End-to-End Learning to Warm-Start for Real-Time Quadratic Optimization

Rajiv Sambharya RAJIVS@PRINCETON.EDU
Operations Research and Financial Engineering, Princeton University, Princeton, NJ, USA

Georgina Hall GEORGINA.HALL@INSEAD.EDU
Decision Sciences, INSEAD, Fontainebleau, France

Brandon Amos BDA@FB.COM
Meta AI, New York City, NY, USA

Bartolomeo Stellato BSTELLATO@PRINCETON.EDU
Operations Research and Financial Engineering, Princeton University, Princeton, NJ, USA

Editors: N. Matni, M. Morari, G. J. Pappas

Abstract

First-order methods are widely used to solve convex quadratic programs (QPs) in real-time applications because of their low per-iteration cost. However, they can suffer from slow convergence to accurate solutions. In this paper, we present a framework which learns an effective warm-start for a popular first-order method in real-time applications, Douglas-Rachford (DR) splitting, across a family of parametric QPs. This framework consists of two modules: a feedforward neural network block, which takes as input the parameters of the QP and outputs a warm-start, and a block which performs a fixed number of iterations of DR splitting from this warm-start and outputs a candidate solution. A key feature of our framework is its ability to do end-to-end learning as we differentiate through the DR iterations. To illustrate the effectiveness of our method, we provide generalization bounds (based on Rademacher complexity) that improve with the number of training problems and the number of iterations simultaneously. We further apply our method to three real-time applications and observe that, by learning good warm-starts, we are able to significantly reduce the number of iterations required to obtain high-quality solutions.

Keywords: Machine learning, real-time optimization, quadratic optimization, warm-start, generalization bounds.

1. Introduction

We consider the problem of solving convex quadratic programs (QPs) within strict real-time computational constraints using first-order methods. QPs arise in various real-time applications in robotics (Kuindersma et al., 2014), control (Borrelli et al., 2017), and finance (Boyd et al., 2017). In the past decade, first-order methods have gained wide popularity in real-time quadratic optimization (Boyd et al., 2011; Beck, 2017; Ryu and Yin, 2022) due to their low cost per iteration and their warm-starting capabilities. However, they still suffer from slow convergence to the optimal solutions, especially for badly-scaled problems (Beck, 2017). As a workaround to this issue, one can make use of the oftentimes parametric nature of the QPs which feature in real-time applications. For example, one can use the solution to a previously solved QP as a warm-start to a new problem (Ferreau et al., 2014; Stellato et al., 2020). Although this approach is popular, it only makes use

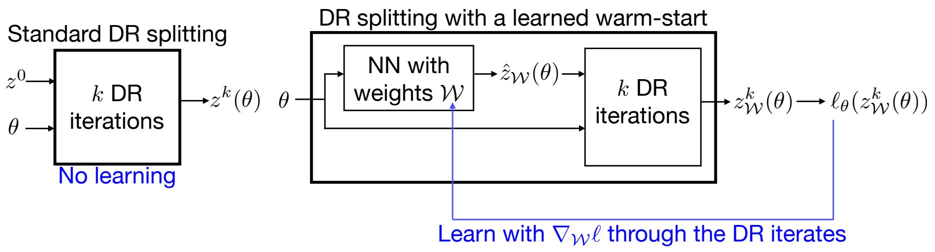
Standard DR splitting
z^0
θ
k DR iterations
$z^k(\theta)$
No learning
DR splitting with a learned warm-start
θ
NN with weights $\mathcal{W}$
$\hat{z}_{\mathcal{W}}(\theta)$
k DR iterations
$z^k_{\mathcal{W}}(\theta)$
$\ell_\theta(z^k_{\mathcal{W}}(\theta))$
Learn with $\nabla_{\mathcal{W}}\ell$ through the DR iterates

Figure 1: Left: standard DR splitting which maps parameter θ and initialization z^0 to an approximate solution $z^k(\theta)$. Right: Proposed learning framework consisting of two modules. The first module is the NN block which maps the parameter θ to a warm-start $\hat{z}_{\mathcal{W}}(\theta)$. The weights of the NN, denoted by w, are the only variables we optimize over. The second module runs k iterations of DR splitting (which also depend on θ) starting with the warm-start $\hat{z}_{\mathcal{W}}(\theta)$ and returning a candidate solution $z^k_{\mathcal{W}}(\theta)$. We backpropagate from the loss $\ell_\theta(z^k_{\mathcal{W}}(\theta))$ through the DR iterates to learn the optimal weights $\mathcal{W}$.

of the data from the previous problem, neglecting the vast majority of data available. More recent approaches in machine learning have sought to exploit data by solving many different parametric problems offline to learn a direct mapping from the parameters to the optimal solutions. The learned solution is then used as a warm-start (Chen et al., 2022; Baker, 2019). These approaches fail to consider the specific characteristics of the algorithm that will run on this warm-start downstream, and they require solving many optimization problems to optimality, which can be expensive. Furthermore, such learning schemes often do not provide generalization guarantees (Amos, 2022) on the algorithmic performance on unseen data. Such guarantees are crucial for real-time and safety critical applications where the algorithms must return high-quality solutions within strict time limits.

Contributions. In this work, we exploit data to learn a mapping from the parameters of the QP to a warm-start of a popular first-order method, Douglas-Rachford (DR) splitting. The goal is to decrease the number of real-time iterations of DR splitting that are required to obtain a good-quality solution in real-time. Our contributions are the following:

- We propose a principled framework to learn high quality warm-starts from data. This framework consists of two modules as indicated in Figure 1. The first module is a feedforward neural network (NN) that predicts a warm-start from the problem parameters. The second module consists of k DR splitting iterations that output the candidate solution. We differentiate the loss function with respect to the neural network weights by backpropagating through the DR iterates, which makes our framework an end-to-end warm-start learning scheme. Furthermore, our approach does not require us to solve optimization problems offline.

