




# Efficient differentiable quadratic programming layers: an ADMM approach

Andrew Butler<sup>1</sup> · Roy H. Kwon<sup>1</sup> 

Received: 25 December 2021 / Accepted: 10 October 2022 / Published online: 25 October 2022  
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

## Abstract

Recent advances in neural-network architecture allow for seamless integration of convex optimization problems as differentiable layers in an end-to-end trainable neural network. Integrating medium and large scale quadratic programs into a deep neural network architecture, however, is challenging as solving quadratic programs exactly by interior-point methods has worst-case cubic complexity in the number of variables. In this paper, we present an alternative network layer architecture based on the alternating direction method of multipliers (ADMM) that is capable of scaling to moderate sized problems with 100–1000 decision variables and thousands of training examples. Backward differentiation is performed by implicit differentiation of a customized fixed-point iteration. Simulated results demonstrate the computational advantage of the ADMM layer, which for medium scale problems is approximately an order of magnitude faster than the state-of-the-art layers. Furthermore, our novel backward-pass routine is computationally efficient in comparison to the standard approach based on unrolled differentiation or implicit differentiation of the KKT optimality conditions. We conclude with examples from portfolio optimization in the integrated prediction and optimization paradigm.

**Keywords** Data driven stochastic-programming · Differentiable neural networks · Quadratic programming · ADMM

## 1 Introduction

Many problems in engineering, statistics and machine learning require solving convex optimization programs. In many real-world applications, the solution to the convex optimization program is a single component in a larger decision-making process. Recent advances in neural-network architecture [1–3] allow for seamless

---

✉ Roy H. Kwon  
rkwon@mie.utoronto.ca

<sup>1</sup> Department of Mechanical and Industrial Engineering, University of Toronto, 5 King's College Rd, Toronto, On M5S 3G8, Canada

integration of convex optimization programs as differentiable layers in an end-to-end trainable neural network. Differentiable convex optimization layers have therefore been fundamental to the growing body of research on fully integrated predictive decision-making (see for example [4, 11, 12, 17, 25, 26, 36]).

In this paper, we consider convex programming layers that take the form of parametric quadratic programs (PQPs) with linear equality and box inequality constraints:

$$\begin{aligned} & \underset{\mathbf{z}}{\text{minimize}} && \frac{1}{2} \mathbf{z}^T \mathbf{Q}(\boldsymbol{\theta}) \mathbf{z} + \mathbf{z}^T \mathbf{p}(\boldsymbol{\theta}) \\ & \text{subject to} && \mathbf{A}(\boldsymbol{\theta}) \mathbf{z} = \mathbf{b}(\boldsymbol{\theta}), \quad \mathbf{l}(\boldsymbol{\theta}) \leq \mathbf{z} \leq \mathbf{u}(\boldsymbol{\theta}) \end{aligned} \quad (1)$$

Here  $\mathbf{z} \in \mathbb{R}^{d_z}$  denotes the decision variable and  $\mathbf{Q}(\boldsymbol{\theta})$ ,  $\mathbf{p}(\boldsymbol{\theta})$ ,  $\mathbf{A}(\boldsymbol{\theta})$ ,  $\mathbf{b}(\boldsymbol{\theta})$ ,  $\mathbf{l}(\boldsymbol{\theta})$ ,  $\mathbf{u}(\boldsymbol{\theta})$  are the parameterized problem variables. Program (1) occurs in many applications to statistics [34, 35], machine-learning [22, 24], signal-processing [23] and finance [6, 27, 28].

In general, a differentiable convex optimization layer can be viewed as a function that maps the program input variables to optimal primal(-dual) solution(s). Optimizing problem variables by backpropagation therefore requires computing the action of the Jacobian of the optimal solution(s) with respect to the corresponding program input variables. For example, the OptNet layer, proposed by Amos and Kolter [3] is a specialized differentiable optimization layer that uses a primal-dual interior-point method for small scale batch quadratic programs. While the authors note that the OptNet layer is computationally efficient and therefore practical for small scale problems ( $d_z < 100$ ), solving convex quadratic programs exactly by interior-point methods has worst-case time complexity on the order of  $\mathcal{O}(d_z^3)$  [21]. Therefore, for medium ( $100 < d_z < 1000$ ) and large ( $d_z > 1000$ ) scale problems, embedding an OptNet layer in a neural network can be computationally intractable.

In this paper, we address the computational challenges in the medium to large scale limit and propose an alternative differentiable network architecture for batch constrained quadratic programs of the form of Program (1). Our differentiable quadratic programming layer is built on top of the alternating direction method of multipliers (ADMM) algorithm, which recently has become increasingly popular for solving large scale convex optimization problems [9, 29, 31–33]. Existing ADMM layer architectures [14, 38, 39] perform backpropagation by unrolling the ADMM computational graph, which requires substantially larger networks and can be computationally demanding. Alternatively, many differentiable convex optimization layers perform backpropagation by implicit differentiation of the Karush-Kuhn-Tucker (KKT) optimality conditions. However, KKT implicit differentiation requires solving a system of equations of dimension  $\mathbb{R}^{3d_z \times 3d_z}$ , which can also be computationally impractical. As an alternative, we present a novel and efficient backward-pass routine that implicitly differentiates a customized fixed-point mapping. A primary advantage is that the fixed-point iteration is of dimension  $d_z$  and therefore the resulting system of equations is approximately 3 times smaller than the equivalent KKT system. Finally, our differentiable ADMM layer and all algorithmic implementations is made available as an open-source R package, available here: <https://github.com/butl3ra/lqp>.

The remainder of the paper is outlined as follows. We begin with a brief discussion of related work in the field of differentiable convex optimization layers. In Sect. 2 we review the ADMM algorithm and present our ADMM-based architecture for forward solving batch PQPs. We review the KKT based implicit differentiation and then present the fixed-point iteration and derive the expression for the relevant gradients. In Sect. 3 we compare the computational efficiency of our ADMM layer with the OptNet layer and a splitting cone solver (SCS) layer based on the *cvxpy-layers* implementation [1, 29]. We demonstrate that for medium scale problems, our ADMM layer implementation is approximately an order of magnitude faster than both the OptNet and SCS layers. Moreover, we compare the computational efficiency of the backpropagation routines based on unrolled differentiation, KKT implicit differentiation and fixed-point implicit differentiation. We demonstrate that the fixed-point implicit differentiation is universally more efficient than the KKT implicit differentiation, and under certain conditions is preferred to the unrolled differentiation. We conclude with a real world application of the ADMM layer to a medium scale portfolio optimization problem in the integrated prediction and optimization paradigm.

## 1.1 Related work

Our ADMM layer is motivated by the OptNet layer [3], which implements a primal-dual interior-point method for solving small scale batch constrained quadratic programs and performs backpropagation by implicit differentiation of the KKT optimality conditions. For reasons mentioned earlier, the author's acknowledge that the OptNet layer may be impractical for optimization problems with a moderate number of variables. Alternatively, Agrawal et al. [1] provide a domain-specific language for differentiable convex programming. The forward-pass applies operator splitting [29] to the homogeneous self-dual embedding and backpropagation is performed by implicit differentiation of the corresponding residual map [2]. More recently, Blondel et al. [7] provide an efficient and modular approach for implicit differentiation of optimization problems. They consider KKT, proximal gradient and mirror descent fixed-point implicit differentiation and provide a software infrastructure for efficiently integrating their modular implicit differentiation routines with state-of-the-art optimization solvers. That said, for solving batches of convex optimization problems it is often preferred and more efficient to avail of optimization solvers that have the ability to exploit fast GPU-based batch solves.

Perhaps most closely related to our work, the ADMM-Net and its generalizations (D-LADMM) [38, 39], directly embeds the iterative ADMM procedure as a fully learnable network graph. However, the ADMM-Net implementation does not support inequality constraints and moreover performs argmin differentiation by unrolling the 'inner-loop' of the ADMM routine, which necessitates substantially larger and less efficient networks [3, 4].

To our knowledge, our ADMM layer implementation is the first of its kind that can efficiently handle medium to large scale constrained PQPs. The forward pass solves batch PQPs by applying the ADMM algorithm in batch form. Backpropagation is

performed by implicit differentiation of a customized fixed-point mapping. A primary advantage of the fixed-point differentiation is that the fixed-point scheme is of dimension  $d_z$  and is therefore approximately 3 times smaller than the corresponding KKT system. In the absence of a pre-factorized KKT system, the fixed-point implicit differentiation is preferred to KKT implicit differentiation and is competitive with the efficient KKT factorization caching provided in the OptNet layer. While unrolled differentiation of the ADMM algorithm is appropriate for small scale problems, for larger scale problems or for problems that require solving the convex optimization problem to a high degree of accuracy, an unrolled differentiation approach can also be computationally demanding. In contrast, the fixed-point implicit differentiation method is shown to be computationally efficient and invariant to the number of ‘inner’ iterations performed in the ADMM forward-pass. Lastly, our implementation is not without its own limitations and areas for improvement, discussed in detail in Sect. 4.

## 2 Methodology

The alternating direction method of multipliers (ADMM) algorithm, first proposed by Gabay and Mercier [19] and Glowinski and Marroco [20], is well suited to many large-scale and distributed problems common to applications of statistics, machine learning, control and finance. We note that the ADMM algorithm is closely related to algorithms such as dual ascent, the augmented Lagrangian method of multipliers, and operator (Douglas-Rachford) splitting and refer to Boyd et al. [9] for a comprehensive overview.

