Disciplined Multi-Convex Programming

Xinyue Shen, Steven Diamond, Madeleine Udell, Yuantao Gu, Stephen Boyd

1. Tsinghua University, Beijing 100084, China
2. Stanford University, CA 94305, U.S.
3. Cornell University, NY 14850, U.S.

Abstract: A multi-convex optimization problem is one in which the variables can be partitioned into sets over which the problem is convex when the other variables are fixed. Multi-convex problems are generally solved approximately using variations on alternating or cyclic minimization. Multi-convex problems arise in many applications, such as nonnegative matrix factorization, generalized low rank models, and structured control synthesis, to name just a few. In most applications to date the multi-convexity is simple to verify by hand. In this paper we study the automatic detection and verification of multi-convexity using the ideas of disciplined convex programming. We describe an implementation of our proposed method that detects and verifies multi-convexity, and then invokes one of the general solution methods.

Key Words: multi-convex optimization, multi-convexity verification, block coordinate descent

1 Introduction

A multi-convex optimization problem is one in which the variables can be partitioned into sets over which the problem is convex when the other variables are fixed. Multi-convex problems appear in domains such as control [1, 2], machine learning [3, 4], signal and information processing [5, 6], and communication [7].

In general multi-convex problems are hard to solve globally, but several algorithms have been proposed as heuristic or local methods, and are widely used in applications. Most of these methods are variations on the block coordinate descent (BCD) method. The idea of optimizing over a single block of variables while holding the remaining variables fixed in each iteration dates back to the 1960's [8, 9]. Convergence results were first discussed for strongly convex differentiable objective function [8], and then under various assumptions on the separability and regularity of the objective function [10, 11]. In [12] the authors propose an inexact BCD approach which updates variable blocks by minimizing a sequence of approximations of the objective function, which can be nondifferentiable or nonconvex. Recent work [13] uses BCD to solve multi-convex problems, where the objective is a sum of a differentiable multi-convex function and several extended-valued convex functions. Gradient methods have also been proposed for multi-convex problems, where the objective is differentiable, and all variables are updated at once along descent directions and then projected into a convex feasible set [14].

The focus of this paper is not on solution methods, but on a modeling framework for expressing multi-convex problems in a way that verifies the multi-convex structure, and can expose the structure to whatever solution algorithm is then used. Modeling frameworks have been developed for convex problems, e.g., CVX [15], YALMIP [16], CVXPY [17], and Convex.jl [18]. These frameworks provide a uniform method for specifying convex problems based on the idea of disciplined convex programming (DCP) [19]. This gives a simple method for verifying convexity and automatically canonicalizing to a standard generic form such as a cone program.

In this paper we extend the idea of DCP to multi-convex problems, and propose disciplined multi-convex programming (DMCP). Problem specifications that conform to the DMCP rule set can be automatically verified as convex in a group of variables, for any fixed values of the other variables. As with DCP, the goal of DMCP is not to analyze multi-convexity of an arbitrary problem, but rather to give a simple set of rules which if followed yields multi-convex problems. In applications to date, such as nonnegative matrix factorization, verification of multi-convexity can be done by hand or just simple observation. With DMCP a larger class of multi-convex problems can be constructed in an organized way. A software implementation of the ideas developed in this paper is described, called DMCP, a Python package that extends CVXPY. It implements the DMCP verification and analysis methods, and then heuristically solves a conforming problem via BCD type algorithms, which we extend for general use to include slack variables. A similar package, MultiConvex, has been developed for the Julia package Convex.jl. We illustrate the framework on a number of examples.

2 Multi-convex programming

2.1 Multi-convex function

Consider a function \( f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\} \), and a partition of the variable \( x \in \mathbb{R}^n \) into blocks of variables

\[
x = (x_1, \ldots, x_N), \quad x_i \in \mathbb{R}^{n_i}, \quad \sum_{i=1}^N n_i = n.
\]

Throughout this paper we will use subsets of indices to refer to sets of the variables. Let \( \mathcal{F} \subseteq \{1, \ldots, N\} \) denote...
an index set, with complement \( F^c = \{1, \ldots, N\} \setminus F \). By fixing the variables with indices in \( F \) of the function \( f \) at a given point \( \hat{x} \in \mathbb{R}^n \), we obtain a function over the remaining variables, with indices in \( F^c \), which we denote as \( \hat{f} = f(\hat{x}, \cdot, F) \). For \( i \in F^c \), \( x_i \) is a variable of the function \( \hat{f} \); for \( i \in F \), \( x_i = \hat{x}_i \).

