g_i is concave, and h_i is convex.

Note that an affine expression (function) is both convex and concave, and hence matches either curvature requirement.

Modeling systems for convex optimization problems based on the DCP ruleset have been well developed over the years. There are multiple DSLs designed for different programming languages, such as `YALMIP` [Lof04] and `CVX` [GB14] for MATLAB, `CVXPY` [DB16, AVDB18] for Python, and `Convex.jl` [UMZ+14] for Julia. Users are able to specify and solve a DSL verified convex optimization problem in a high level, human readable language, close to the corresponding mathematical expressions.

Our framework introduced in this paper is based on the DSL `CVXPY`, and uses the aforementioned DCP ruleset to verify whether an instance of the DLFM fitting problem is supported.

2.2 Multi-convex problems

We follow the notation from Shen *et al.* [SDU+17] for *multi-convex* optimization problems.

Fix variables in a function. Consider a function $f\colon \mathbf{R}^n \to \mathbf{R}$, and a partition of its variable $x \in \mathbf{R}^n$ into blocks:

$$x = (x_1, \ldots, x_N), \quad x_i \in \mathbf{R}^{n_i}, \quad \sum_{i=1}^{N} n_i = n. \tag{2.2}$$

Let $\mathcal{F} \subseteq \{1, \ldots, N\}$ denote an index set, such that $x_{\mathcal{F}} \in \{x_1, \ldots, x_N\}$, and let $\mathcal{F}^c = \{1, \ldots, N\} \setminus \mathcal{F}$ be the complement of $\mathcal{F}$. By saying *the function f with $\mathcal{F}$ fixed at point $\tilde{x} \in \mathbf{R}^n$*[1], we refer to the function $\tilde{f}$ with variables x_i for $i \in \mathcal{F}^c$ and for $i \in \mathcal{F}$, $x_i = \tilde{x}_i$. For example, the function $f(x_1, x_2) = x_1 x_2$ with $\mathcal{F} = \{2\}$ fixed at point $\tilde{x} = (3, 5)$ is the function $\tilde{f}(x_1) = 5x_1$. We sometimes omit the expression 'at point $\tilde{x} \in \mathbf{R}^n$' to refer to the general case where the function f is fixed at some point.

Multi-convex problems. Given an optimization problem with general structure (2.1) of variable $x \in \mathbf{R}^n$ partitioned into blocks as in (2.2). Let $\mathcal{F} \subseteq \{1, \ldots, N\}$ be some index set. We refer to *the problem (2.1) with $\mathcal{F}$ fixed* as the problem

$$\begin{array}{ll} \text{minimize} & \tilde{f}(x_{\mathcal{F}^c}) \\ \text{subject to} & \tilde{g}_i(x_{\mathcal{F}^c}) \sim \tilde{h}_i(x_{\mathcal{F}^c}), \quad i = 1, \ldots, m \end{array} \tag{2.3}$$

with variable $x_{\mathcal{F}^c}$, where the functions $\tilde{f}$, $\tilde{g}_i$, and $\tilde{h}_i$, $i = 1, \ldots, m$ are the functions f, g_i, and h_i with $\mathcal{F}$ fixed. We say *the problem (2.1) is convex with $\mathcal{F}$ fixed*, if the fixed problem (2.3) is convex, *i.e.*, satisfies the DCP ruleset in §2.1. We say the problem (2.1) is *multi-convex*, if there are sets $\mathcal{F}_1, \ldots, \mathcal{F}_p$, such that for all $i = 1, \ldots, p$, the problem (2.1) is convex with $\mathcal{F}_i$ fixed, and $\bigcap_{i=1}^{p} \mathcal{F}_i = \emptyset$. This definition includes convex problems as special cases with $p = 0$, *i.e.*, $\mathcal{F} = \emptyset$. A *biconvex* problem is multi-convex with $p = 2$. For example, the problem given by

$$\begin{array}{ll} \text{minimize} & |x_1 x_2| \\ \text{subject to} & x_1 + x_2 \geq 1 \end{array}$$

[1] Formally, this should be expressed as: *the function f with the variables x_i for $i \in \mathcal{F}$ fixed at point $\tilde{x} \in \mathbf{R}^n$*, but we will follow the convention from Shen *et al.* [SDU+17] to use the less formal expression in the text through the paper for simplicity.

with variable $x \in \mathbf{R}^2$ is *not* convex, but is biconvex with index sets $\mathcal{F}_1 = \{1\}$ and $\mathcal{F}_2 = \{2\}$.

3 The DLFM fitting problem

Fitting a DLFM to some given dataset is the primary objective and the first step for DLFM users. In this section, we provide a formal definition to the fitting problem of a wide range of DLFMs, followed by some analysis about the basic properties of such a problem, interpretations, and some specific examples that are widely used.

For simplicity of notation (and with slight abuse of notation), we will denote the latent factor as $z \in \{e_1, \ldots, e_K\} \subseteq \mathbf{R}^K$ in the sequel, where e_i are the ith standard basis vector in $\mathbf{R}^K$. Such a vector form notation of the latent factor can be readily transformed from the integer form notation via the one-to-one mapping $i \mapsto e_i$ with domain $\{1, \ldots, K\}$. We hope this will not cause confusion. We use the notation $\mathbf{card}\, x$ to denote the cardinality of some vector $x \in \mathbf{R}^n$, *i.e.*, the number of nonzero components of x.

3.1 Standard form DLFM fitting problems

The model fitting problem of DLFMs can be written as the following mixed integer program:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^m z_i^T r_i = \sum_{i=1}^m z_i^T (f(x_i, y_i; \theta_1), \ldots, f(x_i, y_i; \theta_K)) \\ \text{subject to} & z_i \in \{0,1\}^K, \quad \mathbf{card}\, z_i = 1, \quad i = 1, \ldots, m \\ & \theta_i \in \mathcal{C}, \quad i = 1, \ldots, K, \end{array} \tag{3.1}$$

where the problem variables are the latent factors $z_1, \ldots, z_m$ and the model parameters $\theta_1, \ldots, \theta_K$, the problem data are given by feature-observation pairs $\{x_i, y_i\}_{i=1}^m$. We assume the feasible set $\mathcal{C}$ for model parameters is closed and convex, *i.e.*, can be specified via the DCP ruleset for constraints. The function f is some metric representing the fitting error, and is assumed to be convex under the DCP ruleset and resolves to a scalar. The vectors $r_i = (f(x_i, y_i; \theta_1), \ldots, f(x_i, y_i; \theta_K)) \in \mathbf{R}^K$ are then concatenations of the fitting error under each parameter $\theta_1, \ldots, \theta_K$, evaluated on the dataset $\{x_i, y_i\}_{i=1}^m$.

Extensions. Note that the problem (3.1) can be readily extended to the cases where the models for generating observations y given feature x are different across latent factors, *i.e.*, consider

$$r_i = (f_1(x_i, y_i; \theta_1), \ldots, f_K(x_i, y_i; \theta_K)) \quad \text{and} \quad \theta_1 \in \mathcal{C}_1, \quad \ldots, \quad \theta_K \in \mathcal{C}_K,$$

for all $i = 1, \ldots, m$. Theoretically, such an extension does not significantly influence the properties of the problem (3.1), as long as our assumptions about the convexity of $f_1, \ldots, f_K$ and $\mathcal{C}_1, \ldots, \mathcal{C}_K$ still hold. (This can be verified with basic convex analysis.) Besides, the assumption that the observations are generated via different models under individual latent factors is less considered in practice (since, *e.g.*, there might be some conceptual issues when mixing different metrics together). Hence, we will focus on the problem formulation (3.1) in the subsequent discussion. In practice, such an extension is indeed technically supported by our implementation in §5.