In general, the ADMM algorithm is applied to problems of the form:

$$\begin{aligned} & \text{minimize} && f(\mathbf{x}) + g(\mathbf{z}) \\ & \text{subject to} && \mathbf{Ax} + \mathbf{Bz} = \mathbf{c} \end{aligned} \quad (2)$$

with decision variables  $\mathbf{x} \in \mathbb{R}^{d_x}$ ,  $\mathbf{z} \in \mathbb{R}^{d_z}$  and problem variables  $\mathbf{A} \in \mathbb{R}^{d_{eq} \times d_x}$ ,  $\mathbf{B} \in \mathbb{R}^{d_{eq} \times d_z}$  and  $\mathbf{c} \in \mathbb{R}^{d_{eq}}$ . In order to guarantee convergence we assume that  $f : \mathbb{R}^{d_x} \rightarrow \mathbb{R}$  and  $g : \mathbb{R}^{d_z} \rightarrow \mathbb{R}$  are closed, proper convex functions [9]. The augmented Lagrangian of Program (2) is given by:

$$L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{y}) = f(\mathbf{x}) + g(\mathbf{z}) + \lambda^T(\mathbf{r}) + \frac{\rho}{2} \|\mathbf{r}\|_2^2, \quad (3)$$

with Lagrange dual variable  $\lambda$ , residual  $\mathbf{r} = \mathbf{Ax} + \mathbf{Bz} - \mathbf{c}$  and user-defined penalty parameter  $\rho > 0$ . We denote  $\boldsymbol{\mu} = \rho^{-1} \lambda$  and therefore the well-known scaled ADMM iterations are as follows:

$$\begin{aligned} \mathbf{x}^{k+1} &= \underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x}) + \frac{\rho}{2} \|\mathbf{Ax} + \mathbf{Bz}^k - \mathbf{c} + \boldsymbol{\mu}^k\|_2^2 \\ \mathbf{z}^{k+1} &= \underset{\mathbf{z}}{\operatorname{argmin}} g(\mathbf{z}) + \frac{\rho}{2} \|\mathbf{Ax}^{k+1} + \mathbf{Bz} - \mathbf{c} + \boldsymbol{\mu}^k\|_2^2 \\ \boldsymbol{\mu}^{k+1} &= \boldsymbol{\mu}^k + \mathbf{Ax}^{k+1} + \mathbf{Bz}^{k+1} - \mathbf{c} \end{aligned} \quad (4)$$

where  $\mathbf{x}^k$  and  $\mathbf{z}^k$  denote the decision variables at iteration  $k$ .

We denote  $\mathbf{r}^k = \mathbf{A}\mathbf{x}^k + \mathbf{B}\mathbf{z}^k - \mathbf{c}$  and  $\mathbf{s}^k = \rho\mathbf{A}^T\mathbf{B}(\mathbf{z}^k - \mathbf{z}^{k-1})$  as the primal and dual residual at iteration  $k$ . Let  $\epsilon^p > 0$  and  $\epsilon^d > 0$  be the user defined stopping tolerances for the primal and dual residuals, respectively. Therefore a reasonable stopping criteria would be:

$$\mathbf{r}^k \leq \epsilon^p \quad \text{and} \quad \mathbf{s}^k \leq \epsilon^d. \quad (5)$$

## 2.1 ADMM for parametric quadratic programs

We consider convex parametric quadratic programs (PQPs) of the form:

$$\begin{aligned} & \underset{\mathbf{z}}{\text{minimize}} && \frac{1}{2}\mathbf{z}^T\mathbf{Q}(\theta)\mathbf{z} + \mathbf{z}^T\mathbf{p}(\theta) \\ & \text{subject to} && \mathbf{A}(\theta)\mathbf{z} = \mathbf{b}(\theta), \quad \mathbf{l}(\theta) \leq \mathbf{z} \leq \mathbf{u}(\theta), \end{aligned} \quad (6)$$

with decision variable  $\mathbf{z} \in \mathbb{R}^{d_z}$ . The objective function is therefore defined by a vector  $\mathbf{p}(\theta) \in \mathbb{R}^{d_z}$  and symmetric positive definite matrix  $\mathbf{Q}(\theta) \in \mathbb{R}^{d_z \times d_z}$ . Here,  $\mathbf{A}(\theta) \in \mathbb{R}^{d_{eq} \times d_z}$ ,  $\mathbf{b}(\theta) \in \mathbb{R}^{d_{eq}}$ ,  $\mathbf{l}(\theta) \in \mathbb{R}^{d_z}$  and  $\mathbf{u}(\theta) \in \mathbb{R}^{d_z}$  define the linear equality and box inequality constraints. We assume that all problem variables are parameterized by  $\theta$  and are therefore trainable when integrated in an end-to-end neural network; rather than simply being supplied by the user.

### 2.1.1 ADMM layer: forward-pass

Our ADMM layer solves Program (6) in the forward-pass by applying the ADMM algorithm as outlined in Section 2. Note that for ease of notation we temporarily drop the parameterization,  $\theta$ .

We define

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{x}^T\mathbf{p}, \quad (7)$$

with domain  $\{\mathbf{x} \mid \mathbf{A}\mathbf{x} = \mathbf{b}\}$ . Similarly, we define

$$g(\mathbf{z}) = \mathbb{I}_{\mathbf{l} \leq \mathbf{z} \leq \mathbf{u}}(\mathbf{z}) \quad (8)$$

where  $\mathbb{I}_{\mathbf{l} \leq \mathbf{z} \leq \mathbf{u}}(\mathbf{z})$  denotes the indicator function with respect to the linear inequality constraints. Program (6) is then recast to the following convex optimization Program:

$$\begin{aligned} & \text{minimize} && f(\mathbf{x}) + g(\mathbf{z}) \\ & \text{subject to} && \mathbf{x} - \mathbf{z} = \mathbf{0}. \end{aligned} \quad (9)$$

Applying the ADMM iterations, as defined by Equations (4), to Program (9) therefore gives the following iterative optimization algorithm:

$$\mathbf{x}^{k+1} = \underset{\{\mathbf{x} \mid \mathbf{A}\mathbf{x} = \mathbf{b}\}}{\operatorname{argmin}} \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{x}^T\mathbf{p} + \frac{\rho}{2}\|\mathbf{x} - \mathbf{z}^k + \boldsymbol{\mu}^k\|_2^2 \quad (10a)$$

$$\mathbf{z}^{k+1} = \underset{\{\mathbf{l} \leq \mathbf{z} \leq \mathbf{u}\}}{\operatorname{argmin}} \frac{\rho}{2} \|\mathbf{x}^{k+1} - \mathbf{z} + \boldsymbol{\mu}^k\|_2^2 \quad (10b)$$

$$\boldsymbol{\mu}^{k+1} = \boldsymbol{\mu}^k + \mathbf{x}^{k+1} - \mathbf{z}^{k+1} \quad (10c)$$

The ADMM algorithm, as defined by Equations (10), allows for efficient optimization of medium and large scale quadratic programs. Firstly, we note that (10b) is a least squares problem with box-inequality constraints, and therefore can be solved analytically. Specifically, we define the euclidean projection onto a set of box constraints as:

$$\Pi(\mathbf{x}) = \begin{cases} \mathbf{l}_j & \text{if } \mathbf{x}_j < \mathbf{l}_j \\ \mathbf{x}_j & \text{if } \mathbf{l}_j \leq \mathbf{x}_j \leq \mathbf{u}_j \\ \mathbf{u}_j & \text{if } \mathbf{x}_j > \mathbf{u}_j \end{cases}. \quad (11)$$

The analytic solution to Program (10b) is therefore given by:

$$\mathbf{z}^{k+1} = \Pi(\mathbf{x}^{k+1} + \boldsymbol{\mu}^k). \quad (12)$$

Furthermore, Program (10a) is an equality constrained quadratic program, which can also be solved analytically. Specifically, the KKT optimality conditions of Program (10a) can be expressed as the solution to the following linear system of equations:

$$\begin{bmatrix} \mathbf{Q} + \rho \mathbf{I}_x & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}^{k+1} \\ \boldsymbol{\eta}^{k+1} \end{bmatrix} = - \begin{bmatrix} \mathbf{p} - \rho(\mathbf{z}^k - \boldsymbol{\mu}^k) \\ -\mathbf{b} \end{bmatrix}, \quad (13)$$

with identity matrix  $\mathbf{I}_x \in \mathbb{R}^{d_x \times d_x}$ . Applying Eqs. (12) and (13) allows us to express the ADMM iterations in a simplified form:

$$\begin{bmatrix} \mathbf{x}^{k+1} \\ \boldsymbol{\eta}^{k+1} \end{bmatrix} = - \begin{bmatrix} \mathbf{Q} + \rho \mathbf{I}_x & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{p} - \rho(\mathbf{z}^k - \boldsymbol{\mu}^k) \\ -\mathbf{b} \end{bmatrix} \quad (14a)$$

$$\mathbf{z}^{k+1} = \Pi(\mathbf{x}^{k+1} + \boldsymbol{\mu}^k) \quad (14b)$$

$$\boldsymbol{\mu}^{k+1} = \boldsymbol{\mu}^k + \mathbf{x}^{k+1} - \mathbf{z}^{k+1} \quad (14c)$$

Observe that the per-iteration cost of the ADMM algorithm is dominated by solving the system of Eq. (13). Note, however, that this linear system is in general smaller than the Newton system found in a standard primal-dual interior-point solvers by a factor of approximately 5, and therefore remains tractable for medium and large scale problems. Furthermore, if  $\rho$  is static then the left-hand side matrix in Eq. (13) remains unchanged at each iteration and therefore is factorized only once at the onset of the ADMM algorithm. Moreover, if the matrices  $\mathbf{Q}$  and  $\mathbf{A}$  remain unchanged at each epoch of gradient descent, then the left-hand-side matrix needs to only be factorized **once** during the entire training process.