Given an index set \( F \subseteq \{1, \ldots, N\} \), we say that a function \( f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\} \) is convex (or affine) with set \( F \) fixed, if for any \( \hat{x} \in \mathbb{R}^n \) the function \( \hat{f}(\hat{x}, \cdot, F) \) is a convex (or affine) function.

We say the function \( f \) is multi-convex (or multi-affine), if there are index sets \( F_1, \ldots, F_K \), such that for every \( k \) the function \( f \) is convex (or affine) with \( F_k \) fixed, and \( \cap_{k=1}^K F_k = \emptyset \). For \( K = 2 \), we say that the function is bi-convex (or bi-affine).

For a function \( f \), we can consider the set of all index sets \( F \) for which \( \text{fix}(f, \cdot, F) \) is convex for all \( \hat{x} \); among these we are interested in the minimal fixed sets that render a function convex. A minimal fixed set is a set of variables that when fixed make the function convex; but if any variable is removed from the set, the function is not convex. A function is multi-convex if and only if the intersection of these minimal fixed index sets is empty.

### 2.2 Multi-convex problem

We now extend the idea of multi-convexity to the optimization problem

\[
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq 0, \quad i = 1, \ldots, m \\
& \quad g_i(x) = 0, \quad i = 1, \ldots, p,
\end{align*}
\]

(1)

with variable \( x \in \mathbb{R}^n \) partitioned into blocks as \( x = (x_1, \ldots, x_N) \), and functions \( f_i : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\} \) for \( i = 0, \ldots, m \) and \( g_i : \mathbb{R}^n \to \mathbb{R} \) for \( i = 1, \ldots, p \) are proper.

Given an index set \( F \subseteq \{1, \ldots, N\} \), problem (1) is convex with set \( F \) fixed, if for any \( \hat{x} \in \mathbb{R}^n \) the problem

\[
\begin{align*}
\text{minimize} & \quad \text{fix}(f_0, \hat{x}, F) \\
\text{subject to} & \quad \text{fix}(f_i, \hat{x}, F) \leq 0, \quad i = 1, \ldots, m \\
& \quad \text{fix}(g_i, \hat{x}, F) = 0, \quad i = 1, \ldots, p,
\end{align*}
\]

(2)

is convex.

We say the problem (1) is multi-convex, if there are sets \( F_1, \ldots, F_K \), such that for every \( k \) problem (1) is convex with set \( F_k \) fixed, and \( \cap_{k=1}^K F_k = \emptyset \). A convex problem is multi-convex with \( K = 0 \) (i.e., \( F = \emptyset \)). A bi-convex problem is multi-convex with \( K = 2 \).

### 3 Block coordinate descent and variations

In this section we review, and extend, some generic methods for approximately solving the multi-convex problem (1), using block coordinate descent (BCD).

#### 3.1 BCD with slack variables

Assume that sets \( F_k, k = 1, \ldots, K \), are index sets for which the problem (1) with \( F_k \) fixed is convex, with \( \cap_{k=1}^K F_k = \emptyset \). These could be the set of all minimal fixed sets, but any other set of index sets that verify multi-convexity could be used.