Interpretation. The DLFM fitting problem (3.1) can be interpreted as follows. For all samples $\{x_i, y_i\}$ in the dataset, given some model parameters $\theta_1, \ldots, \theta_K$, the model fitting error $f(x_i, y_i; \theta_k)$ is

evaluated for all θ_k, $k = 1, \ldots, K$, *i.e.*, evaluated under all possible latent factors. This corresponds to the vector r_i, $i = 1, \ldots, m$. The constraints on the latent factors, given by $z_i \in \{0,1\}^K$ and $\mathbf{card}\, z_i = 1$ for $i = 1, \ldots, m$, require that the vectors z_i must be one-hot vectors, *i.e.*, in the set of standard basis vectors $\{e_1, \ldots, e_K\}$. Hence, by forming the inner product $z_i^T r_i$ for all samples in the dataset, we assign a unique label corresponding to a latent factor to each sample, where the model fitting error is cumulated into the objective. In other word, by solving the problem (3.1), we would like to separate the dataset $\{x_i, y_i\}_{i=1}^m$ into K clusters, such that the total model fitting error of all clusters is minimized.

NP-hardness. The DLFM fitting problem (3.1) is NP-hard (even if we have restricted the function f to be convex). To show this, consider the ℓ_2-norm squared metric on $\mathbf{R}^n$, given by $f(x, y; \theta) = \|\theta - x - y\|_2^2$. Let the dataset be $\{x_i, y_i = 0\}_{i=1}^m$, and $\mathcal{C} = \mathbf{R}^n$. This leads to an instance of (3.1) given by

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^m z_i^T(\|\theta_1 - x_i\|_2^2, \ldots, \|\theta_K - x_i\|_2^2) \\ \text{subject to} & z_i \in \{0,1\}^K, \quad \mathbf{card}\, z_i = 1, \quad i = 1, \ldots, m \end{array} \tag{3.2}$$

with variables $z_1, \ldots, z_m \in \{e_1, \ldots, e_K\}$ and $\theta_1, \ldots, \theta_K \in \mathbf{R}^n$. The problem (3.2) is equivalent to the k-means clustering problem in $\mathbf{R}^n$ with K clusters. Hence, the DLFM fitting problem (3.1) is at least as hard as the k-means clustering problem (3.2), which is known to be NP-hard [ADHP09].

3.2 Examples

Recall that we assume the function f in (3.1) is scalar valued and convex under the DCP ruleset, and the feasible set $\mathcal{C}$ is convex. Although these assumptions do not support all types of DLFM fitting problems, they still capture a wide range of metric functions for both regression and classification models, including the following examples.

Regression models. Consider the function f in (3.1) with structure

$$f(x, y; \theta) = g(x^T\theta - y), \tag{3.3}$$

where $x, \theta \in \mathbf{R}^n$, $y \in \mathbf{R}$, and $g\colon \mathbf{R} \to \mathbf{R}$ is some loss function, *e.g.*,

- square loss: $g(u) = u^2$;
- ℓ_p-loss: $g(u) = \|u\|_p$ for $p \in [1, \infty]$;
- Huber loss: $f(u) = u^2$ for $|u| \leq \delta$, and $f(u) = 2\delta|u| - \delta^2$ for $|u| > \delta$, where $\delta > 0$ is a parameter.

Since $x^T\theta - y$ is affine in θ, the function f given by (3.3) is convex in θ if the selected loss function g is convex (which is satisfied by the listed three examples). The corresponding DLFMs are sometimes named *mixture of linear regressions*. The same idea can be extended when the observations are in a higher dimensional space. For example, one may consider the least squares loss $g(u) = \|u\|_2^2$ or the ℓ_1-loss $g(u) = \|u\|_1$ for vector valued observations, and $g(U) = \|U\|_F^2 = \mathbf{tr}(U^TU)$ (*i.e.*, the square of the Frobenius norm) for matrix valued observations, given that the variable taken by g is affine in θ.

Classification models. A simple example for classification DLFMs would be k-means clustering, with its corresponding model fitting problem given by (3.2). As a more complex example, consider the loss function f given by

$$f(X, y; \theta) = -\log\left(\frac{y^T \exp u}{\sum_{i=1}^p \exp u_i}\right), \quad u = X\theta, \tag{3.4}$$

where $X \in \mathbf{R}^{p\times n}$, $\theta \in \mathbf{R}^n$, $y \in \{e_1, \ldots, e_p\} \subseteq \mathbf{R}^p$ is a one-hot label vector. The intermediate feature vector $u = X\theta$ is then in $\mathbf{R}^p$ (with its ith entry denoted as u_i). The loss function given by (3.4) is the objective function (*i.e.*, the negative log-likelihood) for multi-class logistic regression problems, which is convex under the DCP ruleset. To show this, notice that f can be written as

$$f(X, y; \theta) = -\log\left(\frac{y^T \exp u}{\sum_{i=1}^p \exp u_i}\right) = -\left(y^T u - \log\sum_{i=1}^p \exp u_i\right). \tag{3.5}$$

(To obtain the second equality, we use the fact that $\log(y^T \exp u) = y^T u$ if y is a standard basis vector.) Since the log-sum-exp expression, given by $\log\sum_{i=1}^p \exp u_i$, is known to be convex in u [BV04, §3.1], and u is affine in θ, it follows immediately that the function f given by (3.4) is convex in θ. The DLFMs corresponding to the problem (3.1) with loss function (3.4) are sometimes referred to as the *hierarchical logistic regression* model. Note that such formulation includes the binary logistic regression as a special case, and can be readily adapted to deal with hinge loss or exponential loss for binary classification models.

Constraints on parameters. In the above examples, we simply assume there is no constraint on the model parameters θ, *i.e.*, $\mathcal{C} = \mathbf{R}^n$. It is also common in practice to restrict the parameters, for instance, $\theta \in \mathbf{R}^n$, to some convex subset $\mathcal{C}$ of $\mathbf{R}^n$, given some prior information. The DCP ruleset supports a vast range of constraints that can jointly specify the convex set $\mathcal{C}$, *e.g.*, nonnegativity constraint $\theta \succeq 0$ (where $\succeq$ denotes componentwise inequality), unit norm constraint $\|\theta\|_2 \leq 1$, and summation constraint $\mathbf{1}^T\theta = 1$, just list a few.

4 Heuristic solution method

In this section, we introduce relaxations to the DLFM fitting problem (3.1) such that it can be transformed into a multi-convex program and solved (at least approximately).

4.1 Multi-convex program formulation

Relaxations. We start from relaxing the discrete cardinality constraints about the latent factors $\{z_1, \ldots, z_m\}$ of the problem (3.1). Recall that the constraints, given by

$$z_i \in \{0, 1\}^K, \quad \mathbf{card}\, z_i = 1, \quad i = 1, \ldots, m, \tag{4.1}$$

restrict the latent factors to be standard basis vectors in $\mathbf{R}^K$, *i.e.*, $z_i \in \{e_1, \ldots, e_K\}$. It is in general difficult to handle these constraints when the dataset $\{x_i, y_i\}_{i=1}^m$ is large, and the machinery of combinatorial optimization is often required. To avoid solving a combinatorial optimization problem, we relax the constraints (4.1) to

$$0 \preceq z_i \preceq \mathbf{1}, \quad \mathbf{1}^T z_i = 1, \quad i = 1, \ldots, m. \tag{4.2}$$

The relaxed constraints about the latent factor $z_1, \ldots, z_m$, given by (4.2), are now 'smooth' in z_i, and can hence be handled by most solving methods.

Interpretation. Compared to (4.1), the constraints given by (4.2) has the following interpretations: By solving the DLFM fitting problem with constraints (4.1), we aim to find a 'hard-encoded' (deterministic) label z_i for each sample $\{x_i, y_i\}$ in the dataset, whereas in (4.3), each entry in z_i can be interpreted as the probability of the sample $\{x_i, y_i\}$ belonging to the corresponding factors. Hence, the relaxed constraints (4.2) allow 'soft-encoded' (stochastic) latent factors for individual samples. In practice, if required, the stochastic latent factors from (4.2) can be readily transformed into the deterministic latent factors in (4.1) via the argmax operator.