### 2.1.2 ADMM layer: unrolled differentiation

Note that the standard unrolled differentiation approach ‘unrolls’ the iterations in Equation (14) by standard backpropagation [30] and requires that each operation in Equation (14) be differentiable. We refer to Domke [15] and Diamond et al. [14] for a more comprehensive overview of unrolled differentiation. Our unrolled differentiation is invoked by the standard auto-differentiation routine in the *torch* library.

### 2.1.3 ADMM layer: KKT implicit differentiation

As an alternative to unrolled differentiation, we note that the system of equations provided by the KKT conditions at optimality is a fixed-point mapping. As outlined by [3, 4], it is therefore possible to apply the implicit function theorem and derive the gradient of the primal-dual variables with respect to the problem variables in Program (6). In this section we derive the KKT implicit differentiation for our ADMM algorithm. We begin with a few definitions.

**Definition 1** Let  $F : \mathbb{R}^{d_v} \times \mathbb{R}^{d_\theta} \rightarrow \mathbb{R}^{d_v}$  be a continuously differentiable function with variable  $\mathbf{v}$  and parameter  $\theta$ . We define  $\mathbf{v}^*$  as a **fixed-point** of  $F$  at  $(\mathbf{v}^*, \theta)$  if:

$$F(\mathbf{v}^*, \theta) = \mathbf{v}^*.$$

**Definition 2** The **residual map**,  $G : \mathbb{R}^{d_v} \times \mathbb{R}^{d_\theta} \rightarrow \mathbb{R}^{d_v}$  of a fixed point,  $(\mathbf{v}^*, \theta)$ , of  $F$  is given by:

$$G(\mathbf{v}^*, \theta) = F(\mathbf{v}^*, \theta) - \mathbf{v}^* = 0.$$

The **implicit function theorem**, as defined by Dontchev and Rockafellar [16], then provides the conditions on  $G$  for which the Jacobian of the solution mapping with respect to  $\theta$  is well defined.

**Theorem 1** Let  $G : \mathbb{R}^{d_v} \times \mathbb{R}^{d_\theta} \rightarrow \mathbb{R}^{d_v}$  be a continuously differentiable function in a neighborhood of  $(\mathbf{v}^*, \theta)$  such that  $G(\mathbf{v}^*, \theta) = 0$ . Denote the nonsingular partial Jacobian of  $G$  with respect to  $\mathbf{v}$  as  $\nabla_{\mathbf{v}} G(\mathbf{v}^*, \theta)$ . Then  $\mathbf{v}(\theta)$  is an implicit function of  $\theta$  and is continuously differentiable in a neighborhood,  $\Theta$ , of  $\theta$  with Jacobian:

$$\nabla_{\theta} \mathbf{v}(\theta) = -[\nabla_{\mathbf{v}} G(\mathbf{v}(\theta), \theta)]^{-1} \nabla_{\theta} G(\mathbf{v}(\theta), \theta) \quad \forall \quad \theta \in \Theta. \quad (15)$$

**Corollary 1** Let  $F : \mathbb{R}^{d_v} \times \mathbb{R}^{d_\theta} \rightarrow \mathbb{R}^{d_v}$  be a continuously differentiable function with fixed-point  $(\mathbf{v}^*, \theta)$ . Then  $\mathbf{v}(\theta)$  is an implicit function of  $\theta$  and is continuously differentiable in a neighborhood,  $\Theta$ , of  $\theta$  with Jacobian:

$$\nabla_{\theta} \mathbf{v}(\theta) = [\mathbf{I}_v - \nabla_{\mathbf{v}} F(\mathbf{v}(\theta), \theta)]^{-1} \nabla_{\theta} F(\mathbf{v}(\theta), \theta) \quad \forall \quad \theta \in \Theta. \quad (16)$$

For constrained quadratic programming, let us denote the primal-dual solution at optimality by  $\mathbf{v}^* = (\mathbf{z}^*, \tilde{\lambda}^*, \boldsymbol{\eta}^*)$ , where  $\mathbf{z}^*$  and  $\boldsymbol{\eta}^*$  are defined by Equations (13) at

optimality. Note that the dual variables associated with the inequality constraints are given by  $\tilde{\lambda}^* = (\lambda_-^*, \lambda_+^*)$  with:

$$\lambda_-^* = -\min(\rho\mu^*, 0) \quad \text{and} \quad \lambda_+^* = \max(\rho\mu^*, 0). \quad (17)$$

We define the box inequality constraints as  $\mathbf{Gz} \leq \mathbf{h}$ , where

$$\mathbf{G} = \begin{bmatrix} -\mathbf{I}_x \\ \mathbf{I}_x \end{bmatrix} \quad \text{and} \quad \mathbf{h} = \begin{bmatrix} -\mathbf{l} \\ \mathbf{u} \end{bmatrix}.$$

Note that all constraints are affine and therefore Slater's condition reduces to feasibility. The KKT conditions for stationarity, primal feasibility, and complementary slackness therefore defines a fixed-point at optimality  $\mathbf{v}^*$  given by:

$$G(\mathbf{v}^*, \theta) = \begin{bmatrix} \mathbf{p} + \mathbf{Qz}^* + \mathbf{G}^T \tilde{\lambda}^* + \mathbf{A}^T \boldsymbol{\eta}^* \\ \text{diag}(\tilde{\lambda}^*)(\mathbf{Gz}^* - \mathbf{h}) \\ \mathbf{Az}^* - \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \quad (18)$$

Applying Theorem 1, we take the differential of these conditions to give the following system of equations:

$$\begin{bmatrix} \mathbf{Q} & \mathbf{G}^T & \mathbf{A}^T \\ \text{diag}(\tilde{\lambda}^*)\mathbf{G} & \text{diag}(\mathbf{Gz}^* - \mathbf{h}) & 0 \\ \mathbf{A} & 0 & 0 \end{bmatrix} \begin{bmatrix} d\mathbf{z} \\ d\boldsymbol{\lambda} \\ d\mathbf{v} \end{bmatrix} = - \begin{bmatrix} d\mathbf{Qz}^* + d\mathbf{p} + d\mathbf{G}^T \tilde{\lambda}^* + d\mathbf{A}^T \boldsymbol{\eta}^* \\ \text{diag}(\tilde{\lambda}^*)d\mathbf{Gz}^* - \text{diag}(\tilde{\lambda}^*)d\mathbf{h} \\ d\mathbf{Az}^* - d\mathbf{b} \end{bmatrix}. \quad (19)$$

Observe that the left side matrix gives the optimality conditions of the convex quadratic problem, which, when solving by interior-point methods, must be factorized in order to obtain the solution to the nominal program [8]. The right side gives the differentials of the relevant functions at the achieved solution with respect to any of the problem variables. In practice, however, we never explicitly form the right-side Jacobian matrix directly. Instead we follow the work of Amos and Kolter [3] and compute the left matrix-vector product of the Jacobian with the previous backward-pass gradient,  $\frac{\partial \ell}{\partial \mathbf{z}^*}$ , as outlined below:

$$\begin{bmatrix} \bar{d}_z \\ \bar{d}_\lambda \\ \bar{d}_\eta \end{bmatrix} = - \begin{bmatrix} \mathbf{Q} & \mathbf{G}^T \text{diag}(\tilde{\lambda}^*) & \mathbf{A}^T \\ \mathbf{G} & \text{diag}(\mathbf{Gz}^* - \mathbf{h}) & 0 \\ \mathbf{A} & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} \left(\frac{\partial \ell}{\partial \mathbf{z}^*}\right)^T \\ 0 \\ 0 \end{bmatrix}. \quad (20)$$

Equation (20) allows for efficient computation of the gradients with respect to any of the relevant problem variables. This is particularly true when using interior-point methods as the required gradients are effectively obtained ‘for free’ upon factorization of the left matrix when obtaining the solution,  $\mathbf{z}^*$ , in the forward-pass. In the ADMM algorithm, however, we never explicitly solve the KKT system of equations and therefore we must form and factorize the left side KKT matrix during the backward-pass routine. Observe that the left-side matrix in Equation (20) is of dimension  $3d_z + d_{eq}$  and therefore for large scale problems solving this system of equations can be computationally burdensome. Finally, for the reader's interest, we state the



gradients for all problem variables and refer the reader to Amos and Kolter [3] for their derivation.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{Q}} &= \frac{1}{2} \left( \bar{\mathbf{d}}_z \mathbf{z}^{*T} + \mathbf{z}^* \bar{\mathbf{d}}_z^T \right) & \frac{\partial \mathcal{L}}{\partial \mathbf{p}} &= \bar{\mathbf{d}}_z \\ \frac{\partial \mathcal{L}}{\partial \mathbf{A}} &= \bar{\mathbf{d}}_\eta \mathbf{z}^{*T} + \eta^* \bar{\mathbf{d}}_z^T & \frac{\partial \mathcal{L}}{\partial \mathbf{b}} &= -\bar{\mathbf{d}}_\eta \\ \frac{\partial \mathcal{L}}{\partial \mathbf{G}} &= \text{diag}(\lambda^*) \bar{\mathbf{d}}_\lambda \mathbf{z}^{*T} + \lambda^* \bar{\mathbf{d}}_z^T & \frac{\partial \mathcal{L}}{\partial \mathbf{h}} &= -\text{diag}(\lambda^*) \bar{\mathbf{d}}_\lambda\end{aligned}\quad (21)$$

### 2.1.4 ADMM layer: fixed-point implicit differentiation

In this section we demonstrate that the ADMM iterations in Equation (14) can be cast as a fixed-point iteration of dimension  $d_z + d_{eq}$ . In many applications,  $d_{eq}$  is typically much smaller than  $d_z$ , and therefore the proposed fixed-point differentiation will almost certainly decrease the computational overhead of the backward-pass routine. We begin with the following proposition. Note that all proofs are available in the Appendix.