The basic form of the proposed method is iterative. In each iteration, we fix the variables in one set \( F_k \) and solve the following subproblem,

\[
\begin{align*}
\text{minimize} & \quad \text{fix}(f_0, \hat{x}, F_k) + \mu \sum_{i=1}^m s_i + \mu \sum_{i=1}^p |s_{i+m}| \\
\text{subject to} & \quad \text{fix}(f_i, \hat{x}, F_k) \leq s_i, \quad s_i \geq 0, \quad i = 1, \ldots, m \\
& \quad \text{fix}(g_i, \hat{x}, F_k) = s_{i+m}, \quad i = 1, \ldots, p,
\end{align*}
\]

(3)

where \( s_i \) for \( i = 1, \ldots, m + p \) are the variables, and \( \mu > 0 \) is a parameter. Here the constant \( \hat{x} \) inherits the value of \( x \) from the least iteration. This subproblem solved in each iteration is convex. The slack variables \( s_i \) for \( i = 1, \ldots, m + p \) ensure that the subproblem (3) cannot be infeasible. The added terms in the objective are a so-called exact penalty [20], meaning that when some technical conditions hold and the subproblem without the slack variables is feasible, for large enough \( \mu \) the solution satisfies \( s_i = 0 \). This technique has been widely used [21, 22].

This algorithm differs from the basic BCD algorithm in the addition of the slack variables, and in the feature that a variable can appear in more than one set \( F_k \), meaning that a variable can be updated in multiple iterations per round of \( K \) iterations. For example, if a problem has variables \( x_1, x_2, x_3 \) and is convex in \((x_1, x_2)\) and \((x_2, x_3)\), our method will update \( x_2 \) in each step.

#### 3.2 Block coordinate proximal iteration

A variation of subproblem (3) adds a proximal term [23], which renders the subproblems strongly convex:

\[
\begin{align*}
\text{minimize} & \quad \text{fix}(f_0, \hat{x}, F_k) + \mu \sum_{i=1}^m s_i + \mu \sum_{i=1}^p |s_{i+m}| \\
& \quad + \frac{1}{\lambda} \sum_{i \in F_k^c} \|x_i - \hat{x}_i\|^2 \\
\text{subject to} & \quad \text{fix}(f_i, \hat{x}, F_k) \leq s_i, \quad s_i \geq 0, \quad i = 1, \ldots, m \\
& \quad \text{fix}(g_i, \hat{x}, F_k) = s_{i+m}, \quad i = 1, \ldots, p,
\end{align*}
\]

(4)

where \( x_i \) for \( i \in F_k^c \) and \( s_i \) for \( i = 1, \ldots, m + p \) are variables, and \( \lambda > 0 \) is the proximal parameter. The proximal term penalizes large changes in the variables being optimized, i.e., it introduces damping into the algorithm.

In some cases it has been observed to yield better final points, i.e., points with smaller objective value, than those obtained without proximal regularization.

Yet another variation uses linearized proximal steps, when \( f \) is differentiable in the variables \( x_i \) for \( i \in F_k^c \). The subproblem solved in this case is

\[
\begin{align*}
\text{minimize} & \quad \mu \sum_{i=1}^m s_i + \mu \sum_{i=1}^p |s_{i+m}| \\
& \quad + \sum_{i \in F_k^c} \left(\frac{\mu}{\lambda} \|x_i - \hat{x}_i\|^2 + (x_i - \hat{x}_i)^T \nabla f(\hat{x}_i)\right) \\
\text{subject to} & \quad \text{fix}(f_i, \hat{x}, F_k) \leq s_i, \quad s_i \geq 0, \quad i = 1, \ldots, m \\
& \quad \text{fix}(g_i, \hat{x}, F_k) = s_{i+m}, \quad i = 1, \ldots, p,
\end{align*}
\]

(5)

where \( x_i \) for \( i \in F_k^c \) and \( s_i \) for \( i = 1, \ldots, m + p \) are variables, and \( \nabla f(\hat{x}_i) \) is the partial gradient of \( f \) with respect to \( x_i \) at the point \( \hat{x} \). The objective is equivalent to the minimization of

\[
\mu \sum_{i=1}^m s_i + \mu \sum_{i=1}^p |s_{i+m}| + \sum_{i \in F_k^c} \frac{1}{2\lambda} \|x_i - \hat{x}_i + \lambda \nabla f(\hat{x}_i)\|^2,
\]

which is the objective of a proximal gradient method.
3.3 Generalized inequality constraints