MCP formulation. Incorporating the relaxed constraints (4.3), the relaxed DLFM fitting problem is then written as:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^m z_i^T r_i = \sum_{i=1}^m z_i^T (f(x_i, y_i; \theta_1), \ldots, f(x_i, y_i; \theta_K)) \\ \text{subject to} & 0 \preceq z_i \preceq \mathbf{1}, \quad \mathbf{1}^T z_i = 1, \quad i = 1, \ldots, m \\ & \theta_i \in \mathcal{C}, \quad i = 1, \ldots, K, \end{array} \tag{4.3}$$

with variables $z_1, \ldots, z_m \in \mathbf{R}^K$ and $\theta_1, \ldots, \theta_K$ depending on the DLFM structure. We will use the notation $\Theta = (z_1, \ldots, z_m, \theta_1, \ldots, \theta_K)$ to denote the vector of all variables of (4.3) in the sequel. The relaxed problem (4.3) is multi-convex (or specifically, biconvex) with index sets $\mathcal{F}_z$ and $\mathcal{F}_\theta$, where $\Theta_{\mathcal{F}_z} = (z_1, \ldots, z_m)$ and $\Theta_{\mathcal{F}_\theta} = (\theta_1, \ldots, \theta_K)$. To show this, we need to verify that:

(a) $\mathcal{F}_z \cap \mathcal{F}_\theta = \emptyset$, and

(b) the problem (4.3) is convex with $\mathcal{F}_z$ fixed, and

(c) the problem (4.3) is convex with $\mathcal{F}_\theta$ fixed.

Proof of multi-convexity. Apparently, the condition (a) is satisfied since $\Theta_{\mathcal{F}_z} \cap \Theta_{\mathcal{F}_\theta} = \emptyset$. To verify the condition (b), consider the problem (4.3) with $\mathcal{F}_z$ fixed, given by

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^m \tilde{z}_i^T (f(x_i, y_i; \theta_1), \ldots, f(x_i, y_i; \theta_K)) \\ \text{subject to} & \theta_i \in \mathcal{C}, \quad i = 1, \ldots, K, \end{array} \tag{4.4}$$

with variables $\theta_1, \ldots, \theta_K$ and data $\{x_i, y_i, \tilde{z}_i\}_{i=1}^m$ (where $\tilde{z}_i$ are some fixed latent factors). Since the loss function f is convex in the variables $\theta_1, \ldots, \theta_K$ and $0 \preceq \tilde{z}_1, \ldots, \tilde{z}_m \preceq \mathbf{1}$, the objective of (4.4) is a nonnegative weighted sum of multiple convex functions (and hence convex). According to our assumption in §3.1, the feasible set $\mathcal{C}$ satisfies the DCP ruleset. Put together, we conclude that the problem (4.4) is a convex optimization problem, and thus the condition (b) is satisfied. Now we show that the condition (c) also holds. Similarly, we have the problem (4.3) with $\mathcal{F}_\theta$ fixed, given by

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^m z_i^T \tilde{r}_i \\ \text{subject to} & 0 \preceq z_i \preceq \mathbf{1}, \quad \mathbf{1}^T z_i = 1, \quad i = 1, \ldots, m, \end{array} \tag{4.5}$$

with variables $z_1, \ldots, z_m \in \mathbf{R}^K$. The data of the problem (4.5) is $\tilde{r}_1, \ldots, \tilde{r}_m \in \mathbf{R}^K$ obtained from evaluating $f(x_i, y_i; \tilde{\theta}_k)$ for $i = 1, \ldots, m$, $k = 1, \ldots, K$, where $\tilde{\theta}_k$ are some fixed model parameters. The problem (4.5) is then readily verified to be a linear program, which is, of course, convex.

4.2 Heuristic solution via block coordinate descent

A generic method for solving multi-convex optimization problems is the class of *block coordinate descent* (BCD) methods. The general idea of BCD-type methods (which dates back to Warga [War63] and Powell [Pow73]) is to alternate between multiple convex problems, where each problem is a subproblem of the original multi-convex program with a specific index set fixed. Specifically, for the relaxed DLFM fitting problem (4.3), in each BCD iteration, we iterate between solving the *parameter (P) problem* and the *factor (F) problem*, given by

$$\begin{array}{lll} \text{(P)} & & \\ \text{minimize} & \sum_{i=1}^m \tilde{z}_i^T r_i & \\ \text{subject to} & r_i = (f(x_i, y_i; \theta_k))_{k=1}^K, \quad \theta_k \in \mathcal{C} & \\ & i = 1, \ldots, m, \quad k = 1, \ldots, K, & \end{array} \qquad \begin{array}{lll} \text{(F)} & & \\ \text{minimize} & \sum_{i=1}^m z_i^T \tilde{r}_i & \\ \text{subject to} & 0 \preceq z_i \preceq \mathbf{1}, \quad \mathbf{1}^T z_i = 1 & \\ & i = 1, \ldots, m. & \end{array} \tag{4.6}$$

The P-problem has variables $\theta_1, \ldots, \theta_K$ and data $\{x_i, y_i\}_{i=1}^m$ from the dataset, $\tilde{z}_1, \ldots, \tilde{z}_m \in \mathbf{R}^K$ corresponding to the optimal point of the F-problem in the last iteration. Correspondingly, the F-problem has variables $z_1, \ldots, z_m \in \mathbf{R}^K$ and data $\tilde{r}_i = (f(x_i, y_i; \tilde{\theta}_1), \ldots, f(x_i, y_i; \tilde{\theta}_K))$, $i = 1, \ldots, m$, where $\tilde{\theta}_1, \ldots, \tilde{\theta}_K$ are the optimal point of the P-problem in the last iteration.

Regularizations. Note that although it is not mentioned explicitly in (4.6), regularization terms can be integrated into the objective function of each problem. As an example, if we would like to obtain some model parameters $\theta_k \in \mathbf{R}^n$ that are sparse (*i.e.*, have as many zero entries as possible) for all $k = 1, \ldots, K$, we can add an ℓ_1-regularization term $\lambda \sum_{k=1}^K \|\theta_k\|_1$ with $\lambda \geq 0$ being the penalty weight to the objective of the P-problem [Tib96]. As another example regarding the F-problem, if the dataset $\{x_i, y_i\}_{i=1}^m$ are from some time series, a commonly considered assumption for DLFM fitting is that the latent factors should change as less as possible, this leads to the regularization $\lambda \sum_{i=1}^{m-1} D_{\text{kl}}(z_i, z_{i+1})$ (with weight $\lambda \geq 0$), where for positive vectors $u, v \in \mathbf{R}_{++}^n$, $D_{\text{kl}}(u, v) = \sum_{i=1}^n (u_i \log(u_i/v_i) - u_i + v_i)$ is the *Kullback-Leibler (KL) divergence* (which is known to be convex jointly in (u, v) [BV04, §3.2]). The sparsity of the change points of latent factors comes from the fact that $\sum_{i=1}^{m-1} D_{\text{kl}}(z_i, z_{i+1}) = \|(D_{\text{kl}}(z_1, z_2), D_{\text{kl}}(z_2, z_3), \ldots, D_{\text{kl}}(z_{m-1}, z_m))\|_1$, since the KL-divergence is always nonnegative.

Termination. There are several options for determining when to quit the BCD iteration. The most straightforward and commonly considered approach is to set a maximum iteration number, as used by Shen *et al.* [SDU+17]. Besides, some criteria can be incorporated to automatically terminate the BCD iteration for the two subproblems (4.6). For instance, one may quit the algorithm when the gap between the optimal values of the P- and F-problem is less than some very small number $\epsilon \geq 0$ (given that there are no additional regularization terms on both problems), or quit if the update on the variables between iterations is smaller than some threshold.

Convergence. In general, very little can be said about the convergence of BCD-type methods for multi-convex problems. Interested readers may refer to, *e.g.*, Warga [War63], Powell [Pow73], and Nutini *et al.* [NLS22], for some convergence results under strong assumptions about convexity or differentiability (of the subproblems with some variables fixed). In case of the relaxed DLFM fitting problem (4.3), one obvious observation is that the objectives of both the P-problem and F-problem are nonincreasing in each BCD iteration, and hence converges. Despite a lack of strong convergence

theory in the general case, the BCD-type methods have been found to be robust and very useful in practice.