**Proposition 1** *Let  $\mathbf{v}^k = \mathbf{x}^{k+1} + \boldsymbol{\mu}^k$ ,  $\tilde{\mathbf{v}}^k = (\mathbf{v}^k, \boldsymbol{\eta}^k)$  and define  $F : \mathbb{R}^{d_v} \times \mathbb{R}^{d_\theta} \rightarrow \mathbb{R}^{d_v}$ . Then the ADMM iterations in Equation (14) can be cast as a fixed-point iteration of the form  $F(\tilde{\mathbf{v}}, \boldsymbol{\theta}) = \tilde{\mathbf{v}}$  given by:*

$$\begin{bmatrix} \mathbf{v}^{k+1} \\ \boldsymbol{\eta}^{k+2} \end{bmatrix} = - \begin{bmatrix} \mathbf{Q} + \rho \mathbf{I}_v & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{p} - \rho(2\Pi(\mathbf{v}^k) - \mathbf{v}^k) \\ -\mathbf{b} \end{bmatrix} + \begin{bmatrix} \mathbf{v}^k \\ \boldsymbol{\eta}^{k+1} \end{bmatrix} - \begin{bmatrix} \Pi(\mathbf{v}^k) \\ \boldsymbol{\eta}^{k+1} \end{bmatrix}. \quad (22)$$

We follow Busseti et al. [10] and define the derivative of the projection operator,  $\Pi$  as:

$$D\Pi(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x}_j < \mathbf{l}_j \\ 1 & \text{if } \mathbf{l}_j \leq \mathbf{x}_j \leq \mathbf{u}_j \\ 0 & \text{if } \mathbf{x}_j > \mathbf{u}_j \end{cases}. \quad (23)$$

Observe that  $D\Pi(\mathbf{x})$  is not continuously differentiable when  $\mathbf{x}_j = \mathbf{l}_j$  or  $\mathbf{x}_j = \mathbf{u}_j$ . In practice, we can overcome the non-differentiability of  $\Pi$  by introducing a small perturbation to  $\mathbf{x}$ , thus moving  $\mathbf{x}$  away from the boundaries. Alternatively, smooth sigmoid based approximations to  $\Pi(\mathbf{x})$  may also be suitable. In all experiments below, however, we invoke  $D\Pi(\mathbf{x})$  directly as defined in Equation (23).

The Jacobian,  $\nabla_{\tilde{\mathbf{v}}} F$ , is therefore defined as:

$$\nabla_{\tilde{\mathbf{v}}} F = - \begin{bmatrix} \mathbf{Q} + \rho \mathbf{I}_v & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix}^{-1} \begin{bmatrix} -\rho(2D\Pi(\mathbf{v}) - \mathbf{I}_v) & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} \mathbf{I}_v & 0 \\ 0 & \mathbf{I}_\eta \end{bmatrix} - \begin{bmatrix} D\Pi(\mathbf{v}) & 0 \\ 0 & \mathbf{I}_\eta \end{bmatrix}. \quad (24)$$

Corollary 1 therefore gives the desired Jacobian,  $\nabla_{\boldsymbol{\theta}} \tilde{\mathbf{v}}(\boldsymbol{\theta})$ , with respect to the parameter  $\boldsymbol{\theta}$ :

$$\nabla_{\theta} \tilde{\mathbf{v}}(\theta) = [\mathbf{I}_{\tilde{\mathbf{v}}} - \nabla_{\tilde{\mathbf{v}}} F(\tilde{\mathbf{v}}(\theta), \theta)]^{-1} \nabla_{\theta} F(\tilde{\mathbf{v}}(\theta), \theta). \quad (25)$$

From the definition of  $\mathbf{v}$  we have that the Jacobians  $\nabla_{\theta} \mathbf{x}(\theta)$  and  $\nabla_{\theta} \boldsymbol{\eta}(\theta)$  are given by:

$$\begin{bmatrix} \nabla_{\theta} \mathbf{x}(\theta) \\ \nabla_{\theta} \boldsymbol{\eta}(\theta) \end{bmatrix} = \begin{bmatrix} D\Pi(\mathbf{v}) & 0 \\ 0 & \mathbf{I}_{\eta} \end{bmatrix} [\mathbf{I}_{\tilde{\mathbf{v}}} - \nabla_{\tilde{\mathbf{v}}} F(\tilde{\mathbf{v}}(\theta), \theta)]^{-1} \nabla_{\theta} F(\tilde{\mathbf{v}}(\theta), \theta) \quad (26)$$

As before we never form the Jacobians  $\nabla_{\theta} \mathbf{x}(\theta)$  and  $\nabla_{\theta} \boldsymbol{\eta}(\theta)$  directly. Instead, we compute the left matrix-vector product of the Jacobian with the previous backward-pass gradient,  $\frac{\partial \ell}{\partial \mathbf{z}^*}$ , as outlined below.

**Proposition 2** Let  $\hat{\mathbf{d}}_{\mathbf{x}}$  and  $\hat{\mathbf{d}}_{\eta}$  be defined as:

$$\begin{aligned} \begin{bmatrix} \hat{\mathbf{d}}_{\mathbf{x}} \\ \hat{\mathbf{d}}_{\eta} \end{bmatrix} &= \begin{bmatrix} \mathbf{Q} + \rho \mathbf{I}_{\mathbf{v}} & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix}^{-1} [\mathbf{I}_{\tilde{\mathbf{v}}} - \nabla_{\tilde{\mathbf{v}}} F(\tilde{\mathbf{v}}(\theta), \theta)]^{-T} \begin{bmatrix} D\Pi(\mathbf{v}) & 0 \\ 0 & \mathbf{I}_{\eta} \end{bmatrix} \begin{bmatrix} (-\frac{\partial \ell}{\partial \mathbf{z}^*})^T \\ 0 \end{bmatrix} \\ &= \left[ \begin{bmatrix} D\Pi(\mathbf{v}) & 0 \\ 0 & \mathbf{I}_{\eta} \end{bmatrix} \begin{bmatrix} \mathbf{Q} + \rho \mathbf{I}_{\mathbf{v}} & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix} + \begin{bmatrix} -\rho(2D\Pi(\mathbf{v}) - \mathbf{I}_{\mathbf{v}}) & 0 \\ 0 & 0 \end{bmatrix} \right]^{-1} \\ &\quad \begin{bmatrix} D\Pi(\mathbf{v}) & 0 \\ 0 & \mathbf{I}_{\eta} \end{bmatrix} \begin{bmatrix} (-\frac{\partial \ell}{\partial \mathbf{z}^*})^T \\ 0 \end{bmatrix}. \end{aligned} \quad (27)$$

Then the gradients of the loss function,  $\ell$ , with respect to problem variables  $\mathbf{Q}$ ,  $\mathbf{p}$ ,  $\mathbf{A}$  and  $\mathbf{b}$  are given by:

$$\begin{aligned} \frac{\partial \ell}{\partial \mathbf{Q}} &= \frac{1}{2} (\hat{\mathbf{d}}_{\mathbf{x}} \mathbf{x}^{*T} + \mathbf{x}^* \hat{\mathbf{d}}_{\mathbf{x}}^T) & \frac{\partial \ell}{\partial \mathbf{p}} &= \hat{\mathbf{d}}_{\mathbf{x}} \\ \frac{\partial \ell}{\partial \mathbf{A}} &= \hat{\mathbf{d}}_{\eta} \mathbf{x}^{*T} + \boldsymbol{\eta}^* \hat{\mathbf{d}}_{\mathbf{x}}^T & \frac{\partial \ell}{\partial \mathbf{b}} &= -\hat{\mathbf{d}}_{\eta} \end{aligned} \quad (28)$$

Computing the gradients of the loss with respect to the box constraint variables,  $\mathbf{l}$  and  $\mathbf{u}$  is also straightforward.

**Proposition 3** Define  $\tilde{\boldsymbol{\mu}}^*$  and  $\hat{\mathbf{d}}_{\lambda}$  as:

$$\tilde{\boldsymbol{\mu}}_j^* = \begin{cases} \boldsymbol{\mu}_j^* & \text{if } \boldsymbol{\mu}_j^* \neq 0 \\ 1 & \text{otherwise,} \end{cases} \quad (29)$$

and

$$\hat{\mathbf{d}}_{\lambda} = \text{diag}(\rho \tilde{\boldsymbol{\mu}}^*)^{-1} \left( - \left( \frac{\partial \ell}{\partial \mathbf{z}^*} \right)^T - \mathbf{Q} \hat{\mathbf{d}}_{\mathbf{x}} - \mathbf{A}^T \hat{\mathbf{d}}_{\eta} \right). \quad (30)$$

Then the gradients of the loss function,  $\ell$ , with respect to problem variables  $\mathbf{l}$  and  $\mathbf{u}$  are given by:

$$\frac{\partial \ell}{\partial \mathbf{l}} = \text{diag}(\boldsymbol{\lambda}_-^*) \hat{\mathbf{d}}_{\lambda} \quad \frac{\partial \ell}{\partial \mathbf{u}} = -\text{diag}(\boldsymbol{\lambda}_+^*) \hat{\mathbf{d}}_{\lambda}. \quad (31)$$

We now have a framework for computing the gradient with respect to all problem variables by implicit differentiation of the fixed-point mapping of the transformed ADMM iterations. We re-iterate that the implicit differentiation of the KKT conditions requires solving a system of equations on the order of  $3d_z + d_{eq}$ . In contrast, the fixed-point iteration, presented in Equation (22) is of dimension  $d_z + d_{eq}$ . As we will demonstrate shortly, reducing the dimension of the fixed-point mapping results in a considerable improvement in computational efficiency in the backward-pass.