One useful extension is to generalize problem (1) by allowing generalized inequality constraints. Suppose the functions $f_i$ and $g_i$ for $i = 1, \ldots, p$ are the same as in problem (1), but $f_i : \mathbb{R}^n \to \mathbb{R}^m \cup \{\infty\}$ for $i = 1, \ldots, m$. Consider the following program with generalized inequalities.

$$
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq K_i, \quad i = 1, \ldots, m \\
& \quad g_i(x) = 0, \quad i = 1, \ldots, p,
\end{align*}
$$

where $x = (x_1, \ldots, x_N) \in \mathbb{R}^n$ is the variable, and the generalized inequality constraints are with respect to proper cones $K_i \subseteq \mathbb{R}^d$, $i = 1, \ldots, m$. The definitions of multi-convex program and minimal set can be directly extended. Slack variables are added in the following way:

$$
\begin{align*}
\text{minimize} & \quad \text{fix}(f_0, \hat{x}, F_k) + \mu \sum_{i=1}^m s_i + \mu \sum_{i=m+1}^p s_{i+m} \\
\text{subject to} & \quad \text{fix}(f_i, \hat{x}, F_k) \leq K_i, s_{i+e(i)}, \quad i = 1, \ldots, m \\
& \quad s_i \geq 0, \quad i = 1, \ldots, m \\
& \quad \text{fix}(g_i, \hat{x}, F_k) = s_{i+m}, \quad i = 1, \ldots, p,
\end{align*}
$$

where $e_i$ is a given positive element in cone $K_i$ for $i = 1, \ldots, m$.

4 Disciplined multi-convex programming

4.1 Disciplined convex programming

Disciplined convex programming (DCP) is a methodology that imposes a set of conventions that must be followed when constructing convex programs [19]. Conforming problems are called disciplined convex programs, which can be automatically verified and transformed into an equivalent cone program by replacing each function with its graph implementation [24].

Every function in a disciplined convex program must be formed as an expression involving constants or parameters, variables, and a dictionary of atomic functions, and certain function composition rules must be followed. Signed DCP is an extension of DCP that keeps track of the signs of functions using simple sign arithmetic. The monotonicity of functions in the atom library can then depend on the sign of their arguments.

We make an observation that is critical for our work here: The DCP analysis does not use the values of any constants or parameters in the expression. The number 4.57 is simply treated as positive; if a parameter has been declared as positive, then it is treated as positive. It follows immediately that DCP analysis has verified not just that the specific expression is convex, but that it is convex for any other values of the constants and parameters, with the same signs as the given ones, if the sign matters.

4.2 Disciplined multi-convex programming

To determine that problem (1) is multi-convex requires us to verify that functions $\text{fix}(f_i, \hat{x}, F)$ are convex, and that $\text{fix}(g_i, \hat{x}, F)$ are affine, for all $\hat{x} \in \mathbb{R}^n$. We can use the idea of DCP, specifically with signed parameters, to carry this out, which gives us a practical method for multi-convexity verification.

4.2.1 Multi-convex atoms

We start by generalizing the library of DCP atom functions to include multi-convex atomic functions. A function is a multi-convex atom if it has $N$ arguments $N > 1$, and it reduces to a DCP atomic function, when and only when all but the $i$th arguments are constant for each $i = 1, \ldots, N$.

Given a description of problem (1) under a library of DCP and multi-convex atomic functions, we say that it is disciplined convex programming with set $F \subseteq \{1, \ldots, N\}$ fixed, if the corresponding problem (2) for any $\hat{x} \in \mathbb{R}^n$ conforms to the DCP rules with respect to the DCP atomic function set. When there is no confusion, we simply say that problem (1) is DCP with $F$ fixed.

To verify if a problem is DCP with $F$ fixed, a method first fixes the problem by replacing variables in $F$ with parameters of the same signs and dimensions. Then it verifies DCP of the fixed problem with parameters according to the DCP ruleset. The parameter is the correct model for fixed variables, in that the DCP rules ensure that the verified curvature holds for any value of the parameter.