- We combine operator theory and Rademacher complexity theory to obtain novel generalization bounds that guarantee good performance for parametric QPs with unseen data. The bounds improve with the number of training problems and the number of DR iterations simultaneously, thereby allowing great flexibility in our learning task.

- We benchmark our approach on real-time quadratic optimization examples, showing that our method can produce an excellent warm-start that reduces the number of DR iterations required to reach a desired accuracy by at least 30% and as much as 90%.

2. Related work

Learning warm-starts. A common approach to reduce the number of iterations of iterative algorithms is to learn a mapping from problem parameters to high-quality initializations. Baker (2019) trains a random forest to predict a warm-start for the optimal power flow problem. In the model predictive control (MPC) (Borrelli et al., 2017) paradigm, Chen et al. (2022) use a neural network to accelerate the optimal control law computation by warm-starting an active set method. Other works in MPC use machine learning to predict an approximate optimal solution and, instead of using it to warm-start an algorithm, directly ensure feasibility and optimality. Chen et al. (2018) and Karg and Lucia (2020) use a constrained neural network architecture that guarantees feasibility by projecting its output onto the QP feasible region. Zhang et al. (2019) uses a neural network to predict the solution while also certifying suboptimality of the output. In these works, the machine learning models do not consider that additional algorithmic steps will be performed after warm-starting. Our work differs in that the training of the NN is designed to minimize the loss after many steps of DR splitting. Additionally, our work is more general in scope since we consider general parametric QPs.

Learning algorithm steps. There has been a wide array of works to speedup machine learning tasks by tuning algorithmic steps of stochastic gradient descent methods (Li and Malik, 2016; Andrychowicz et al., 2016; Metz et al., 2022; Chen et al., 2021; Amos, 2022). Similarly, Gregor and LeCun (2010) and Liu et al. (2019) accelerate the solution of sparse encoding problems by learning the steps of the iterative soft thresholding algorithm. Operator splitting algorithms (Ryu and Yin, 2022) can also be sped up by learning acceleration steps (Venkataraman and Amos, 2021) or the closest contractive fixed-point iteration to achieve fast convergence (Bastianello et al., 2021). Reinforcement learning has gained popularity as a versatile technique to accelerate the solution of parametric QPs by learning a policy to tune the step size of first-order methods (Ichnowski et al., 2021; Jung et al., 2022). A common tactic in these works is to differentiate through the steps of an algorithm to minimize a performance loss using gradient-based methods. This known as *loop unrolling* which has been used in other areas such as meta-learning (Finn et al., 2017) and variational autoencoders (Kim et al., 2018). While we also unroll the algorithm iterations, our work differs in that we learn a high-quality warm-start rather than the algorithm steps. This allows us to guarantee convergence and also provide generalization bounds over the number of iterations.

Learning surrogates. Instead of solving the original parametric problem, several works aim to learn a surrogate model that can be solved quickly in real-time applications. For example, by predicting which constraints are active (Misra et al., 2022) and the value of the optimal integer solutions (Bertsimas and Stellato, 2021, 2019) we can significantly accelerate the real-time solution of mixed-integer convex programs by solving a surrogate low-dimensional convex problem instead. Other approaches learn a mapping to reduce the dimensionality of the decision variables in the surrogate problem (Wang et al., 2020). This is achieved by embedding the surrogate problem as an implicit layer of a neural network and differentiating its KKT optimality conditions (Amos and Kolter, 2017; Amos et al., 2018; Agrawal et al., 2019). In contrast, our method does not approximate any problem and, instead, we predict a warm-start of the algorithmic procedure with a focus on real-time computations. This allows us to clearly quantify the suboptimality achieved within a fixed number of real-time iterations.

3. End-to-end learning framework

Problem formulation. We consider the following parametric (convex) QP:

$$\begin{array}{ll} \text{minimize} & (1/2)x^T Px + c^T x \\ \text{subject to} & Ax + s = b \\ & s \geq 0, \end{array} \qquad \text{with parameter} \quad \theta = (\mathbf{vec}(P), \mathbf{vec}(A), c, b) \in \mathbf{R}^d, \tag{1}$$

and decision variables $x \in \mathbf{R}^n$ and $s \in \mathbf{R}^m$. We denote $\mathbf{S}_+^n$ and $\mathbf{S}_{++}^n$ to be the set of positive semidefinite and positive definite matrices respectively of size n. Here, P belongs to $\mathbf{S}_+^n$, the matrix A is in $\mathbf{R}^{m\times n}$, and b and c are vectors in $\mathbf{R}^m$ and $\mathbf{R}^n$ respectively. For a matrix Y, $\mathbf{vec}(Y)$ is the vector obtained by stacking the columns of Y. The dimension d of θ is upper bounded by $mn + n^2 + m + n$, but is smaller when only some of the data changes across the problems. The parameters of the QP, θ, in problem (1) are randomly drawn from a distribution $\mathcal{D}$ with compact support set Θ. We assume that all problems admit an optimal solution for any $\theta \in \Theta$. Our goal is to quickly solve these parametric QPs.