### 3 Computational experiments

We present several experimental results that highlight the computational efficiency and performance accuracy of the ADMM layer. In all experiments, computational efficiency is measured by the average runtime (in seconds), required to implement the forward-pass and backward-pass algorithms of each method. We compare across 5 methods:

1. *ADMM FP* ADMM in the forward-pass and fixed-point implicit differentiation in the backward-pass.
2. *ADMM KKT* ADMM in the forward-pass and KKT implicit differentiation in the backward-pass.
3. *ADMM Unroll* ADMM in the forward-pass and unrolled differentiation in the backward-pass.
4. *OptNet* primal-dual interior-point method in the forward-pass and efficient KKT implicit differentiation in the backward-pass.
5. *SCS* serial version of splitting cone solver (SCS) [29] in the forward-pass and fixed-point implicit differentiation in the backward-pass. See [1] for more details.

The forward-pass of each solver terminates when the mean  $L_2$ -norm of the primal and dual residuals are sufficiently small (i.e. less than some user-defined tolerance  $\epsilon_{\text{tol}}$ ). In many applications, however, it is not always necessary to solve the batch QPs exactly during training. We therefore consider and compare the computational efficiency of each method over several stopping tolerances:  $\epsilon_{\text{tol}} \in \{10^{-1}, 10^{-3}, 10^{-5}\}$ . Going forward, the method label ‘ADMM FP 3’, for example, denotes the ADMM FP method with a stopping tolerance of  $10^{-3}$ .

Lastly, first-order methods are known to be vulnerable to ill-conditioned problems and the resulting convergence rates can vary significantly when the data and algorithm parameters ( $\rho$ ) are poorly scaled. Many first-order solvers therefore implement a preconditioning and problem scaling initialization step (see for example [29, 31, 33]). In our case, however, the QP problem variables are parameterized and are therefore expected to change at each epoch. Scaling and conditioning the problem variables at each epoch would potentially result in excessive computational overhead. As a result, our ADMM layer implementation does not include a preconditioning step. Instead, in all experiments presented below, we normalize the problem data to have approximately unit standard deviation (on average) and manually scale

the problem variables:  $\mathbf{p}$ ,  $\mathbf{Q}$ ,  $\mathbf{A}$  and  $\mathbf{b}$  where appropriate. We find that for unit-scaled problem data, a value of  $\rho \in \{0.10, 1.0\}$  provides a consistent rate of convergence. Indeed, an efficient and dynamic preconditioning and scaling routine is an interesting area of future research.

All experiments are conducted on an Apple Mac Pro computer (2.7 GHz 12-Core Intel Xeon E5, 128 GB 1066 MHz DDR3 RAM) running macOS ‘Catalina’. The software was written using the R programming language (version 4.0.0) and torch (version 0.6.0).

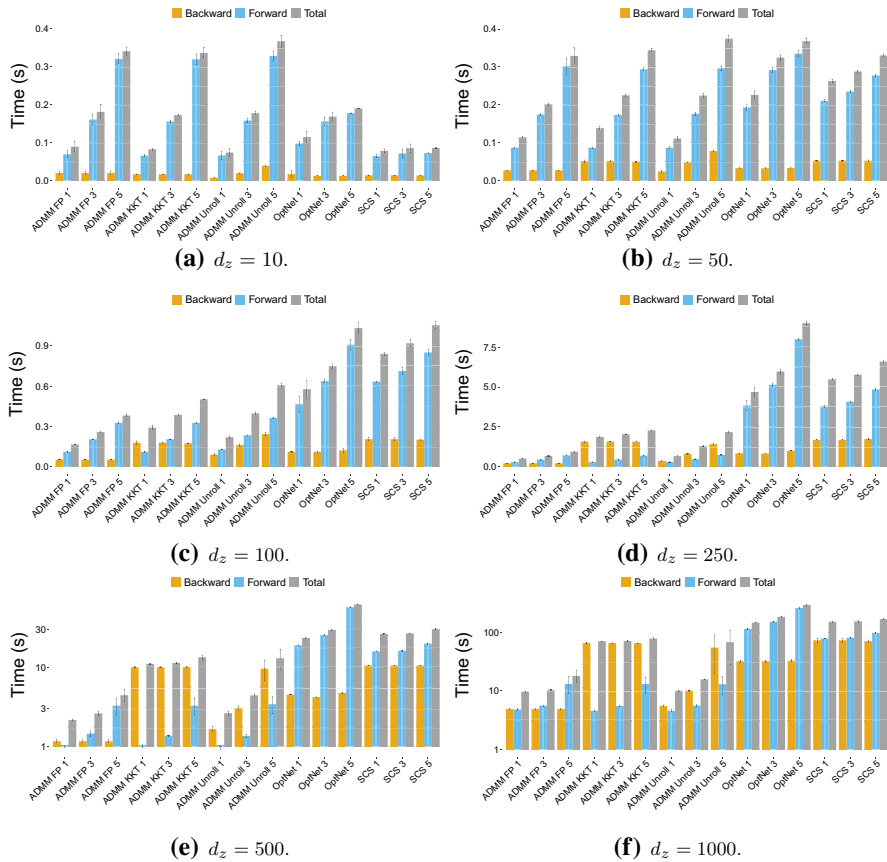
### 3.1 Experiment 1: ADMM layer performance

We conduct an experiment comparing the computational efficiency of the ADMM, OptNet and SCS methods with various stopping tolerances. We randomly generate problem data of dimension:

$$d_z \in \{10, 50, 100, 250, 500, 1000\}$$

and for each trial implement the forward and backward algorithms on a mini-batch size of 128. Problem variables are generated as follows. We set  $\mathbf{Q} = \frac{1}{2d_z} \mathbf{U}^T \mathbf{U}$  where entries of  $\mathbf{U} \in \mathbb{R}^{2d_z \times d_z}$  are sampled from a standard normal distribution. We randomly generate  $\mathbf{p}$  by sampling from the standard normal distribution and randomly generate  $\mathbf{l}$  and  $\mathbf{u}$  by randomly sampling from the uniform distribution with domain  $[-2, -1]$  and  $[1, 2]$ , respectively. Finally we set  $\mathbf{A} = \mathbf{I}$  and  $\mathbf{b} = \mathbf{1}$ .

Figure 1 provides the average runtime and 95%-ile confidence interval, evaluated over 10 trials, of the forward and backward-pass algorithms. We refer the reader to Table 4 in Appendix A.5 for a summary of the relative computational efficiency benchmarked against the ADMM FP method. We make several important observations. First, for small scale problems ( $d_z \leq 100$ ), there is negligible performance differences across all methods of the same stopping tolerance. As expected, the total runtime increases as the required stopping tolerance decreases. For medium to large scale problems ( $100 < d_z \leq 1000$ ) we observe a substantial performance degradation in the ADMM KKT, OptNet and SCS methods, in comparison to the ADMM FP and ADMM Unroll methods. Specifically, the OptNet and SCS methods exhibit an increase in total computation time that is anywhere from 6.8x to 17.6x larger than the corresponding ADMM FP implementation. For example, when  $d_z = 1000$  and  $\epsilon_{\text{tol}} = 10^{-3}$ , the total runtime for the OptNet and SCS methods is 185 seconds and 155 seconds, respectively, whereas the total runtime for the ADMM method is less than 11 seconds; over an order of magnitude faster. This increase in computational performance will ultimately enable training architectures that can practically support substantially larger quadratic optimization problems. Furthermore, the ADMM KKT backward computation time is shown to be 6.9x to 13.5x larger than the corresponding fixed-point method. This result is not surprising as the ADMM KKT backward-pass algorithm must first form and then factorize the KKT system of equations, which is of dimension  $3d_z + d_{eq}$  whereas the fixed-point backward-pass routine solves a system of equations of size  $d_z + d_{eq}$ .



**Fig. 1** Computational performance of ADMM FP, ADMM KKT, ADMM Unroll, Optnet and SCS for various problem sizes,  $d_z$ , and stopping tolerances. Batch size = 128. Log axis used for  $d_z = 500$  and  $d_z = 1000$

Lastly, we note that the ADMM Unroll method is relatively efficient for small scale problems. Observe, however, that for medium scale problems, as the stopping tolerance decreases the computation time of the unrolled backward-pass increases and is anywhere from 1.1x to 11.3x slower than the fixed-point method. In contrast, ADMM FP is invariant to the number of ‘inner’ iterations performed in the forward-pass, resulting in a 1.0x to 3.8x improvement in total computation time. Going forward, we choose to work with the ADMM FP method as it is most efficient for medium and large scale problems. Moreover, the OptNet and SCS methods provide very similar performance, particularly when  $\epsilon_{\text{tol}} > 10^{-5}$ . All subsequent experiments therefore compare the ADMM FP and the OptNet with  $\epsilon_{\text{tol}} > 10^{-5}$ .

### 3.2 Experiment 2: learning $\mathbf{p}$

We now consider a full training experiment whereby the objective is to learn a parameterized model for the variable  $\mathbf{p}$ , that is optimal in the context of the remaining QP problem variables. This problem was considered by Donti et al. [17] with applications to power scheduling and battery storage, and more recently by Butler and Kwon [12] for optimal return forecasting within the context of a mean-variance portfolio. We refer to the aforementioned work for more details.