4.2.2 Disciplined multi-convex program

Given a description of problem (1) under a library of DCP and multi-convex atomic functions, it is disciplined multi-convex programming (DMCP), if there are sets $F_1, \ldots, F_K$ such that problem (1) with every $F_k$ fixed is DCP, $\cap_{k=1}^K F_k = \emptyset$. We simply say that problem (1) is DMCP if there is no confusion. A problem that is DMCP is guaranteed to be multi-convex, just as a problem that is DCP is guaranteed to be convex. Moreover, when a BCD method is applied to a DMCP problem, each iteration involves the solution of a DCP problem.

A direct way of DMCP verification is to check if the problem is DCP with $\{i\}^c$ fixed for every $i = 1, \ldots, N$ because of the following claim.

Claim 4.1 For a problem consisting only of DCP atoms (or multi-convex atoms with all but one arguments constant), it is DCP with $F$ fixed, if and only if it is DCP with $\{i\}^c$ fixed for all $i \in F^c$.

Its proof is based on the hierarchy of curvature types in DCP, and we include it in a longer paper of this work [25].

4.3 Efficient search for minimal sets

A problem may have multiple collections of index sets for which it is DMCP. The simplest option is to always choose the collection $\{1\}^c, \ldots, \{N\}^c$, in which case BCD optimizes over one variable at a time. A more sophisticated, and usually better, approach is to reduce the collection $\{1\}^c, \ldots, \{N\}^c$ to minimal sets, which allows BCD to optimize over multiple variables each iteration.

We find minimal sets by first determining which variables can be optimized together. Concretely, we construct a conflict graph $(V, E)$, where $V$ is the set of all variables, and $i \sim j \in E$ if and only if variables $i$ and $j$ appear in two different arguments of a multi-convex atom, which means the variables cannot be optimized together. More details on
how to efficiently construct such a graph are explained in a longer paper of this work [25].

Given the conflict graph, for \( i = 1, \ldots, N \), we find a maximal independent set \( F_i \) containing \( i \) using a standard fast algorithm and replace \( [i]^c \) with \( F_i^c \). The final collection is all index sets \( F_i^c \) that are not supersets of another index set \( F_j^c \). More generally, we can choose any collection of index sets \( F_1, \ldots, F_K \) such that \( F_1^c, \ldots, F_K^c \) are independent sets in the conflict graph.

5 Implementation

The methods of DMCP verification, the search of minimal sets to fix, and the cyclic optimization with minimal sets fixed are implemented as an extension of CVXPY in a package DMCP that can be accessed at https://github.com/cvxgrp/dmcp.

5.1 Some useful functions

Multi-convex atomic functions. In order to allow multi-convex functions, we extend the atomic function set of CVXPY. The following atoms are allowed to have non-constant expressions in both arguments, while in base CVXPY one of the arguments must be constant.

- multiplication: \( \text{expr1} \ast \text{expr2} \)
- elementwise multiplication: \( \text{mul}_\text{elemwise} (\text{expr1}, \text{expr2}) \)
- convolution: \( \text{conv} (\text{expr1}, \text{expr2}) \)

Find minimal sets. Given a problem, the function

\[
\text{find_minimal_sets} (\text{problem})
\]

runs the algorithm discussed in §4.3 and returns a list of minimal sets of indices of variables. The indices are with respect to the list \( \text{problem}.\text{variables}() \), namely, the variable corresponding to index 0 is \( \text{problem}.\text{variables}()[0] \).

DMCP verification. Given a problem, the function

\[
\text{is_dmcp} (\text{problem})
\]

returns a boolean indicating if it is a DMCP problem.

Fix variables. The function

\[
\text{fix} (\text{expression}, \text{fix}_\text{vars})
\]

returns a new expression with the variables in the list \( \text{fix}_\text{vars} \) replaced with parameters of the same signs and values. If \( \text{expression} \) is replaced with a CVXPY problem, then a new problem with the corresponding variables fixed is returned.