Optimality conditions. The KKT optimality conditions of problem (1), that is, primal feasibility, dual feasibility, and complementary slackness, are given by

$$Ax + s = b, \quad A^T y + Px + c = 0, \quad s \geq 0, \quad y \geq 0, \quad s \perp y = 0,$$

where $y \in \mathbf{R}^m$ is the dual variable to problem (1). We can compactly write these conditions as a linear complementarity problem (O'Donoghue, 2021, Sec. 3), *i.e.*, the problem of finding a $u = (x, y) \in \mathbf{R}^{m+n}$ such that

$$\mathcal{C} \ni u \perp Mu + q \in \mathcal{C}^*, \quad \text{where} \quad M = \begin{bmatrix} P & A^T \\ -A & 0 \end{bmatrix} \in \mathbf{R}^{(m+n)\times(m+n)},$$

and $q = (c, b) \in \mathbf{R}^{m+n}$. Here, $\mathcal{C} = \mathbf{R}^n \times \mathbf{R}_+^m$ and $\mathcal{C}^* = \{0\}^n \times \mathbf{R}_+^m$ is the dual cone to $\mathcal{C}$, *i.e.*, $\mathcal{C}^* = \{w \mid w^T u \geq 0,\ u \in \mathcal{C}\}$. This problem is equivalent to finding $u \in \mathbf{R}^{m+n}$ that satisfies the following inclusion (Bauschke and Combettes, 2011, Ex. 26.22)

$$0 \in Mu + q + N_{\mathcal{C}}(u), \tag{2}$$

where $N_{\mathcal{C}}(u)$ is the normal cone for cone $\mathcal{C}$ defined as $N_{\mathcal{C}}(z) = \{x \mid (y - u)^T x \leq 0, \forall y \in \mathcal{C}\}$ if $u \in \mathcal{C}$ and $\emptyset$ otherwise. Since $P \succeq 0$, $\mathcal{C}$ is a convex polyhedron, and problem (1) always admits an optimal solution, $Mu + q + N_{\mathcal{C}}(u)$ is maximal monotone (Ryu and Yin, 2022, Thm. 7, Thm. 11). Maximal monotonicity (see (Ryu and Yin, 2022, Sec. 2.2) for a definition) ensures convergence of Douglas-Rachford splitting, which we introduce next.

Douglas-Rachford splitting. We apply Douglas-Rachford (DR) splitting (Lions and Mercier, 1979; Douglas and Rachford, 1956) to solve problem (2). DR splitting consists of evaluating the *resolvent* of operators $Mu + q$ and $N_{\mathcal{C}}$, which for an operator F is defined as $(I + F)^{-1}$ (Ryu and Yin, 2022, pp 40). By noting that the resolvent of $Mu + q$ is $(M + I)^{-1}(z - q)$ and the resolvent of $N_{\mathcal{C}}(u)$ is $\Pi_{\mathcal{C}}(z)$, *i.e.*, the projection onto $\mathcal{C}$ (Ryu and Yin, 2022, Eq. 2.8, pp 42), we obtain Algorithm 1.

Algorithm 1 The DR Splitting algorithm for k iterations to solve problem (2).

```
Inputs: initial point $z^0$, problem data $(M, q)$, $k$ number of iterations
Output: approximate solution $z^k$
for $i = 0, \dots, k-1$ do
    $\tilde{u}^{i+1} = (M+I)^{-1}\left(z^i - q\right)$
    $u^{i+1} = \Pi_{\mathcal{C}}\left(2\tilde{u}^{i+1} - z^i\right)$
    $z^{i+1} = z^i + u^{i+1} - \tilde{u}^{i+1}$
end
```

The linear system in the first step is always solvable since $M + I$ has full rank (O'Donoghue, 2021), but it varies from problem to problem. The projection onto $\mathcal{C}$, however, is the same for all problems and simply clips negative values to zero and leaves non-negative values unchanged. For compactness, in the remainder of the paper, we write Algorithm 1 as

$$z^{i+1} = T_\theta\left(z^i\right) \quad \text{where} \quad T_\theta(z) = z + \Pi_{\mathcal{C}}\left(2(M+I)^{-1}(z-q) - z\right) - (M+I)^{-1}(z-q).$$

We make the dependence of T on θ explicit here as M and q are parametrized by θ. DR splitting is guaranteed to converge to a fixed point $z^\star \in \textbf{fix}\, T_\theta$ such that $T_\theta(z^\star) = z^\star$. Here, $\textbf{fix}\, T_\theta$ is defined as the set of fixed points of T_θ which is non-empty by the assumption that all problems have an optimal solution. Algorithm 1 returns an approximate solution z^k, from which we can recover an approximate primal-dual solution to (1) by computing $(x^k, y^k) = u^k = (M+I)^{-1}(z^k - q)$ and $s^k = b - Ax^k$.

Our end-to-end learning architecture. Our architecture consists of two modules as depicted in Figure 1. The first module is a NN with weights $\mathcal{W}$: it predicts a good-quality initial point (or warm-start), $\hat{z}_{\mathcal{W}}(\theta)$, to Algorithm 1 from the parameter θ of the QP in problem (1). The NN has L layers with ReLU activation functions (Ramachandran et al., 2017). We then write the warm-start prediction as follows:

$$\hat{z}_{\mathcal{W}}(\theta) = h_{\mathcal{W}}(\theta) = h_L\left(h_{L-1} \dots h_1(\theta)\right),$$

where $h_l(y_l) = (W_l y_l + b_l)_+$ for $l = 1, \dots, L-1$ and $h_L(y_L) = (W_L y_L + b_L)$. The weight matrices are $\{W_l\}_{l=1}^L$, and the bias terms are $\{b_l\}_{l=1}^L$. Here, $h_{\mathcal{W}}(\cdot)$ is a mapping from $\mathbf{R}^d$ to $\mathbf{R}^{m+n}$ corresponding to the prediction. We denote the set of all such mappings by $\mathcal{H}$. We emphasize the dependency of h on the weights and bias terms via the subscript $\mathcal{W} = (W_1, b_1, \dots, W_L, b_L)$. The second module corresponds to k iterations of DR splitting from the initial point $\hat{z}_{\mathcal{W}}(\theta)$. It outputs an approximate solution $z^k_{\mathcal{W}}(\theta) = T^k_\theta(\hat{z}_{\mathcal{W}}(\theta))$, from which we can recover an approximate solution to problem (1) as explained above. To obtain the solution to a QP given parameter θ, we perform a forward pass of the architecture, *i.e.*, compute $T^k_\theta(h_{\mathcal{W}}(\theta))$, with k chosen as needed.