5 Implementation

In this section, we introduce the implementation of our framework for specifying and solving DLFM fitting problems, based on the *disciplined parameterized programming* (DPP) [AAB+19] implementation of the heuristic solution method discussed in §4. DPP incorporates symbolic representation to (a subset of) the problem data (*i.e.*, problem *parameters*), such that the value of these parameters can be modified without reconstructing the entire problem. (Note that the meaning of the word 'parameter' for an optimization problem is different from that for a DLFM, as we referred to previously.) For example, in the case of the problems (4.6), the parameters are considered to be $\tilde{z}_1, \ldots, \tilde{z}_m \in \mathbf{R}^K$ for the P-problem, and $\tilde{r}_1, \ldots, \tilde{r}_m \in \mathbf{R}^K$ for the F-problem. By specifying a DCP problem according to DPP, solving it repeatedly for different values of the parameters can be much faster than repeatedly solving a new problem.[2]

The implementation of our framework is given in the listing below. The corresponding code is fully open-source and is available at

https://github.com/nrgrp/dlfm.

```
import numpy as np
import cvxpy as cp

### problem data
xs = None  # ndarray: dataset features
ys = None  # ndarray: dataset observations
m = None  # int: number of samples in the dataset

### hyperparameters
eps = 1e-6  # float: termination criterion

### P-problem
K = None  # int: number of latent factors
thetas = []  # list of cp.Variable objects: model parameters
r = []  # list of cp.Expression objects: loss functions

ztil = cp.Parameter((m, K), nonneg=True)
Pobj = cp.sum(cp.multiply(ztil, cp.vstack(r).T))
Preg = 0  # cp.Expression: regularization on model parameters
Pconstr = []  # list of cp.Constraint objects: model parameter constraints
Pprob = cp.Problem(cp.Minimize(Pobj + Preg), Pconstr)
assert Pprob.is_dcp()

### F-problem
rtil = cp.Parameter((K, m))
```

[2] For large problems, the non-DPP implementation (which can be readily obtained from our DPP based implementation provided below) is sometimes faster than DPP, since for these problems the canonicalization step in the `CVXPY` backend may take a very long time.

```
z = cp.Variable((m, K))
Fobj = cp.sum(cp.multiply(z, rtil.T))
Freg = 0  # cp.Expression: regularization on latent factors
Fconstr = [z >= 0, z <= 1, cp.sum(z, axis=1) == 1]
Fprob = cp.Problem(cp.Minimize(Fobj + Freg), Fconstr)
assert Fprob.is_dcp()

### solve, terminate when the F- and P-objective converge
while True:
    if ztil.value is None:
        ztil.value = np.random.dirichlet(np.ones(K), size=m)
    else:
        ztil.value = np.abs(z.value)
    Pprob.solve()

    rtil.value = cp.vstack(r).value
    Fprob.solve()

    if np.abs(Pobj.value - Fobj.value) < eps:
        break
```

To fully specify the fitting problem for customized DLFMs, the users only need to instantiate those objects that are commented in the listing, to which we provide a detailed description in the subsequent paragraphs.

Dataset. The dataset for the DLFM fitting problem is specified by the features `xs`, the observations `ys`, and the number of samples in the dataset `m`. In the general case, the first two objects should be instantiated as a `numpy.ndarray` with the first dimension equal to the integer `m`.

P-problem. The P-problem requires the user to specify three mandatory objects: `K`, `thetas`, `r`, and two optional objects: `Preg`, `Pconstr`. The lists, `thetas` and `r`, represent the parameters and the loss function of the DLFM corresponding to each latent factor, respectively. The integer `K` is the number of latent factors. If regularization and/or constraints need to be applied to the model parameters, the users need to additionally instantiate the objects `Preg` and/or `Pconstr`. The lists `theta` and `r` should consist of `cvxpy.Variable` and `cvxpy.Expression` objects, respectively. Note that each expression in the list `r` has to resolve to a vector in $\mathbf{R}^K$, according to (4.6). In the general case, the integer `K` should equal to the number of elements in `thetas` and `r`. The regularization term `Preg` is a `cvxpy.Expression` object that resolves to a scalar. The constraints `Pconstr` is a list of `cvxpy.Constraint` objects that can be specified according to the standard `CVXPY` grammar.

F-problem. Not much input from the user is required to fully specify the F-problem, except when one would like to integrate regularization terms to the latent factor variables $z_i, \ldots, z_m$ (*i.e.*, the variable `z` in the code). In this case, the user needs to instantiate the object `Freg` as a `cvxpy.Expression` object that resolves to a scalar (similar to `Preg` for the P-problem).

Termination. We implement the solving iteration as alternating between solving the P- and F-problem, until the gap between the optimal value of the two problems is small enough, controlled by

the nonnegative floating point number `eps`. The default value of the number `eps` is set to be 10^{-6}, which turns out to work well for many DLFM fitting problems (see the numerical examples in §6). The users can freely change this threshold according to their application scenario.

6 Examples

This section provides some numerical examples that apply our framework for fitting different DLFMs. The code corresponding to individual examples is based on the implementation provided in §5 with related lines adapted to respective problems. Interested readers can access the full implementation at

`https://github.com/nrgrp/dlfm`

as a coding reference.

6.1 Constrained k-means clustering

Problem description. We start from a toy example to demonstrate the basic usage of our framework. Suppose we are given a dataset consisting of $m = 500$ points with feature dimension $n = 2$, given by $x_1, \ldots, x_m \in \mathbf{R}^2$, uniformly generated according to $x_i = \bar{x}_i + v_i$, where

$$\bar{x}_i \in \{x \in \mathbf{R}^2 \mid \|x\|_1 = 2\}, \quad v_i \sim \mathcal{N}(0,\ 0.05^2 I), \tag{6.1}$$

and $y_1 = \cdots = y_m = 0$. (The matrix I is the identity matrix.) The objective is to partition these points into $K = 4$ clusters such that the within-cluster variance is minimized. Besides, we require that the center of each cluster $\theta_1, \ldots, \theta_K \in \mathbf{R}^2$ is constrained in a polyhedron, given by

$$\theta_i \in \{x \in \mathbf{R}^2 \mid Ax \preceq b\}, \quad \text{where} \quad A = \begin{bmatrix} 0.8 & 0.6 \\ -0.7 & 0.9 \\ -1 & -0.5 \\ 1 & -1 \\ 0.3 & 0.9 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0.8 \\ 0.6 \\ 0.7 \\ 0.8 \end{bmatrix}. \tag{6.2}$$

This problem corresponds to the k-means clustering problem (3.2) with linear constraints (6.2). Formally, the optimization problem of this example is written as

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^m z_i^T(\|\theta_1 - x_i\|_2^2, \ldots, \|\theta_K - x_i\|_2^2) \\ \text{subject to} & z_i \in \{0,1\}^K, \quad \mathbf{card}\, z_i = 1, \quad i = 1, \ldots, m \\ & A\theta_i \preceq b, \quad i = 1, \ldots, K, \end{array} \tag{6.3}$$

where $\theta_1, \ldots, \theta_K \in \mathbf{R}^2$ and $z_1, \ldots, z_m \in \mathbf{R}^K$ are the variables, $x_1, \ldots, x_m \in \mathbf{R}^2$, $A \in \mathbf{R}^{5\times 2}$ and $b \in \mathbf{R}^5$ given by (6.1) and (6.2) are the data.

Problem specification. The problems we solve in each BCD iteration, corresponding to the constrained k-means clustering problem (6.3), are given by

$$\begin{array}{ll} \text{(P)} & \\ \text{minimize} & \sum_{i=1}^m \tilde{z}_i^T r_i \\ \text{subject to} & r_i = \left(\|\theta_k - x_i\|_2^2\right)_{k=1}^K, \quad i = 1, \ldots, m \\ & A\theta_k \preceq b, \quad k = 1, \ldots, K, \end{array} \qquad \begin{array}{ll} \text{(F)} & \\ \text{minimize} & \sum_{i=1}^m z_i^T \tilde{r}_i \\ \text{subject to} & 0 \preceq z_i \preceq \mathbf{1}, \quad \mathbf{1}^T z_i = 1 \\ & i = 1, \ldots, m. \end{array} \tag{6.4}$$

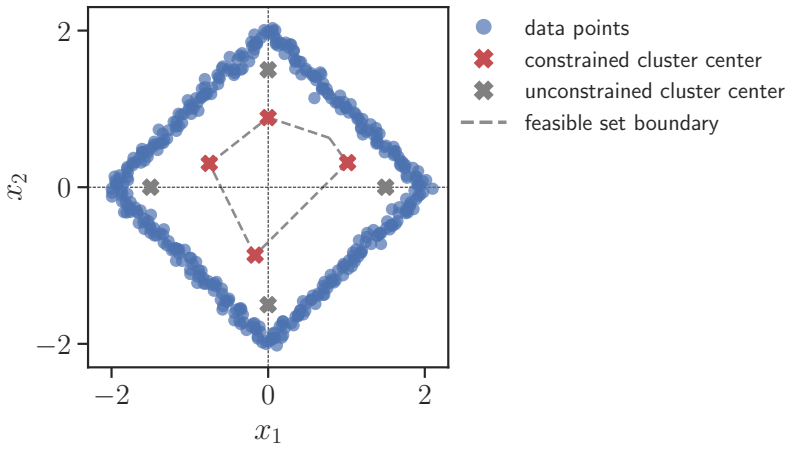

Figure 1 Results for the constrained k-means clustering example.