Random initialization. It is suggested that users provide an initial point \( x^0 \) for the method such that functions \( \text{fix}(f_i, x^0, F_i) \) are proper for \( i = 0, \ldots, m \), where \( F_i \) is the first minimal set given by \( \text{find_minimal_sets} \). If not, the function \( \text{rand_initial} (\text{problem}) \) will be called to generate random values from the uniform distribution over the interval \([0, 1]\) \( (\{-1, 0\} \) for variables with non-negative (non-positive) sign, and from the standard normal distribution for variables with no sign. There is no guarantee that such a simple random initialization can always work for any problem.

5.2 Update options and algorithm parameters

The solving method is to cyclically fix every minimal set found by \( \text{find_minimal_sets} \) and update the variables. Three ways of updating variables are implemented. The default one can be called by \( \text{problem}.\text{solve} (\text{method} = \text{bcd}, \text{update} = \text{proximal}) \), which is to solve the subproblem with proximal operators, \( i.e. \), problem (4). To update by minimizing the subproblem without proximal operators, \( i.e. \), problem (3), the solve method is called with \( \text{update} = \text{minimize} \). To use the prox-linear operator in updates, \( i.e. \), problem (5), the solve method should be called with \( \text{update} = \text{prox_linear} \).

The parameter \( \mu \) is updated in every cycle by \( \mu_{t+1} = \min (\rho \mu, \mu_{\text{max}}) \). The algorithm parameters are \( \rho, \mu_0, \mu_{\text{max}}, \lambda \), and the maximum number of iterations. They can be set by passing values of the parameters \( \rho, \mu_0, \mu_{\text{max}}, \lambda \), and \( \text{max_iter} \), respectively, to the \( \text{solve} \) method.

6 Numerical examples

6.1 A basic example

First we give a Hello World example. For the problem

\[
\begin{align*}
\text{minimize} \quad & |x_1 x_2 + x_3 x_4 | \\
\text{subject to} \quad & x_1 + x_2 + x_3 + x_4 = 1,
\end{align*}
\]

the code written in DMCP is as follows.

\[
\begin{align*}
\text{prob} &= \text{Problem} (\text{obj}, \text{constr})
\end{align*}
\]

To find all minimal sets, we call \( \text{find_minimal_sets} (\text{prob}) \), which gives output \( [[2, 1], [3, 1], [2, 0], [3, 0]] \), where index \( i \) corresponds to \( x_{i+1} \). To verify if it is DMCP, \( \text{is_dmcp} (\text{prob}) \) returns True. The solution method with default setting is called by \( \text{prob}.\text{solve} (\text{method} = \text{bcd}) \). This finds a feasible point with objective value nearly 0, which solves the problem globally.

6.2 Sparse feedback matrix design

Problem description. We seek a sparse linear constant output feedback control \( u = Ky \) for the system

\[
\dot{x} = Ax + Bu, \quad y = Cx,
\]
which results in a decay rate $r$ no less than a given threshold $\theta > 0$ in the closed-loop system. We formulate this as the following optimization problem [1, 26]

$$\begin{align*}
\text{minimize} & \quad \sum_{ij} |K_{ij}| \\
\text{subject to} & \quad P \succeq I, \quad r \geq \theta \\
& \quad -2rP \geq (A + BK C)^T P + P (A + BK C),
\end{align*}$$

where $K$, $P$, and $r$ are variables, and $A$, $B$, $C$, and $\theta$ are given. The notation $P \succeq I$ means that $P - I$ is semidefinite. The problem is biconvex with minimal sets of variables to fix $\{P\}$ and $\{K, r\}$.

**DMCP specification.** The following code specifies and solves the problem.

```python
P = Variable(n,n)
K = Variable(m1,m2)
r = Variable(1)
cost = norm(K,1)
constr = [np.eye(n) << P, r >= theta]
constr += [(A+B*K*C).T*P+P*(A+B*K*C) << -rP*2]
prob = Problem(Minimize(cost), constr)
prob.solve(method = 'bdc')
```

**Numerical result.** An example with $n = m_1 = 5$, $m_2 = 4$, $\theta = 0.01$ is tested, where the matrices $A$, $B$, and $C$ are the same ones used in [1]. We use initial values $P^0 = I$, $r^0 = 1$, and $K^0 = 0$. The solution found is $r = 0.01$ and

$$K = \begin{bmatrix}
0 & 0.32 & 0 & 0 \\
0 & -0.46 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0.11 & 0 & 0
\end{bmatrix},$$

which is sparse. The three nonzero entries are in the second column, so only the second output $\dot{y}_2$ is used in the control. (In [1] a different sparse feedback matrix is found, also with three nonzero entries.)