The learning process can be posed as a bi-level optimization program where the objective is to learn a parameter  $\theta$  in order to minimize the average QP loss induced by the optimal decision policies  $\{\mathbf{z}^*(\theta)^{(i)}\}_{i=1}^m$ . Program (32) is referred to as an integrated predict and optimize (IPO) model as the prediction model for  $\mathbf{p}$  is fully integrated with the resulting down-stream decision-based optimization model.

$$\begin{aligned} & \underset{\theta}{\text{minimize}} && \frac{1}{m} \sum_{i=1}^m \left( \mathbf{z}^*(\theta)^{T(i)} \mathbf{p}^{(i)} + \frac{1}{2} \mathbf{z}^*(\theta)^{T(i)} \mathbf{Q}^{(i)} \mathbf{z}^*(\theta)^{(i)} \right) \\ & \text{subject to} && \mathbf{z}^*(\theta)^{(i)} = \underset{\mathbf{z}}{\operatorname{argmin}} -\mathbf{z}^T \hat{\mathbf{p}}(\theta)^{(i)} + \frac{1}{2} \mathbf{z}^T \hat{\mathbf{Q}}^{(i)} \mathbf{z} \quad \forall i = 1, \dots, m \quad (32) \\ & && \mathbf{A} \mathbf{z}^*(\theta)^{(i)} = \mathbf{b} \quad \forall i = 1, \dots, m \\ & && \mathbf{l} \leq \mathbf{z}^*(\theta)^{(i)} \leq \mathbf{u} \quad \forall i = 1, \dots, m. \end{aligned}$$

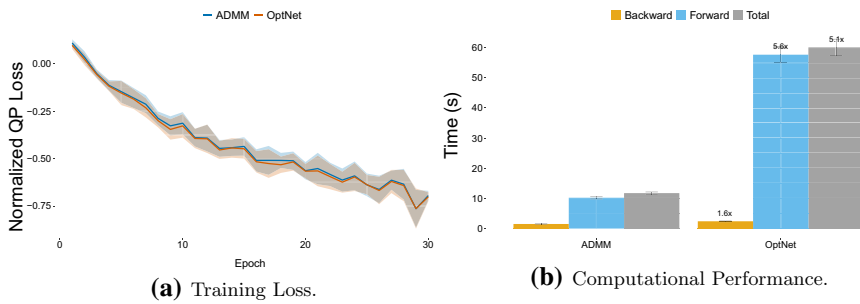
Here  $\mathbf{p}^{(i)}$  and  $\mathbf{Q}^{(i)}$  denote the ground truth problem data and are generated as follows. We let  $\mathbf{p}^{(i)} \sim \mathcal{N}(\mathbf{w}^{T(i)} \theta_0 + \tau \epsilon^{(i)}, \mathbf{Q})$  where  $\mathbf{Q} \in \mathbb{R}^{d_z \times d_z}$  has entry  $(j, k)$  equal to  $\rho_p^{|j-k|}$ . We set  $\rho_p = 0.50$  and generate the auxiliary feature data from the standard normal distribution,  $\mathbf{w}^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_w)$ . The residuals,  $\epsilon^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ , preserve the desired correlation structure and the scalar value  $\tau$  controls the signal-to-noise level. All experiments target a signal-to-noise level of 0.10.

We let  $\hat{\mathbf{p}}(\theta)^{(i)}$  denote the estimate of  $\mathbf{p}^{(i)}$  according to the linear model:

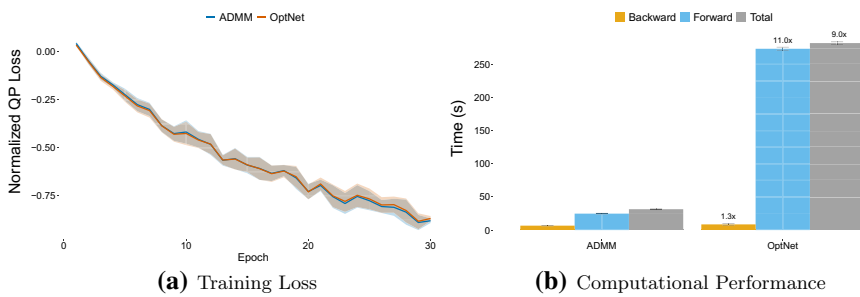
$$\hat{\mathbf{p}}(\theta)^{(i)} = \mathbf{w}^{T(i)} \theta. \quad (33)$$

The bound constraints,  $\mathbf{l}$  and  $\mathbf{u}$ , are generated by randomly sampling from the uniform distribution with domain  $[-1, 0]$  and  $[0, 1]$ , respectively and we set  $\mathbf{A} = \mathbf{1}$  and  $\mathbf{b} = 1$ . In all experiments we set the stopping tolerance to  $\epsilon_{\text{tol}} = 10^{-3}$ .

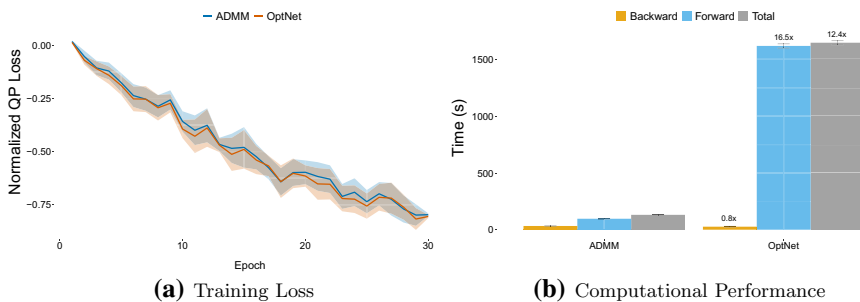
We randomly generate problem data of dimension  $d_z \in \{250, 500, 1000\}$ . The training process for each trial consists of 30 epochs with a mini-batch size of 32. Figures 2a, 3, 4a report the average training loss at each epoch and the 95%-ile confidence interval, evaluated over 10 independent trials. Observe that the loss curves for the ADMM model and OptNet model are almost identical at each epoch, suggesting that the training accuracy of the ADMM model and OptNet model are equivalent. Conversely, Figures 2b, 3, 4b, compare the average and 95%-ile confidence interval time spent executing the forward and backward pass algorithms. When  $d_z = 250$  the ADMM model is shown to be approximately 5 times faster than the OptNet model. Furthermore, when  $d_z = 500$  and  $d_z = 1000$ , the ADMM model is a full order of magnitude faster than the OptNet model.



**Fig. 2** Training loss and computational performance for learning  $\mathbf{p}$ . Batch size = 32 and  $d_z = 250$



**Fig. 3** Training loss and computational performance for learning  $\mathbf{p}$ . Batch size = 32 and  $d_z = 500$



**Fig. 4** Training loss and computational performance for learning  $\mathbf{p}$ . Batch size = 32 and  $d_z = 1000$

More concretely, when  $d_z = 1000$  the entire learning process takes approximately 1600 seconds to train the OptNet model, but less than 130 seconds to train the ADMM model to an equal level of accuracy.

### 3.3 Experiment 3: learning $\mathbf{A}$

We now present a real-world experiment from portfolio optimization whereby the objective is to learn a parameterized model for the variable  $\mathbf{A}$ . We consider an asset

universe of  $d_z = 254$  liquid US stocks traded on major U.S. exchanges (NYSE, NASDAQ, AMEX, ARCA). The universe is summarized in Table 3, with representative stocks from each of the Global Industry Classification Standard (GICS) sectors. Weekly price data is given from January 1990 through December 2020, and is provided by Quandl.

We denote the matrix of weekly return observations as  $\mathbf{A} = [\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \dots, \mathbf{a}^{(m)}] \in \mathbb{R}^{m \times d_z}$  with  $m > d_z$ . Let  $\mathbf{Q}^{(i)} \in \mathbb{R}^{d_z \times d_z}$  denote the symmetric positive definite covariance matrix of asset returns. We define the portfolio  $\mathbf{z}^{(i)} \in \mathbb{R}^{d_z}$ , where the element,  $\mathbf{z}_j^{(i)}$ , denotes the proportion of total capital invested in the  $j^{\text{th}}$  asset at time  $i$ .

We define the Sharpe ratio at observation  $i$  as the ratio of portfolio return to portfolio risk, where risk is measured by the portfolio volatility (standard deviation).

$$S_R^{(i)} = \frac{\mathbf{a}^{T(i)} \mathbf{z}^{(i)}}{\sqrt{\mathbf{z}^{T(i)} \mathbf{Q}^{(i)} \mathbf{z}^{(i)}}} \quad (34)$$

We consider a long-only ( $\mathbf{z}^{(i)} \geq 0$ ), fully invested ( $\mathbf{1}^T \mathbf{z}^{(i)} = 1$ ) max-Sharpe portfolio optimization, presented in Program (35):

$$\begin{aligned} & \underset{\mathbf{z}}{\text{maximize}} && \frac{\mathbf{a}^{T(i)} \mathbf{z}}{\sqrt{\mathbf{z}^T \mathbf{Q}^{(i)} \mathbf{z}}} \\ & \text{subject to} && \mathbf{1}^T \mathbf{z} = 1, \quad 0 \leq \mathbf{z} \leq 1 \end{aligned} \quad (35)$$

Observe, however, that the Sharpe ratio is not convex in  $\mathbf{z}$  but is homogeneous of degree zero. We follow Cornuejols and Tutuncu [13] and re-cast Program (35) as a convex quadratic optimization program:

$$\begin{aligned} & \underset{\mathbf{z}}{\text{minimize}} && \frac{1}{2} \mathbf{z}^T \mathbf{Q}^{(i)} \mathbf{z} \\ & \text{subject to} && \mathbf{a}^{T(i)} \mathbf{z} = 1, \quad \mathbf{z} \geq 0. \end{aligned} \quad (36)$$

Note that the fully-invested constraint can be enforced by normalizing the optimal weights,  $\mathbf{z}^*$ .