Learning task. We define the loss function as the fixed-point residual of operator T_θ, *i.e.*,

$$\ell_\theta(z) = \|T_\theta(z) - z\|_2.$$

This loss measures the distance to convergence of Algorithm 1. The goal is to minimize the expected loss, which we define as the *risk*,

$$R^k(h_{\mathcal{W}}) = \mathbf{E}_{\theta \sim \mathcal{D}}\left[\ell_\theta\left(T^k_\theta(h_{\mathcal{W}}(\theta))\right)\right],$$

with respect to the weights $\mathcal{W}$ of the NN. In general, we cannot evaluate $R^k(h_{\mathcal{W}})$ exactly and, instead, we minimize the *empirical risk*,

$$\hat{R}^k(h_{\mathcal{W}}) = \frac{1}{N}\sum_{i=1}^{N} \ell_{\theta_i}\left(T^k_{\theta_i}(h_{\mathcal{W}}(\theta_i))\right). \tag{3}$$

Here, N is the number of training problems. We use gradient methods on mini-batch or stochastic approximations of the empirical risk during the training process (Sra et al., 2011).

Differentiability of our architecture. To see that we can differentiate ℓ_θ with respect to $\mathcal{W}$, note that the second module consists of repeated linear system solves and projections onto $\mathcal{C}$ (see Algorithm 1). Since the linear systems always have unique solutions, $\tilde{u}^{i+1}$ is linear in z^i and the linear system solves are differentiable. Furthermore, as the projection step involves clipping non-negative values to zero, it is differentiable everywhere except at zero. In the first module, the NN consists entirely of differentiable functions except for the ReLU activation function, which is likewise differentiable everywhere except at zero.

4. Generalization bounds

In this section, we provide an upper bound on the expected loss $R^k(h_{\mathcal{W}})$ of our framework for any $h_{\mathcal{W}} \in \mathcal{H}$ if the operator, T_θ, is contractive. This bound involves the empirical expected loss $\hat{R}^k(h_{\mathcal{W}})$, the Rademacher complexity of the NN appearing in the first module *only* and a term which decreases with both the number of iterations of DR splitting, k, and the number of training samples, N. To obtain this bound, we rely on the key property of contractive operators,

$$\|T_\theta(z) - T_\theta(w)\|_2 \leq \beta_\theta \|z - w\|_2,$$

where $\beta_\theta \in (0,1)$ is the contractive factor of operator T_θ. In the next theorem we use the notion of multivariate empirical Rademacher complexity as defined, *e.g.*, in (Bertsimas and Kallus, 2014, Def. 3), of the NN. We denote this complexity measure as $\mathbf{erad}(\mathcal{H})$.

Theorem 1 *Assume that each T_θ is β_θ-contractive and let $\beta = \max_{\theta\in\Theta}\beta_\theta \in (0,1)$. Let $\mathcal{H}$ be the set of L-layer ReLU neural networks such that for any $h_{\mathcal{W}} \in \mathcal{H}$ and $\theta \in \Theta$, $\mathbf{dist}_{\mathbf{fix}\,T_\theta}(h_{\mathcal{W}}(\theta)) \leq B$ for some $B > 0$. Then, with probability at least $1-\delta$ over the draw of i.i.d samples,*

$$R^k(h_{\mathcal{W}}) \leq \hat{R}^k(h_{\mathcal{W}}) + 2\beta^k\left(2\sqrt{2}\mathbf{erad}(\mathcal{H}) + 3B\frac{\log(2/\delta)}{2N}\right) \quad \forall h_{\mathcal{W}} \in \mathcal{H},$$

where k is the number of DR iterations and N is the number of training samples.

When the Rademacher complexity of $\mathcal{H}$ can be upper bounded, such as in the case of 1 or 2-layer NNs with bounded norms on the weights, the generalization error and its dependence on k and N can be made even more explicit (Golowich et al., 2018; Neyshabur et al., 2019).

Corollary 2 *Let $\mathcal{H}$ be the set of linear functions with bounded norm, i.e., $\mathcal{H} = \{h \mid h(\theta) = W\theta\}$ where $\theta \in \mathbf{R}^d$, $W \in \mathbf{R}^{(m+n)\times d}$ and $(1/2)||W||_F^2 \leq D$ for some $D > 0$. Then, with probability at least $1-\delta$ over the draw of i.i.d samples,*

$$R^k(h_{\mathcal{W}}) \leq \hat{R}^k(h_{\mathcal{W}}) + 2\beta^k\left(4\rho_2(\theta)D\sqrt{\frac{m+n}{N}} + 3B\frac{\log(2/\delta)}{2N}\right) \quad \forall h_{\mathcal{W}} \in \mathcal{H},$$

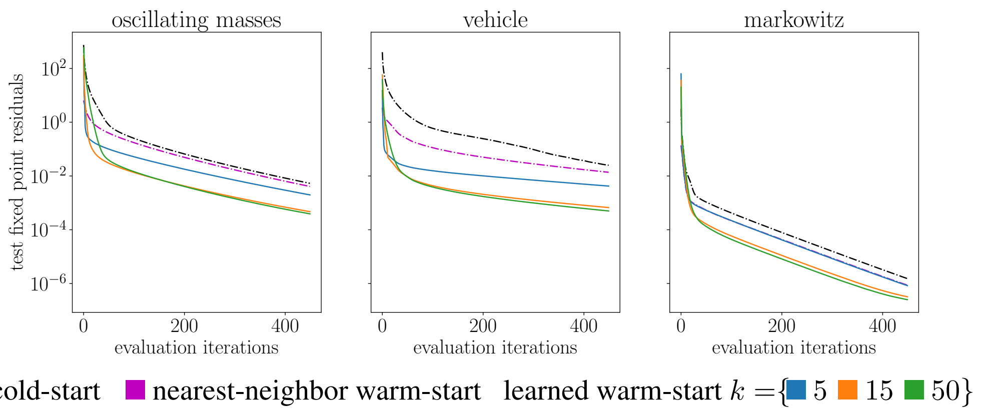

Figure 2: We plot the test fixed point residuals for different warm-starts of DR splitting. We train our architecture with $k = 5, 15$, and 50 DR iterations with loss function (3). We compare our results against a cold-start (black) and the nearest-neighbor warm-start (magenta). Left: oscillating masses example. Middle: vehicle dynamics example. Right: portfolio optimization example.

where k, B, and N are defined as in Thm. 1, and $\rho_2(\theta) = \max_{\theta \in \Theta} ||\theta||_2$ (El Balghiti et al., 2019, Thm. 6).