### 6.3 Bilinear control

**Problem description.** A discrete time $m$-input bilinear control system is of the following form [2]

$$x_{t+1} = Ax_t + \sum_{i=1}^m (u_t)_i B^i x_t, \quad t = 1, \ldots, n - 1,$$

where $u_t \in \mathbb{R}^m$ is the input, and $x_t \in \mathbb{R}^d$ is the system state at time $t$. In an optimal control problem with fixed initial state, given system matrices $A$, $B^i \in \mathbb{R}^{n \times n}$ and convex objective functions $f$ and $g$, an optimization can be formulated as

$$\begin{align*}
\text{minimize} & \quad f(x) + g(u) \\
\text{subject to} & \quad x_1 = x_{ini}, \\
& \quad (x_t, u_t) \in \Omega, \quad t = 1, \ldots, n - 1 \\
& \quad x_{t+1} = Ax_t + \sum_{i=1}^m (u_t)_i B^i x_t, \\
& \quad t = 1, \ldots, n - 1,
\end{align*}$$

where $x_t$ and $u_t$ are variables, and $\Omega$ is a given convex set describing bounds on $u_t$ and $x_t$. The problem is multiconvex.

As a special case, a standard continuous-time model of D.C.-motor is a bilinear system of the following form [27]

$$\dot{x} = A_0 x + u A_1 x + b u.$$ 

Here $x_1$ is the armature current, $x_2$ is the speed of rotation, $u$ is the field current, and $v$ is the armature voltage. For nominal operation, $x = (1, 1)$, and $u = v = 1$. A control problem is the braking with short-circuited armature, where the field current $u$ is controlled such that the rotation speed decreases to zero as fast as possible, and that the armature current is not excessively large. By discretizing over time with 10 samples per second, the problem can be formulated as follows

$$\begin{align*}
\text{minimize} & \quad \sqrt{\sum_{t=1}^n (x_{t+1})^2} \\
\text{subject to} & \quad x_1 = (1, 1) \\
& \quad \max_{s=1, \ldots, n} |(x_t)_1| \leq M \\
& \quad (x_{t+1} - x_t)/0.1 = A_0 x_t + u_t A_1 x_t, \\
& \quad t = 1, \ldots, n - 1,
\end{align*}$$

where $u_t \in \mathbb{R}$ and $x_t \in \mathbb{R}^2$ for $t = 1, \ldots, n$ are variables. The problem is biconvex if we consider $x = (x_1, \ldots, x_n)$ as one variable and $u = (u_1, \ldots, u_t)$ as the other one.

**DMCP specification.** The following code specifies and solves the problem.

```python
x = Variable(2,n)
u = Variable(n-1)
constr = [x[:,0] == 1,
          max_entries(abs(x[0,:])) <= M]
for t in range(n-1):
    constr += [x[:,t+1]-x[:,t] <= 0.1*(A0*x[:,t]+A1*x[:,t]*u[:,t]),
                obj = Minimize(norm(x[1,:]))]
prob = Problem(obj, constr)
prob.solve(method = 'bdc')
```

**Numerical result.** We take an example with $n = 100$, $M = 8$, and

$$A_0 = \begin{bmatrix} -1 & 0 \\ 0 & -0.1 \end{bmatrix}, \quad A_1 = \begin{bmatrix} 0 & -19 \\ 0.1 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 20 \\ 0 \end{bmatrix}.$$ 

We use initial values $x = 0$, and for $u$ a vector linearly decreasing from 0.5 to 0. The result is shown in Figure 1, where the braking is faster than that in a linear control system shown in [27].

**REFERENCES**


Figure 1: D.C.-motor braking.