As before, the learning process can be posed as a bi-level optimization program where the objective is to learn a parameter  $\theta$  and the associated constraints,  $\hat{\mathbf{a}}(\theta)^{(i)}$ , in order to maximize the average realized Sharpe ratio induced by the optimal decision policies  $\{\mathbf{z}^*(\theta)^{(i)}\}_{i=1}^m$ .

$$\begin{aligned} & \underset{\theta}{\text{minimize}} && -\frac{1}{m} \sum_{i=1}^m \frac{\mathbf{a}^{T(i)} \mathbf{z}^*(\theta)}{\sqrt{\mathbf{z}^*(\theta)^{T(i)} \mathbf{Q}^{(i)} \mathbf{z}^*(\theta)^{(i)}}} \\ & \text{subject to} && \mathbf{z}^*(\theta) = \underset{\mathbf{z}}{\text{argmin}} \frac{1}{2} \mathbf{z}^T \mathbf{Q}^{(i)} \mathbf{z} \quad \forall i = 1, \dots, m \\ & && \hat{\mathbf{a}}(\theta)^{T(i)} \mathbf{z}^*(\theta)^{(i)} = 1, \quad \mathbf{1}^T \mathbf{z}^*(\theta)^{(i)} = 1, \quad \mathbf{z}^*(\theta)^{(i)} \geq 0 \quad \forall i = 1, \dots, m \end{aligned} \quad (37)$$



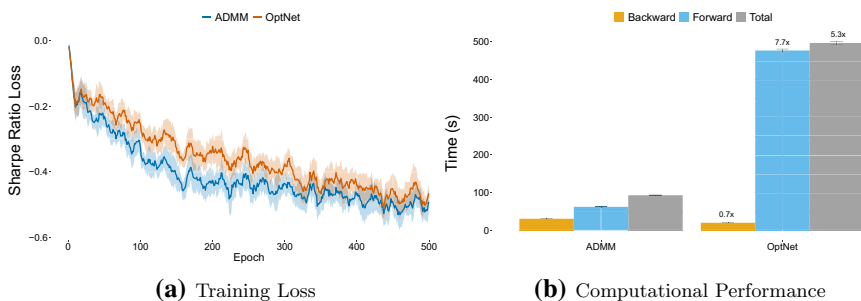
In reality, we do not know the true value of  $\mathbf{a}^{(i)}$  at decision time and instead we estimate  $\mathbf{a}^{(i)}$  through associated auxiliary feature variables  $\mathbf{w}^{(i)} \in \mathbb{R}^{d_w}$ . Again we consider a linear model of the form:

$$\hat{\mathbf{a}}^{(i)} = \boldsymbol{\theta}^T \mathbf{w}^{(i)}. \quad (38)$$

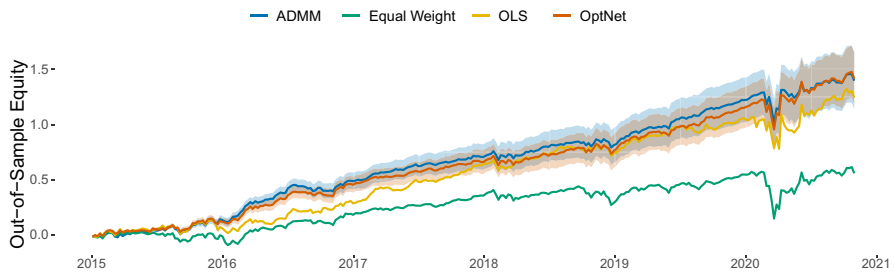
In this experiment, asset returns,  $\mathbf{a}^{(i)}$  are modelled using the well-known Fama-French Five (FF5) factor model [18], provided by the Kenneth R. French data library.

The goal is to observe the training and out-of-sample performance of the ADMM model in comparison to the OptNet model. As a benchmark, we include the out-of-sample performance of an equally weighted portfolio, and a max-Sharpe portfolio where  $\boldsymbol{\theta}$  is fit by ordinary least-squares (OLS). All experiments are trained on data from January 1990 through December 2014. The out-of-sample period begins in January 2015 and ends in December 2020. Portfolios are formed at the close of each week, and rebalanced on a weekly basis.

The training process for each trial consists of 500 epochs with a mini-batch size of 32. Portfolio models are fit to an accuracy of  $\epsilon_{\text{tol}} = 10^{-4}$  in training, and a higher accuracy of  $\epsilon_{\text{tol}} = 10^{-6}$  in the out-of-sample period in order to guarantee strict adherence to the constraint set. Figure 5a reports the average training loss at each epoch and the 95%-ile confidence interval, evaluated over 10 independent trials. Once again, we observe that the loss curves for the ADMM model and OptNet model are very similar. Interestingly, we observe that the ADMM model produces a consistently lower average training loss. Recall that both models use implicit differentiation to compute the relevant gradient, which assumes an exact fixed point at each optimal solution  $\mathbf{z}^*(\boldsymbol{\theta})^{(i)}$ . In practice, each  $\mathbf{z}^*(\boldsymbol{\theta})^{(i)}$  is only approximately optimal, to within a tolerance  $\epsilon_{\text{tol}}$ , and therefore differentiating at a solution that is not an exact fixed point will result in small errors in the gradient that likely explain the observed difference. That said, the training loss profile of the ADMM and OptNet models are very similar, and the final models achieve approximately equal loss after 500 epochs. Figure 5b compares the average and 95%-ile confidence interval of the total time spent executing the forward and backward pass algorithms during training. Once again we observe that



**Fig. 5** Training loss and computational performance for learning  $\mathbf{A}$  on US stock data. Batch size = 32 and  $d_z = 254$



**Fig. 6** Out-of-sample equity growth for ADMM IPO max-Sharpe portfolio, Equal Weight portfolio, OLS max-Sharpe portfolio and OptNet IPO max-Sharpe portfolio

**Table 1** Out-of-sample economic performance metrics for ADMM IPO max-Sharpe portfolio, Equal Weight portfolio, OLS max-Sharpe portfolio and OptNet IPO max-Sharpe portfolio

	ADMM	Equal weight	OLS	OptNet
Mean	0.2382	0.0950	0.2122	0.2413
Volatility	0.1777	0.1950	0.2435	0.1880
Sharpe ratio	1.3407	0.4872	0.8747	1.2836

the ADMM model is shown to be approximately 5 times faster than the OptNet model and requires less than 100 seconds to train.

Figure 6 reports the out-of-sample equity growth of the ADMM IPO max-Sharpe portfolio, Equal Weight portfolio, OLS max-Sharpe portfolio and OptNet IPO max-Sharpe portfolio. The out-of-sample economic performance metrics are reported in Table 1. First, observe that all max-Sharpe models outperform the Equal Weight benchmark on an absolute and risk-adjusted basis. Furthermore, the ADMM and OptNet IPO max-Sharpe models achieve an out-of-sample Sharpe ratio that is approximately 50% higher than that of the naive ‘predict, then optimize’ OLS max-Sharpe model, thus highlighting the benefit of training a fully integrated system. Lastly, the ADMM model achieves a marginally higher out-of-sample Sharpe ratio in comparison to the OptNet model, though the difference is not statistically significant.

### 3.4 Experiment 4: learning $Q$

We consider another real-world experiment from portfolio optimization whereby the objective is to learn a parameterized model for the variable  $Q$ . We use the same asset universe of  $d_z = 254$  liquid US stocks from Experiment 3 and an identical experimental design. Here, we consider the long-only, fully-invested minimum variance portfolio optimization, described in Program (39).

$$\begin{aligned} & \underset{\mathbf{z}}{\text{minimize}} \quad \frac{1}{2} \mathbf{z}^T \mathbf{Q}^{(i)} \mathbf{z} \\ & \text{subject to} \quad \mathbf{1}^T \mathbf{z} = 1, \quad 0 \leq \mathbf{z} \leq 1. \end{aligned} \quad (39)$$

As before, the learning process is posed as a bi-level optimization program where the objective is to learn a parameter  $\theta$  and the associated covariance matrix,  $\hat{\mathbf{Q}}(\theta)^{(i)}$ , in order to minimize the average realized variance induced by the optimal decision policies  $\{\mathbf{z}^*(\theta)^{(i)}\}_{i=1}^m$ .

$$\begin{aligned} & \underset{\theta}{\text{minimize}} \quad \frac{1}{m} \sum_{i=1}^m \mathbf{z}^*(\theta)^{T(i)} \mathbf{Q}^{(i)} \mathbf{z}^*(\theta)^{(i)} \\ & \text{subject to} \quad \mathbf{z}^*(\theta) = \underset{\mathbf{z}}{\text{argmin}} \quad \frac{1}{2} \mathbf{z}^T \hat{\mathbf{Q}}(\theta)^{(i)} \mathbf{z} \quad \forall i = 1, \dots, m \\ & \quad \mathbf{1}^T \mathbf{z}^*(\theta)^{(i)} = 1, \quad 0 \leq \mathbf{z}^*(\theta)^{(i)} \leq 1 \quad \forall i = 1, \dots, m \end{aligned} \quad (40)$$