5. Numerical experiments

We now illustrate our method with examples of quadratic optimization problems deployed and repeatedly solved in control and portfolio optimization settings where rapid solutions are important for real-time execution and backtesting. Our architecture was implemented in the JAX library (Bradbury et al., 2018) with the Adam (Kingma and Ba, 2015) training optimizer. All computations were run on the Princeton HPC Della Cluster, and all examples could be trained in under 2 hours. We use 10000 training problems and evaluate on 2000 test problems. In our examples we use a NN with three hidden layers of size 500 each. We compare our learned warm-start against two other initialization approaches. The first initializes DR-splitting with a random point which we call a *cold-start*. The other is a *nearest-neighbor warm-start* which initializes the test problem with the optimal solution of the nearest training problem measured by distance in terms of its parameter θ. The problems are sufficiently distant from each other that warm-starting with the nearest-neighbor approach does not yield satisfactory results. Our code is available at `https://github.com/stellatogrp/l2ws`.

5.1. Oscillating masses

We consider the problem of controlling a physical system that involves connected springs and masses (Wang and Boyd, 2010; Chen et al., 2022, System 4),

$$\begin{array}{ll} \text{minimize} & x_T^T Q_T x_T + \sum_{t=1}^{T-1} x_t^T Q x_t + u_t^T R u_t \\ \text{subject to} & x_{t+1} = A x_t + B u_t \quad t = 0, \dots, T-1 \\ & u_{\min} \le u_t \le u_{\max} \quad t = 0, \dots, T-1 \\ & x_{\min} \le x_t \le x_{\max} \quad t = 1, \dots, T \\ & x_0 = x_{\text{init}}, \end{array}$$

Table 1: Oscillating masses problem. We compare the number of iterations of DR splitting required to reach different levels of accuracy with different warm-starts (learned warm-start with $k = 5, 15, 50$, cold-start, and a nearest-neighbor warm-start). The reduction columns are the iterations reduced as a fraction of the cold-start iterations.

	cold-start	nearest-neighbor		train $k = 5$		train $k = 15$		train $k = 50$	
ϵ	iters	iters	reduction	iters	reduction	iters	reduction	iters	reduction
0.01	381	353	0.07	279	0.27	176	0.54	127	0.67
0.001	651	616	0.05	555	0.15	438	0.33	338	0.48
0.0001	1019	973	0.05	932	0.09	816	0.20	663	0.35

where the states $x_t \in \mathbf{R}^{n_x}$ and the inputs $u_t \in \mathbf{R}^{n_u}$ are subject to lower and upper bounds. Matrices $A \in \mathbf{R}^{n_x \times n_x}$ and $B \in \mathbf{R}^{n_x \times n_u}$ define the system dynamics where n_x and n_u are the number of states and controls. The horizon length is T and the parameter θ is the initial state, x_{init}. Matrices $Q \in \mathbf{S}_+^{n_x}$ and $R \in \mathbf{S}_{++}^{n_u}$ define the state and input costs at each stage, and $Q_T \in \mathbf{S}_+^{n_x}$ the final stage cost.

Numerical example. The problem takes values $n_x = 36$, $n_u = 9$, and $T = 50$. Matrices A and B are obtained by discretizing the state dynamics with time step 0.5 (Chen et al., 2022, System 4). We set $u_{\text{min}} = -u_{\text{max}} = 1/2$ and $x_{\text{min}} = -x_{\text{max}} = 2$. We set $Q_t = I$ and $R_t = I$ for all t. We sample $\theta = x_{\text{init}}$ uniformly in $[-2, 2]^{36}$. Figure 2 and Table 1 show the convergence behavior of our method.

5.2. Vehicle dynamics control problem

We consider problem of controlling a vehicle, modeled as a parameter-varying linear dynamical system (Takano et al., 2003), to track a reference trajectory (Zhang et al., 2019). We formulate it as the following QP:

$$\begin{array}{ll}
\text{minimize} & (y_T - y_T^{\text{ref}})^T Q (y_T - y_T^{\text{ref}}) + \sum_{t=1}^{T-1} (y_t - y_t^{\text{ref}})^T Q_T (y_t - y_t^{\text{ref}}) + u_t^T R u_t \\
\text{subject to} & x_{t+1} = A(v) x_t + B(v) u_t + E(v) \delta_t \quad t = 0, \dots, T-1 \\
& |u_t| \leq \bar{u}, \quad |u_t - u_{t-1}| \leq \overline{\Delta u}, \quad t = 0, \dots, T-1 \\
& y_t = C x_t \quad t = 0, \dots, T-1 \\
& x_0 = x_{\text{init}},
\end{array}$$

where $x_t \in \mathbf{R}^4$ is the state and $u_t \in \mathbf{R}^3$ is the input, and $\delta_t \in \mathbf{R}$ is the driver steering input, which is linear over time. We aim to minimize the distance between the output, $y_t \in \mathbf{R}^3$ and the reference trajectory $y_t^{\text{ref}} \in \mathbf{R}^3$ over time. Matrices $Q \in \mathbf{S}_+^{n_x}$ and $Q_T \in \mathbf{S}_+^{n_x}$ define the state costs, $R \in \mathbf{S}_{++}^{n_u}$ the input cost, and $C \in \mathbf{R}^{3 \times 4}$ the output y_t. The term $v \in \mathbf{R}$ is the longitudinal velocity of the vehicle that parametrizes $A \in \mathbf{R}^{4 \times 4}, B \in \mathbf{R}^{4 \times 3}$, and $E \in \mathbf{R}^4$. Vectors $\bar{u}$ and $\overline{\Delta u}$ bound the magnitude of the inputs and change in inputs respectively. The parameters θ for the problem are the initial state x_{init}, the initial velocity v, the previous control input u_{-1}, the reference signals $\{y_t^{\text{ref}}\}_{t=0}^T$, and the steering inputs $\{\delta_t\}_{t=0}^{T-1}$.