The P-problem has variables $\theta_1, \ldots, \theta_K \in \mathbf{R}^2$ and data $x_1, \ldots, x_m \in \mathbf{R}^2$, $\tilde{z}_1, \ldots, \tilde{z}_m \in \mathbf{R}^K$, $A \in \mathbf{R}^{5\times 2}$, $b \in \mathbf{R}^5$. The F-problem has variables $z_1, \ldots, z_m \in \mathbf{R}^K$ and data $\tilde{r}_1, \ldots, \tilde{r}_m \in \mathbf{R}^K$. To type the problems (6.4) into our framework, the major adaptations we need to make are given by the following code.

```
for k in range(K):
    thetas.append(cp.Variable((1, n)))
    r.append(cp.sum(cp.square(xs - thetas[-1]), axis=1))
Pconstr = [A @ theta.T <= b for theta in thetas]
```

Numerical result. The result for this example is shown in figure 1. The blue dots represent the dataset, and the gray dashed lines are the boundary of the feasible set given by (6.2). The returned centers for different clusters are shown as red crosses, which are all at the vertices of the polyhedron defined by the feasible set. As a reference, the cluster centers for the same dataset when removing the constraints (6.2) are plotted in gray crosses.

6.2 Mixture of linear regressions

Problem description. We now demonstrate an application of our framework in fitting a mixture of linear regressions. Such a DLFM model has a long application history in machine learning [BF94, GS99, LL18]. Suppose we are given a dataset $\{x_i, y_i\}_{i=1}^m$ generated according to

$$x_i \sim \mathcal{U}(a, b), \quad \theta_i \sim \text{Cat}(\{\theta_1, \ldots, \theta_K\},\ p), \quad y_i \sim \mathcal{N}(x_i^T\theta_i, \sigma^2),$$

i.e., for the ith sample, the feature vector $x_i \in \mathbf{R}^n$ is first generated from a uniform distribution between the lower bound $a \in \mathbf{R}^n$ and upper bound $b \in \mathbf{R}^n$ (assume $a \prec b$), then the linear combination coefficients $\theta_i \in \mathbf{R}^n$ are generated from a categorical distribution on the set $\{\theta_1, \ldots, \theta_K\}$ according to probabilities $p \in \mathbf{R}_+^K$ with $\mathbf{1}^T p = 1$, and finally the observation $y_i \in \mathbf{R}$ is generated from the Gaussian distribution with mean $x_i^T\theta_i$ and variance σ^2. To fit this mixture of linear regressions given

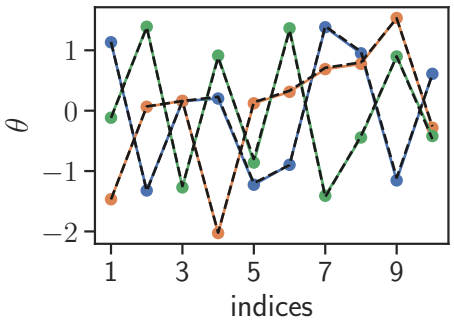

Figure 2 Results for the mixture of linear regressions example. The three colored solid lines are the recovered parameters and the black dashed lines are the corresponding ground truth.

the dataset $\{x_i, y_i\}_{i=1}^m$, we need to recover the set of parameters $\{\theta_1, \ldots, \theta_K\}$, as well as the latent factor labels indicating which $\theta_i \in \{\theta_1, \ldots, \theta_K\}$ was used to generate the observation y_i given feature x_i, for all $i = 1, \ldots, m$. This optimization problem in standard form (3.1) is given by

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^m z_i^T((x_i^T\theta_1 - y_i)^2, \ldots, (x_i^T\theta_K - y_i)^2) \\ \text{subject to} & z_i \in \{0,1\}^K, \quad \mathbf{card}\, z_i = 1, \quad i = 1, \ldots, m, \end{array} \tag{6.5}$$

where $\theta_1, \ldots, \theta_K \in \mathbf{R}^n$ and $z_1, \ldots, z_m \in \mathbf{R}^K$ are the variables, $x_1, \ldots, x_m \in \mathbf{R}^n$ and $y_1, \ldots, y_m \in \mathbf{R}$ are the data.

Problem specification. The problems we solve in each BCD iteration corresponding to (6.5) are given by

$$\begin{array}{lll} \text{(P)} & & \\ \text{minimize} & \sum_{i=1}^m \tilde{z}_i^T r_i & \\ \text{subject to} & r_i = \left((x_i^T\theta_k - y_i)^2\right)_{k=1}^K & \\ & i = 1, \ldots, m, & \end{array} \qquad \begin{array}{lll} \text{(F)} & & \\ \text{minimize} & \sum_{i=1}^m z_i^T \tilde{r}_i & \\ \text{subject to} & 0 \preceq z_i \preceq \mathbf{1}, & \mathbf{1}^T z_i = 1 \\ & i = 1, \ldots, m. & \end{array} \tag{6.6}$$

The P-problem has variables $\theta_1, \ldots, \theta_K \in \mathbf{R}^n$ and data $x_1, \ldots, x_m \in \mathbf{R}^n$, $y_1, \ldots, y_m \in \mathbf{R}$, $\tilde{z}_1, \ldots, \tilde{z}_m \in \mathbf{R}^K$; the F-problem has variables $z_1, \ldots, z_m \in \mathbf{R}^K$ and data $\tilde{r}_1, \ldots, \tilde{r}_m \in \mathbf{R}^K$. The major adaptations we need to make to specify the problems (6.6) using our framework are given by the following code.

```
for k in range(K):
    thetas.append(cp.Variable(n))
    r.append(cp.square(xs @ thetas[-1] - ys))
```

Numerical result. The dataset put into test in this example consists of $m = 500$ samples in $\mathbf{R}^{10}$ with $K = 3$, and was generated under the following parameters:

$$\begin{aligned} &a = (-10, \ \ldots, \ -10) \in \mathbf{R}^{10}, \quad b = (10, \ \ldots, \ 10) \in \mathbf{R}^{10} \\ &\theta_1 = (-1.47, \ 0.07, \ 0.16, \ -2.02, \ 0.14, \ 0.33, \ 0.71, \ 0.80, \ 1.53, \ -0.26) \end{aligned}$$

$$\theta_2 = (-0.12,\ 1.38,\ -1.25,\ 0.88,\ -0.80,\ 1.33,\ -1.43,\ -0.42,\ 0.90,\ -0.47)$$
$$\theta_3 = (1.14,\ -1.33,\ 0.16,\ 0.23,\ -1.20,\ -0.90,\ 1.40,\ 0.98,\ -1.11,\ 0.60)$$
$$p = (0.4,\ 0.3,\ 0.3), \quad \sigma^2 = 1.5^2.$$

The recovered set of parameters is shown in figure 2, where the solid lines are the fitted parameters and the black dashed lines are the ground truth. It can be seen that the recovered and true parameters align exactly. Besides, the accuracy of recovering the latent factor labels for each sample in the dataset is 0.94.