Numerical example. The time horizon is $T = 30$. Matrices A, B, E result from discretizing the dynamics (Takano et al., 2003). We sample all parameters uniformly from their bounds: the velocity

Table 2: Vehicle problem. We compare the number of iterations of DR splitting required to reach different levels of accuracy with different warm-starts (learned warm-start with $k = 5, 15, 50$, no warm-start, and a nearest-neighbor warm-start). The reduction columns are the iterations reduced as a fraction of the cold-start iterations.

	cold-start	nearest-neighbor		train $k = 5$		train $k = 15$		train $k = 50$	
ϵ	iters	iters	reduction	iters	reduction	iters	reduction	iters	reduction
0.01	639	520	0.19	203	0.68	48	0.92	48	0.92
0.001	1348	1163	0.14	895	0.34	351	0.74	299	0.78
0.0001	2126	1948	0.08	1653	0.22	1006	0.53	882	0.59

Table 3: Markowitz problem. We compare the number of iterations of DR splitting required to reach different levels of accuracy with different warm-starts (learned warm-start with $k = 5, 15, 50$, no warm-start, and a nearest-neighbor warm-start). The reduction columns are the iterations reduced as a fraction of the cold-start iterations.

	cold-start	nearest-neighbor		train $k = 5$		train $k = 15$		train $k = 50$	
ϵ	iters	iters	reduction	iters	reduction	iters	reduction	iters	reduction
0.01	14	7	0.5	7	0.5	9	0.36	11	0.21
0.001	54	24	0.56	22	0.59	16	0.7	19	0.65
0.0001	186	148	0.2	147	0.21	72	0.61	61	0.67

$v \in [2, 35]$, the output $y_t \in [-\bar{y}, \bar{y}]$ where $\bar{y} = (25, 40, 30)$ in degrees, and the previous control $u_{-1} \in [-\bar{u}, \bar{u}]$ where $\bar{u} = 10^3(1, 20, 30)$. We sample the initial steering angle from $[-45, 45]$ and its linear increments from $[-30, 30]$. Figure 2 and Table 2 show the performance of our method.

5.3. Portfolio optimization

We consider the portfolio optimization problem where we want to allocate assets to maximize the risk-adjusted return (Markowitz, 1952; Boyd et al., 2017),

$$\begin{array}{ll} \text{maximize} & \rho\mu^T x - x^T \Sigma x \\ \text{subject to} & \mathbf{1}^T x = 1, \quad x \geq 0, \end{array}$$

where $x \in \mathbf{R}^n$ represents the portfolio, $\mu \in \mathbf{R}^n$ the expected returns, $1/\rho > 0$ the risk-aversion parameter, and $\Sigma \in \mathbf{S}^n_+$ the return covariance. For this problem, $\theta = \mu$.

Numerical example. We use real-world stock return data from 3000 popular assets from 2015-2019 (Nasdaq, 2022). We use an l-factor model for the risk and set $\Sigma = F\Sigma_F F^T + D$ where $F \in \mathbf{R}^{n,l}$ is the factor-loading matrix, $\Sigma_F \in \mathbf{S}^l_+$ estimates the factor returns, and $D \in \mathbf{S}^n_{++}$ is a diagonal matrix accounting for additional variance for each asset also called the idiosyncratic risk. We compute the factor model with 15 factors by using the same approach as in (Boyd et al., 2017). The return parameters are $\mu = \alpha(\hat{\mu}_t + \epsilon_t)$ where $\hat{\mu}_t$ is the realized return at time t, $\epsilon_t \sim \mathcal{N}(0, \sigma_\epsilon I)$, and $\alpha = 0.24$ is selected to minimize the mean squared error $\mathbf{E}\|\mu_t - \hat{\mu}_t\|_2^2$ (Boyd et al., 2017). We iterate and repeatedly cycle over the five year period to sample a μ vector for each of our problems. Figure 2 and Table 3 show the performance of our method.

Appendix A. Proof of the generalization bound

A.1. Proof of Theorem 1

Assume that the loss is bounded by C, *i.e.*, $\ell_\theta(T_\theta^k(h_\mathcal{W}(\theta))) \leq C \quad \forall h_\mathcal{W} \in \mathcal{H}, \theta \in \Theta$. Then, for any $\delta > 0$, with probability at least $1-\delta$ over the draw of an i.i.d. sample of size N, the following holds (Bartlett and Mendelson, 2002; Mohri et al., 2012, Thm. 3.3):

$$R^k(h_\mathcal{W}) \leq \hat{R}^k(h_\mathcal{W}) + \frac{2}{N}\mathbf{E}\left[\sup_{h_\mathcal{W}\in\mathcal{H}} \sum_{i=1}^N \sigma_i \ell_{\theta_i}(T_{\theta_i}^k(h_\mathcal{W}(\theta_i)))\right] + 3C\sqrt{\log(2/\delta)/(2N)}, \tag{4}$$

where the σ_i's are i.i.d. Rademacher random variables. We need to bound the worst-case loss C and the expectation term in equation (4). We first prove that $\ell_\theta(T_\theta^k(\cdot))$ is $2\beta^k$-Lipschitz:

$$\begin{aligned}
|\ell_\theta(T_\theta^k(z)) - \ell_\theta(T_\theta^k(w))| &= |\|T_\theta^{k+1}(z) - T_\theta^k(z)\|_2 - \|T_\theta^{k+1}(w) - T_\theta^k(w)\|_2| \\
&\leq \|T_\theta^{k+1}(z) - T_\theta^{k+1}(w) + T_\theta^k(w) - T_\theta^k(z)\|_2 \\
&\leq \|T_\theta^{k+1}(z) - T_\theta^{k+1}(w)\|_2 + \|T_\theta^k(w) - T_\theta^k(z)\|_2 \\
&\leq 2\beta^k\|z-w\|_2.
\end{aligned}$$