6.3 Hierarchical forgetting Q-learning

Problem description. We consider the *hierarchical forgetting Q-learning* model that is widely used in neuroscience and psychology for the modeling of subject's decision making behavior under multi-armed bandit tasks [ID09, BNLS22, ZCK+24, ZB25]. Consider an agent repeatedly facing a choice among p actions, with (potentially different) probabilities of obtaining a reward from each action. After the choice at time step $t-1$, the agent receives a *reward signal* $u(t) \in \mathbf{R}^p$ that depends on the selected action, *e.g.*,

$$u_i(t) = \begin{cases} 1 & \text{if action } i \text{ was selected } \textit{and} \text{ rewarded} \\ 0 & \text{otherwise,} \end{cases} \tag{6.7}$$

for $i = 1, \ldots, p$, where $u_i(t)$ denotes the ith element of the vector $u(t)$. To select the action for the next time step t, the agent formulates a *value function* $v(t) \in \mathbf{R}^p$ according to

$$v(t) = X(t)\theta(t), \quad X(t) = \begin{bmatrix} u(t) & u(t-1) & \cdots & u(t-n+1) \end{bmatrix} \in \mathbf{R}^{p\times n}, \tag{6.8}$$

where $\theta(t) \in \{\theta_1, \ldots, \theta_K\} \subseteq \mathbf{R}^n$ is the linear combination coefficient, and the vectors $u(t-\tau+1)$ with $t-\tau+1 \le 0$ for $\tau = 1, \ldots, n$ are padded with zero. Then the action $y(t) \in \{e_1, \ldots, e_p\} \subseteq \mathbf{R}^p$ is selected according to

$$y(t) \sim \mathrm{Cat}(\{e_1, \ldots, e_p\},\ \exp v(t)/\mathbf{1}^T \exp v(t)). \tag{6.9}$$

The fitting problem of a hierarchical forgetting Q-learning model consists in recovering the parameters $\{\theta_1, \ldots, \theta_K\}$ as well as the latent factor labels indicating which $\theta(t) \in \{\theta_1, \ldots, \theta_K\}$ was used at time step t, for all $t = 1, \ldots, m$, given a dataset $\{X(t), y(t)\}_{t=1}^m$ observed from an agent following (6.7) to (6.9).

Dataset and problem formulation. In this example, we consider a multi-armed bandit environment with the number of arms $p = 3$ and reward probabilities (0.1, 0.2, 0.7). The agent selects its action at each time step following (6.7) to (6.9) with $\theta(t) \in \{\theta_1, \theta_2\} \subseteq \mathbf{R}^5$ (*i.e.*, $K = 2$, $n = 5$), given by

$$\begin{aligned} \theta_1 &= (9.9,\ 9.9\times 10^{-2},\ 9.9\times 10^{-4},\ 9.9\times 10^{-6},\ 9.9\times 10^{-8}) \\ \theta_2 &= (-4,\ -0.8,\ -0.16,\ -0.032,\ -0.0064). \end{aligned}$$

These two groups of parameters can be interpreted as follows. Under parameters θ_1, the agent tends to exploit the action that was mostly rewarded in the last 5 trials, whereas under parameters θ_2, the agent tends to explore the action that had the least reward in the last 5 trials. In the fitting

problem, suppose we are given the prior information that θ_1 is nonnegative and nonincreasing, and θ_2 is nonpositive and nondecreasing. Such prior information corresponds to the following constraints:

$$\theta_1 \geq 0, \quad \theta_{1,1} \geq \cdots \geq \theta_{1,5}, \quad \theta_2 \leq 0, \quad \theta_{2,1} \leq \cdots \leq \theta_{2,5},$$

where $\theta_{1,i}$ denotes the ith entry of the vector θ_1. The dataset consists of $m = 200$ trials, where the agent starts with θ_1 and switches to another combination coefficient every 20 trials. Such a latent factor transition dynamics indicates that a regularization term on the latent factor labels should be applied such that its transition is sparse. Put together, the optimization problem for this example is given by

$$\begin{array}{ll} \text{minimize} & -\sum_{t=1}^{m} z(t)^T \log\left(\frac{y(t)^T \exp(X(t)\theta_1)}{\mathbf{1}^T \exp(X(t)\theta_1)}, \ \frac{y(t)^T \exp(X(t)\theta_2)}{\mathbf{1}^T \exp(X(t)\theta_2)}\right) \\ & \quad +\lambda \sum_{t=1}^{m-1} D_{\mathrm{kl}}(z(t), z(t+1)) \\ \text{subject to} & z(t) \in \{0,1\}^2, \quad \mathbf{card}\, z(t) = 1, \quad i = 1, \ldots, m \\ & \theta_1 \geq 0, \quad \theta_{1,1} \geq \cdots \geq \theta_{1,5} \\ & \theta_2 \leq 0, \quad \theta_{2,1} \leq \cdots \leq \theta_{2,5} \end{array} \tag{6.10}$$

(*cf.*, (3.4) and (3.5)). The problem (6.10) has variables $\theta_1, \theta_2 \in \mathbf{R}^5$, $z(1), \ldots, z(m) \in \mathbf{R}^2$, and data $X(1), \ldots, X(m) \in \mathbf{R}^{3\times 5}$, $y(1), \ldots, y(m) \in \mathbf{R}^3$. The regularization weight $\lambda \geq 0$ is the hyperparameter.

Problem specification. The problems we solve in each BCD iteration corresponding to (6.10) are given by

$$\begin{array}{lll} & \text{minimize} & \sum_{t=1}^{m} \tilde{z}(t)^T r(t) \\ \text{(P)} & \text{subject to} & r(t) = -\log\left(\frac{y(t)^T \exp(X(t)\theta_1)}{\mathbf{1}^T \exp(X(t)\theta_1)}, \ \frac{y(t)^T \exp(X(t)\theta_2)}{\mathbf{1}^T \exp(X(t)\theta_2)}\right), \quad t = 1, \ldots, m \\ & & \theta_1 \geq 0, \quad \theta_{1,1} \geq \cdots \geq \theta_{1,5} \\ & & \theta_2 \leq 0, \quad \theta_{2,1} \leq \cdots \leq \theta_{2,5}, \\ \text{(F)} & \text{minimize} & \sum_{t=1}^{m} z(t)^T \tilde{r}(t) + \lambda \sum_{t=1}^{m-1} D_{\mathrm{kl}}(z(t), z(t+1)) \\ & \text{subject to} & 0 \preceq z(t) \preceq \mathbf{1}, \quad \mathbf{1}^T z(t) = 1, \quad t = 1, \ldots, m. \end{array} \tag{6.11}$$

The P-problem has variables $\theta_1, \theta_2 \in \mathbf{R}^5$ and data $X(1), \ldots, X(m) \in \mathbf{R}^{3\times 5}$, $y(1), \ldots, y(m) \in \mathbf{R}^3$, $\tilde{z}(1), \ldots, \tilde{z}(m) \in \mathbf{R}^2$. The F-problem has variables $z(1), \ldots, z(m) \in \mathbf{R}^2$ and data $\tilde{r}(1), \ldots, \tilde{r}(m) \in \mathbf{R}^2$. The major adaptations we need to make to specify the problems (6.11) using our framework are given by the following code.

```
for k in range(K):
    thetas.append(cp.Variable(n))
    r.append(cp.hstack([-(xs[i] @ thetas[-1] @ ys[i] -
                cp.log_sum_exp(xs[i] @ thetas[-1])) for i in range(m)]))
Pconstr = [thetas[0] >= 0, cp.diff(thetas[0]) <= 0,
           thetas[1] <= 0, cp.diff(thetas[1]) >= 0]
Freg = lbd * cp.sum(cp.kl_div(z[:-1], z[1:]))
```

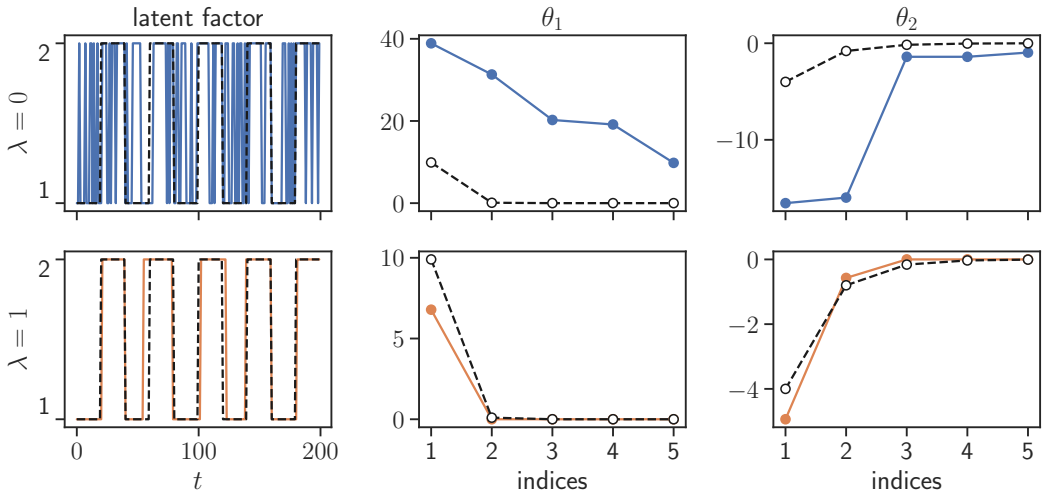

Figure 3 Results for the hierarchical forgetting Q-learning example under different λ values. The colored solid lines are the recovered latent factor labels (left) and model parameters (middle and right). The black dashed lines are the corresponding ground truth.

Numerical result. We tested the fitting performance under two different regularization weights, *i.e.*, $\lambda = 0$ and $\lambda = 1$, where the former condition is equivalent to removing the smoothness regularization term on $z(t)$. The results are shown in figure 3. It can be seen directly that if no regularization terms are added ($\lambda = 0$), the recovered latent factor labels switch drastically (figure 3, top left), which leads to only a 0.71 accuracy of recovering the correct label. As a result, the recovered model parameters θ_1 and θ_2 are far from the ground truth (figure 3, top middle and right). When we solve the problems (6.11) with hyperparameter $\lambda = 1$, on the contrary, the recovered latent factor labels align almost exactly to the ground truth (figure 3, bottom left), resulting in an 0.93 accuracy of latent factor label identification. Correspondingly, the recovered model parameters θ_1 and θ_2 are much closer to the ground truth value (figure 3, bottom middle and right).

6.4 Input-output hidden Markov model

Problem description. We consider a special case of the *input-output hidden Markov model* (IO-HMM) [BF94] with linear outputs, which has been widely used in neuroscience for behavior modeling and neural activities analysis [EFKP11, CPM19, BSP+22, ARS+22, JAP24]. Let $\hat{z} \in \{1, \dots, K\}$ be the latent factor label of a K-state IO-HMM with initial state distribution $p_{\text{init}} \in \mathbf{R}^K$ with $\mathbf{1}^T p_{\text{init}} = 1$ and transition matrix $P_{\text{tr}} \in \mathbf{R}^{K \times K}$ with $P_{\text{tr}}\mathbf{1} = \mathbf{1}$. At the time step t, the latent factor label $\hat{z}(t)$ is sampled according to

$$\hat{z}(t) \sim \begin{cases} \text{Cat}(p_{\text{init}}) & t = 0 \\ \text{Cat}(p_{\hat{z}(t-1)}) & t > 0, \end{cases}$$

where the vector $p_{\hat{z}(t-1)} \in \mathbf{R}^K$ denotes the $\hat{z}(t-1)$th row of the matrix P_{tr}. The input feature vector $x(t) \in \mathbf{R}^n$ to the IO-HMM at this time step is generated according to

$$x(t) = (\bar{x}(t),\ 1), \quad \bar{x}(t) \sim \mathcal{U}(a, b), \tag{6.12}$$

where $a, b \in \mathbf{R}^{n-1}$ ($a \prec b$) are the lower and upper bounds of the uniform distribution. Note that in (6.12), we implicitly integrate a bias term into the last entry of each $x(t)$. The output $y(t) \in \{0, 1\}$ of this IO-HMM at time step t is then generated from a logistic model, *i.e.*,

$$\mathbf{prob}(y(t) = 1) = \frac{1}{1 + \exp(-x(t)^T \theta_{\hat{z}(t)})},$$

where $\theta_{\hat{z}(t)} \in \{\theta_1, \ldots, \theta_K\} \subseteq \mathbf{R}^n$ is the linear combination coefficient. For the fitting problem of this IO-HMM, the objective is to recover the Markov process transition matrix P_{tr} (and the initial state distribution p_{init} if multiple observed sequences are given), the model parameters $\theta_1, \ldots, \theta_K$, and the unobserved latent factor labels $\hat{z}(1), \ldots, \hat{z}(m)$, given the dataset $\{x(t), y(t)\}_{t=1}^m$.

Dataset and problem formulation. In this example, we consider an IO-HMM with $K = 3$ and $n = 2$, given by the following parameters:

$$\begin{array}{l} a = -5, \quad b = 5 \\ \theta_1 = (-2, 0), \quad \theta_2 = (2, 6), \quad \theta_3 = (3, -5) \\ p_{\mathrm{init}} = (1, 0, 0), \quad P_{\mathrm{tr}} = \left[\begin{array}{ccc} 0.90 & 0.05 & 0.05 \\ 0.01 & 0.98 & 0.01 \\ 0.03 & 0.02 & 0.95 \end{array}\right]. \end{array}$$

The dataset $\{x(t), y(t)\}_{t=1}^m$ is a single sequence observed from this IO-HMM with $m = 500$ samples. To formulate the optimization problem, we assume that prior information about the model parameters $\theta_1, \theta_2, \theta_3$ is given by the following constraints:

$$\theta_{1,1} \leq 0, \quad \theta_{2,1} \geq 0, \quad \theta_{3,1} \geq 0.$$

Besides, we also integrate an ℓ_2-regularization term on the model parameters, and a smoothness regularization term on the latent factor labels as in (6.10). Put together, we have

$$\begin{array}{ll} \text{minimize} & -\sum_{t=1}^m z(t)^T \Big(y(t)x(t)^T\theta_k - \log(1 + \exp(x(t)^T\theta_k))\Big)_{k=1}^3 \\ & \quad +\lambda_\theta \sum_{k=1}^3 \|\theta_k\|_2 + \lambda_z \sum_{t=1}^{m-1} D_{\mathrm{kl}}(z(t), z(t+1)) \\ \text{subject to} & z(t) \in \{0, 1\}^3, \quad \mathbf{card}\, z(t) = 1, \quad i = 1, \ldots, m \\ & \theta_{1,1} \leq 0, \quad \theta_{2,1} \geq 0, \quad \theta_{3,1} \geq 0, \end{array} \tag{6.13}$$

where $\theta_1, \theta_2, \theta_3 \in \mathbf{R}^2$ and $z(1), \ldots, z(m) \in \mathbf{R}^3$ are the problem variables, $x(1), \ldots, x(m) \in \mathbf{R}^2$ and $y(1), \ldots, y(m) \in \{0, 1\}$ are the problem data, $\lambda_\theta \geq 0$ and $\lambda_z \geq 0$ are the hyperparameters. Note that the objective of recovering the transition matrix P_{tr} is not included in (6.13). Instead, the transition matrix P_{tr} is estimated according to the returned latent factor labels $z(1), \ldots, z(m)$ after solving (6.13).

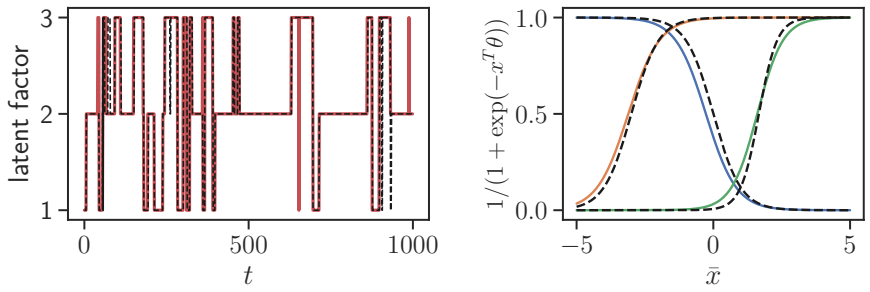

Figure 4 Results for the input-output hidden Markov model example. The colored solid lines represent the recovered latent factor labels (left) and the decision curve under parameters $\theta_1, \theta_2, \theta_3$, respectively (right). The black dashed lines are the corresponding ground truth.