The first and second inequalities follow from the reverse triangle inequality and triangle property respectively. In the last line, we use the contractive property of T_θ. Using this Lipschitz-property, we can provide a worst-case bound of $2\beta^k B$ as follows:

$$\begin{aligned}
\ell_\theta(T_\theta^k(h_\mathcal{W}(\theta))) &= |\ell_\theta(T_\theta^k(h_\mathcal{W}(\theta))) - \ell_\theta(T_\theta^k(z^\star(\theta)))| \\
&\leq 2\beta^k \mathbf{dist}_{\mathbf{fix}\, T_\theta}(h_\mathcal{W}(\theta)) \\
&\leq 2\beta^k B.
\end{aligned}$$

In the first line, we denote the fixed point of the problem parametrized by θ as $z^\star(\theta)$. Since the operators are contractive, there is only one such fixed point. The equality follows from the non-negativity of the the loss function and the fact that $z^\star(\theta)$ is a fixed point. In the second line, we use the Lipschitz property proven before. The last line follows from the assumption of the theorem.

Next, we bound the expectation term in (4) as follows:

$$\begin{aligned}
\mathbf{E}\left[\sup_{h_\mathcal{W}\in\mathcal{H}} \sum_{i=1}^N \sigma_i \ell_{\theta_i}(T_{\theta_i}^k(h_\mathcal{W}(\theta_i)))\right] &\leq 2\sqrt{2}\beta^k\, \mathbf{E}\left[\sup_{h_\mathcal{W}\in\mathcal{H}} \sum_{i=1}^N \sum_{j=1}^{m+n} \sigma_{i,j}(h_\mathcal{W}(\theta_i))_j\right] \\
&= 2\sqrt{2}\beta^k \mathbf{erad}(\mathcal{H}).
\end{aligned}$$

In the first line, $\sigma_{i,j}$ are doubly-indexed i.i.d. Rademacher random variables. The inequality comes from using the Lipschitz factor of $2\beta^k$ and the vector concentration inequality from (Maurer, 2016, Cor. 4). The second line comes from the definition of the multivariate empirical Rademacher complexity of the function class $\mathcal{H}$ completing the proof.

Acknowledgments

The author(s) are pleased to acknowledge that the work reported on in this paper was substantially performed using the Princeton Research Computing resources at Princeton University which is consortium of groups led by the Princeton Institute for Computational Science and Engineering (PICSciE) and Office of Information Technology's Research Computing.

References

Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa Moursi. Differentiating through a cone program. *Journal of Applied and Numerical Optimization*, 1(2):107–115, 2019.

Brandon Amos. Tutorial on amortized optimization for learning to optimize over continuous domains, 2022. URL https://arxiv.org/abs/2202.00665.

Brandon Amos and Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/amos17a.html.

Brandon Amos, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable mpc for end-to-end planning and control. *arXiv preprint arXiv:1810.13400*, 2018.

Marcin Andrychowicz, Misha Denil, Sergio Gómez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 3988–3996, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Kyri Baker. Learning warm-start points for ac optimal power flow. In *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, 2019. doi: 10.1109/MLSP.2019.8918690.

Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *J. Mach. Learn. Res.*, 3:463–482, 2002. URL http://dblp.uni-trier.de/db/journals/jmlr/jmlr3.html#BartlettM02.

Nicola Bastianello, Andrea Simonetto, and Emiliano Dall'Anese. Opreg-boost: Learning to accelerate online algorithms with operator regression, 2021. URL https://arxiv.org/abs/2105.13271.

Heinz. H. Bauschke and Patrick. L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer, 1st edition, 2011.

Amir Beck. *First-Order Methods in Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2017. doi: 10.1137/1.9781611974997. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611974997.

Dimitriis Bertsimas and Bartolomeo Stellato. Online mixed-integer optimization in milliseconds. *arXiv e-prints*, July 2019. URL https://arxiv.org/abs/1907.02206.

Dimitris Bertsimas and Nathan Kallus. From predictive to prescriptive analytics. *Manag. Sci.*, 66: 1025–1044, 2014.

Dimitris Bertsimas and Bartolomeo Stellato. The voice of optimization. *Machine Learning*, 110: 249–277, Feb 2021. URL https://doi.org/10.1007/s10994-020-05893-5.

Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017. doi: 10.1017/9781139061759.

Stephen Boyd, Enzo Busseti, Steven Diamond, Ronald N. Kahn, Kwangmoo Koh, Peter Nystrup, and Jan Speth. Multi-period trading via convex optimization, 2017. URL https://arxiv.org/abs/1705.00109.

Stephen P. Boyd, Neal Parikh, Eric King wah Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3:1–122, 2011.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Steven Chen, Kelsey Saulnier, Nikolay Atanasov, Daniel D. Lee, Vijay Kumar, George J. Pappas, and Manfred Morari. Approximating explicit model predictive control using constrained neural networks. In *2018 Annual American Control Conference (ACC)*, pages 1520–1527, 2018. doi: 10.23919/ACC.2018.8431275.

Steven W. Chen, Tianyu Wang, Nikolay Atanasov, Vijay Kumar, and Manfred Morari. Large scale model predictive control with neural networks and primal active sets. *Automatica*, 135:109947, 2022. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2021.109947. URL https://www.sciencedirect.com/science/article/pii/S0005109821004738.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark, 2021. URL https://arxiv.org/abs/2103.12828.

Jim Douglas and H. H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American Mathematical Society*, 82(2):421–439, 1956. ISSN 00029947. URL http://www.jstor.org/stable/1993056.

Othman El Balghiti, Adam N. Elmachtoub, Paul Grigas, and Ambuj Tewari. Generalization bounds in the predict-then-optimize framework. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/a70145bf8b173e4496b554ce57969e24-Paper.pdf.

Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpoases: a parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6:327–363, 2014.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/finn17a.html.

Noah Golowich, Alexander Rakhlin, and Ohad Shamir. Size-independent sample complexity of neural networks. In Sébastien Bubeck, Vianney Perchet, and Philippe Rigollet, editors, *Proceedings of the 31st Conference On Learning Theory*, volume 75 of *Proceedings of Machine Learning Research*, pages 297–299. PMLR, 06–09 Jul 2018.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 399–406, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Jeffrey Ichnowski, Paras Jain, Bartolomeo Stellato, Goran Banjac, Michael Luo, Francesco Borrelli, Joseph. E Gonzales, Ion Stoica, and Ken Goldberg. Accelerating quadratic optimization with reinforcement learning. In *Advances in Neural Information Processing Systems 35*, 12 2021. URL https://arxiv.org/pdf/2107.10847.pdf.

Haewon Jung, Junyoung Park, and Jinkyoo Park. Learning context-aware adaptive solvers to accelerate quadratic programming, 2022. URL https://arxiv.org/abs/2211.12443.

Benjamin Karg and Sergio Lucia. Efficient representation and approximation of model predictive control laws via deep learning. *IEEE Transactions on Cybernetics*, PP:1–13, 06 2020. doi: 10.1109/TCYB.2020.2999556.

Yoon Kim, Sam Wiseman, Andrew Miller, David Sontag, and Alexander Rush. Semi-amortized variational autoencoders. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2678–2687. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/kim18e.html.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Scott Kuindersma, Frank Permenter, and Russ Tedrake. An efficiently solvable quadratic program for stabilizing dynamic locomotion. 06 2014. doi: 10.1109/ICRA.2014.6907230.

Ke Li and Jitendra Malik. Learning to optimize, 2016. URL https://arxiv.org/abs/1606.01885.

P. L. Lions and B. Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, 1979. doi: 10.1137/0716071. URL https://doi.org/10.1137/0716071.

Jialin Liu, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. ALISTA: Analytic weights are as good as learned weights in LISTA. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1lnzn0ctQ.

Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/2975974.

Andreas Maurer. A vector-contraction inequality for rademacher complexities. In Ronald Ortner, Hans Ulrich Simon, and Sandra Zilles, editors, *Algorithmic Learning Theory*, pages 3–17, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46379-7.

Luke Metz, James Harrison, C. Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, and Jascha Sohl-Dickstein. Velo: Training versatile learned optimizers by scaling up, 2022. URL https://arxiv.org/abs/2211.09760.

Sidhant Misra, Line Roald, and Yeesian Ng. Learning for constrained optimization: Identifying optimal active constraint sets. *INFORMS J. on Computing*, 34(1):463–480, jan 2022. ISSN 1526-5528. doi: 10.1287/ijoc.2020.1037. URL https://doi.org/10.1287/ijoc.2020.1037.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet S. Talwalkar. Foundations of machine learning. In *Adaptive computation and machine learning*, 2012.

Nasdaq. End-of-day us stock prices. https://data.nasdaq.com/databases/EOD/documentation, 2022. This data was obtained and used solely by Princeton University.

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BygfghAcYX.

Brendan O'Donoghue. Operator splitting for a homogeneous embedding of the linear complementarity problem. *SIAM Journal on Optimization*, 31(3):1999–2023, 2021. doi: 10.1137/20M1366307. URL https://doi.org/10.1137/20M1366307.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Swish: a self-gated activation function. *arXiv: Neural and Evolutionary Computing*, 2017.

Ernest K. Ryu and Wotao Yin. *Large-Scale Convex Optimization: Algorithms amp; Analyses via Monotone Operators*. Cambridge University Press, 2022.

Suvrit Sra, Sebastian Nowozin, and Stephen J. Wright. *Optimization for Machine Learning*. The MIT Press, 2011. ISBN 026201646X.

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Boyd Stephen. OSQP: An Operator Splitting Solver for Quadratic Programs. *Mathematical Programming Computation*, 12(4):637–672, 10 2020. URL https://doi.org/10.1007/s12532-020-00179-2.

Shuichi Takano, Masao Nagai, Tetsuo Taniguchi, and Tadashi Hatano. Study on a vehicle dynamics model for improving roll stability. *JSAE Review*, 24(2):149–156, 2003. ISSN 0389-4304. doi: https://doi.org/10.1016/S0389-4304(03)00012-2. URL https://www.sciencedirect.com/science/article/pii/S0389430403000122.

Shobha Venkataraman and Brandon Amos. Neural fixed-point acceleration for convex optimization. *CoRR*, abs/2107.10254, 2021. URL https://arxiv.org/abs/2107.10254.

Kai Wang, Bryan Wilder, Andrew Perrault, and Milind Tambe. Automatically learning compact quality-aware surrogates for optimization problems. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Yang Wang and Stephen Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278, 2010. doi: 10.1109/TCST.2009.2017934.

Xiaojing Zhang, Monimoy Bujarbaruah, and Francesco Borrelli. Safe and near-optimal policy learning for model predictive control using primal-dual neural networks. In *2019 American Control Conference (ACC)*, pages 354–359, 2019. doi: 10.23919/ACC.2019.8814335.