Problem specification. The problems we solve in each BCD iteration corresponding to (6.13) are given by

$$
\begin{array}{lll}
 & \text{minimize} & \sum_{t=1}^{m} \tilde{z}(t)^T r(t) + \lambda_\theta \sum_{k=1}^{3} \|\theta_k\|_2 \\
\text{(P)} & \text{subject to} & r(t) = -\left(y(t)x(t)^T\theta_k - \log(1+\exp(x(t)^T\theta_k))\right)_{k=1}^{3}, \quad t = 1, \ldots, m \\
 & & \theta_{1,1} \leq 0, \quad \theta_{2,1} \geq 0, \quad \theta_{3,1} \geq 0, \\
 & & \\
 & \text{minimize} & \sum_{t=1}^{m} z(t)^T \tilde{r}(t) + \lambda_z \sum_{t=1}^{m-1} D_{\text{kl}}(z(t), z(t+1)) \\
\text{(F)} & \text{subject to} & 0 \preceq z(t) \preceq \mathbf{1}, \quad \mathbf{1}^T z(t) = 1, \quad t = 1, \ldots, m.
\end{array}
\tag{6.14}
$$

The P-problem has variables $\theta_1, \theta_2, \theta_3 \in \mathbf{R}^2$ and data $x(1), \ldots, x(m) \in \mathbf{R}^2$, $y(1), \ldots, y(m) \in \{0, 1\}$, $\tilde{z}(1), \ldots, \tilde{z}(m) \in \mathbf{R}^3$. The F-problem has variables $z(1), \ldots, z(m) \in \mathbf{R}^3$ and data $\tilde{r}(1), \ldots, \tilde{r}(m) \in \mathbf{R}^3$. The major adaptations we need to make to specify the problems (6.11) using our framework are given by the following code.

```
for k in range(K):
    thetas.append(cp.Variable(n))
    r.append(-(cp.multiply(ys, xs @ thetas[-1]) -
                cp.logistic(xs @ thetas[-1])))
Preg = lbd_theta * cp.sum(cp.norm2(cp.vstack(thetas), axis=1))
Pconstr = [thetas[0][0] <= 0, thetas[1][0] >= 0, thetas[2][0] >= 0]
Freg = lbd_z * cp.sum(cp.kl_div(z[:-1], z[1:]))
```

Numerical result. We specified and solved the problems (6.14) with hyperparameters $\lambda_\theta = 0.5$ and $\lambda_z = 1$. The results are shown in figure 4. The recovered transition matrix P_{tr} is

$$
\begin{bmatrix}
0.920 & 0.044 & 0.036 \\
0.003 & 0.978 & 0.019 \\
0.030 & 0.030 & 0.940
\end{bmatrix}.
$$

Acknowledgments

This work has been funded as part of BrainLinks-BrainTools, which is funded by the Federal Ministry of Economics, Science and Arts of Baden-Württemberg within the sustainability program for projects of the Excellence Initiative II, and CRC/TRR 384 “IN-CODE”.

References

[AAB+19] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, 32, 2019.

[ADHP09] D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75:245–248, 2009.

[ARS+22] Z. C. Ashwood, N. A. Roy, I. R. Stone, International Brain Laboratory, A. E. Urai, A. K. Churchland, A. Pouget, and J. W. Pillow. Mice alternate between discrete strategies during perceptual decision-making. *Nature Neuroscience*, 25(2):201–212, 2022.

[AVDB18] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[BF94] Y. Bengio and P. Frasconi. An input output HMM architecture. *Advances in Neural Information Processing Systems*, 7, 1994.

[BNLS22] C. C. Beron, S. Q. Neufeld, S. W. Linderman, and B. L. Sabatini. Mice exhibit stochastic and efficient action switching during probabilistic decision making. *Proceedings of the National Academy of Sciences*, 119(15):e2113961119, 2022.

[BSP+22] S. S. Bolkan, I. R. Stone, L. Pinto, Z. C. Ashwood, J. M. Iravedra Garcia, A. L. Herman, P. Singh, A. Bandi, J. Cox, C. A. Zimmerman, J. R. Cho, B. Engelhard, J. W. Pillow, and I. B. Witten. Opponent control of behavior by dorsomedial striatal pathways depends on task demands and internal state. *Nature Neuroscience*, 25(3):345–357, 2022.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[CPM19] A. J. Calhoun, J. W. Pillow, and M. Murthy. Unsupervised identification of the internal states that shape natural behavior. *Nature Neuroscience*, 22(12):2040–2049, 2019.

[DB16] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[EFKP11] S. Escola, A. Fontanini, D. Katz, and L. Paninski. Hidden Markov models for the stimulus-response relationships of multistate neural systems. *Neural Computation*, 23(5):1071–1132, 2011.

[GB08] M. C. Grant and S. P. Boyd. Graph implementations for nonsmooth convex programs. In *Recent Advances in Learning and Control*, pages 95–110. Springer, 2008.

[GB14] M. Grant and S. Boyd. CVX: MATLAB software for disciplined convex programming, version 2.1. http://cvxr.com/cvx, 2014.

[GBY06] M. Grant, S. Boyd, and Y. Ye. Disciplined convex programming. In L. Liberti and N. Maculan, editors, *Global Optimization: From Theory to Implementation*, volume 84 of *Nonconvex Optimization and Its Applications*, pages 155–210. Springer Science & Business Media, 2006.

[Gra04] M. Grant. *Disciplined Convex Programming*. PhD thesis, Stanford University, 2004.

[GS99] S. Gaffney and P. Smyth. Trajectory clustering with mixtures of regression models. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 63–72, 1999.

[ID09] M. Ito and K. Doya. Validation of decision-making models and analysis of decision variables in the rat basal ganglia. *Journal of Neuroscience*, 29(31):9861–9874, 2009.

[JAP24] A. Jha, Z. C. Ashwood, and J. W. Pillow. Active learning for discrete latent variable models. *Neural Computation*, 36(3):437–474, 2024.

[JMP21] A. Jha, M. J. Morais, and J. W. Pillow. Factor-analytic inverse regression for high-dimension, small-sample dimensionality reduction. In *International Conference on Machine Learning*, pages 4850–4859. PMLR, 2021.

[LL18] Y. Li and Y. Liang. Learning mixtures of linear regressions with nearly optimal complexity. In *Conference on Learning Theory*, pages 1125–1144. PMLR, 2018.

[Lof04] J. Lofberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *Proceedings of the IEEE International Symposium on Computed Aided Control Systems Design*, pages 294–289, 2004.

[Mur23] K. P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.

[NLS22] J. Nutini, I. Laradji, and M. Schmidt. Let's make block coordinate descent converge faster: Faster greedy rules, message-passing, active-set complexity, and superlinear convergence. *Journal of Machine Learning Research*, 23(131):1–74, 2022.

[NN92] Y. Nesterov and A. Nemirovsky. Conic formulation of a convex programming problem and duality. *Optimization Methods and Software*, 1(2):95–115, 1992.

[Pow73] M. J. D. Powell. On search directions for minimization algorithms. *Mathematical Programming*, 4(1):193–201, 1973.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[SDU+17] X. Shen, S. Diamond, M. Udell, Y. Gu, and S. Boyd. Disciplined multi-convex programming. In *29th Chinese Control and Decision Conference*, pages 895–900. IEEE, 2017.

[Tib96] R. Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.

[UMZ+14] M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, and S. Boyd. Convex optimization in Julia. In *Proceedings of the Workshop for High Performance Technical Computing in Dynamic Languages*, pages 18–28, 2014.

[War63] J. Warga. Minimizing certain convex functions. *Journal of the Society for Industrial and Applied Mathematics*, 11(3):588–593, 1963.

[ZB25] H. Zhu and J. Boedecker. Solving inverse problem for multi-armed bandits via convex optimization. *arXiv Preprint arXiv:2501.18945*, 2025.

[ZCK+24] H. Zhu, B. De La Crompe, G. Kalweit, A. Schneider, M. Kalweit, I. Diester, and J. Boedecker. Multi-intention inverse Q-learning for interpretable behavior representation. *Transactions on Machine Learning Research*, 2024.