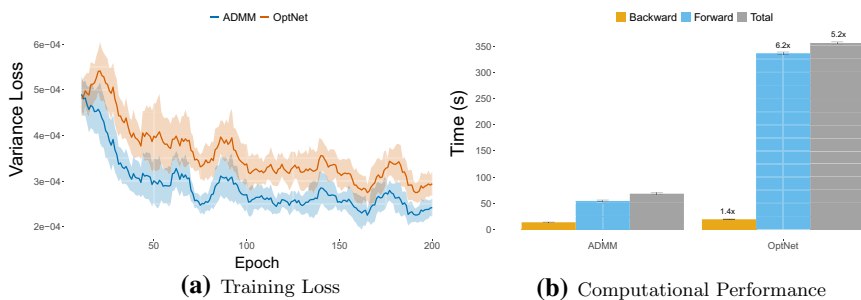
Asset returns,  $\mathbf{a}^{(i)}$  are modelled using the Fama-French Five (FF5) factor model. We follow Butler and Kwon [11] and model  $\mathbf{w}^{(i)}$  according to a multivariate GARCH(1, 1) process with constant correlation. We let  $\hat{\mathbf{W}}^{(i)}$  denote the time-varying covariance estimate of the auxiliary feature variables. We therefore model the stock covariance matrix as follows:

$$\hat{\mathbf{a}}^{(i)} = \theta^T \mathbf{w}^{(i)}, \quad \hat{\mathbf{Q}}^{(i)} = \theta^T \hat{\mathbf{W}}^{(i)} \theta + \hat{\mathbf{F}}, \quad (41)$$

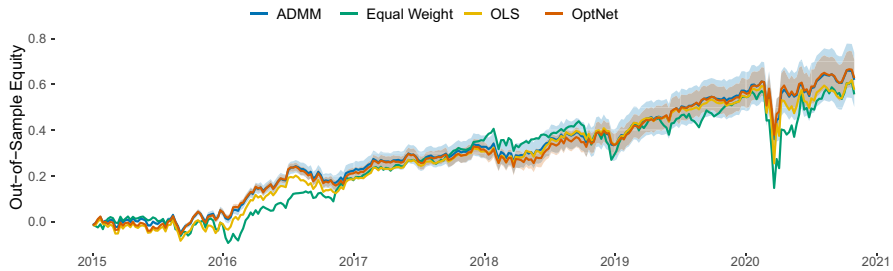
where  $\hat{\mathbf{F}}$  denotes the diagonal matrix of residual variances.

Again, the goal is to observe the training and out-of-sample performance of the ADMM model in comparison to the OptNet model. As a benchmark, we include the out-of-sample performance of the equal weight portfolio, and a minimum variance portfolio where  $\theta$  is fit by OLS. The training process for each trial consists of 200 epochs with a mini-batch size of 32. Portfolio models are fit to an accuracy of  $\epsilon_{\text{tol}} = 10^{-4}$  in training, and a higher accuracy of  $\epsilon_{\text{tol}} = 10^{-6}$  in the out-of-sample period.

Figure 7a reports the average training loss at each epoch and the 95%-ile confidence interval, evaluated over 10 independent trials. We observe that the loss curves for the ADMM model and OptNet model are very similar, thus



**Fig. 7** Training loss and computational performance for learning  $\mathbf{Q}$  on US stock data. Batch size = 32 and  $d_z = 254$



**Fig. 8** Out-of-sample equity growth for ADMM IPO minimum variance portfolio, Equal Weight portfolio, OLS minimum variance portfolio and OptNet IPO minimum variance portfolio

**Table 2** Out-of-sample economic performance metrics for ADMM IPO minimum variance portfolio, Equal Weight portfolio, OLS minimum variance portfolio and OptNet IPO minimum variance portfolio

	ADMM	Equal weight	OLS	OptNet
Mean	0.1058	0.0950	0.0984	0.1070
Volatility	0.1420	0.1950	0.1786	0.1443
Sharpe ratio	0.7446	0.4872	0.5510	0.7418

highlighting the accuracy of the ADMM layer. Once again we observe that the ADMM model produces a consistently lower average training loss and we refer to the discussion in Experiment 3 for a likely explanation. Figure 7b compares the average and 95%-ile confidence interval of the total time spent executing the forward and backward pass algorithms in training. As before, we observe that the ADMM model requires approximately 60 seconds to train and is approximately 5 times faster than the OptNet model, which requires over 300 seconds.

Finally, Figure 8 reports the out-of-sample equity growth of the ADMM IPO minimum variance portfolio, Equal Weight portfolio, OLS minimum variance portfolio and OptNet IPO minimum variance portfolio. The out-of-sample economic performance metrics are reported in Table 2. Again, observe that all minimum-variance models outperform the Equal Weight benchmark on an absolute and risk-adjusted basis. Furthermore, the ADMM and OptNet IPO minimum variance models achieve an out-of-sample volatility that is approximately 25% lower and Sharpe ratio that is approximately 35% higher than that of the naive ‘predict, then optimize’ OLS minimum variance model. These results are broadly consistent with the findings in Butler and Kwon [11], who consider an identical stock universe but with considerably smaller portfolios ( $d_z \leq 100$ ). Our ADMM model, on the other hand, is able to overcome the computational challenges in the medium to large scale limit (described in Butler and Kwon [11]), without any apparent loss in performance accuracy.

## 4 Conclusion and future work

In this paper, we provide a novel and efficient framework for differentiable constrained quadratic programming. Our differentiable optimization layer is built on top of the ADMM algorithm, which for medium to large scale problems is shown to be approximately an order of magnitude more efficient than current state-of-the-art layers. The backward-pass algorithm computes the relevant problem variable gradients by implicit differentiation of a custom fixed-point iteration, which is computationally favorable to KKT implicit differentiation and unrolled differentiation. Simulated and real problem data experiments demonstrates the efficacy of the ADMM layer, which for medium to large scale problems exhibits state-of-the-art accuracy and improved computational performance.

Our results should be interpreted as a proof-of-concept and we acknowledge that further testing on alternative data sets or with different problem variable assumptions is required in order to better determine the efficacy of the ADMM layer as a general purpose solver. Indeed, there is a plethora of areas for active research and algorithm improvement. First, our ADMM layer currently only supports linear equality and box inequality constraints, whereas the OptNet layer supports general linear inequality constraints. Indeed, incorporating more general inequality constraints as well as augmenting the QP with parameterized regularization norms is an active area of research. Secondly, the ADMM algorithm is known to be vulnerable to ill-conditioned problems and the resulting convergence rates can vary significantly when the data and algorithm parameter,  $\rho$ , are poorly scaled. To overcome this, most first-order solvers implement a preconditioning and scaling initialization step. The ADMM layer implementation does not currently support preconditioning and scaling, which is challenged by virtue of the fact that the problem data is expected to change at each epoch of training. Instead, we leave it to the user to select  $\rho$  and manually scale the problem data and acknowledge that preconditioning, scaling and automatic parameter selection is an important area of future development.

Furthermore, there are several heuristic methods that could be implemented in order to improve the convergence rates and computational efficiency of our ADMM forward-pass. For example, acceleration methods [5] have recently been shown to improve the convergence rates of first-order solvers [32, 37]. Indeed, applying acceleration methods to the modified fixed-point algorithm presented in this paper may provide a numerically efficient scheme for improving the convergence rate of the ADMM algorithm and is an interesting area of future research. Alternatively, methods that hybridize the efficiency of first-order methods with the precision of interior-point methods, such solution polishing and refinement [10], is another area of future exploration. Nonetheless, our proposed ADMM layer is shown to be highly effective and in its current form can be instrumental for efficiently solving real-world medium and large scale learning problems.

## A Appendix

### A.1 Proof of Proposition 1

We define  $\mathbf{v}^k = \mathbf{x}^{k+1} + \boldsymbol{\mu}^k$ . We can therefore express Equation (14b) as:

$$\mathbf{z}^{k+1} = \Pi(\mathbf{x}^{k+1} + \boldsymbol{\mu}^k) = \Pi(\mathbf{v}^k), \quad (42)$$

and Equation (14c) as:

$$\boldsymbol{\mu}^{k+1} = \boldsymbol{\mu}^k + \mathbf{x}^{k+1} - \mathbf{z}^{k+1} = \mathbf{v}^k - \Pi(\mathbf{v}^k). \quad (43)$$

Substituting Equations (42) and (43) into Equation (14a) gives the desired fixed-point iteration:

$$\begin{bmatrix} \mathbf{v}^{k+1} \\ \boldsymbol{\eta}^{k+2} \end{bmatrix} = \begin{bmatrix} \mathbf{x}^{k+2} + \boldsymbol{\mu}^{k+1} \\ \boldsymbol{\eta}^{k+2} \end{bmatrix} \quad (44)$$

$$= - \begin{bmatrix} \mathbf{Q} + \rho \mathbf{I}_v & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{p} - \rho(\mathbf{z}^{k+1} - \boldsymbol{\mu}^{k+1}) \\ -\mathbf{b} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\mu}^{k+1} \\ 0 \end{bmatrix} \quad (45)$$

$$= - \begin{bmatrix} \mathbf{Q} + \rho \mathbf{I}_v & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{p} - \rho(2\Pi(\mathbf{v}^k) - \mathbf{v}^k) \\ -\mathbf{b} \end{bmatrix} + \begin{bmatrix} \mathbf{v}^k \\ \boldsymbol{\eta}^{k+1} \end{bmatrix} - \begin{bmatrix} \Pi(\mathbf{v}^k) \\ \boldsymbol{\eta}^{k+1} \end{bmatrix}. \quad (46)$$

## A.2 Proof of Proposition 2

We define  $F : \mathbb{R}^{d_v} \times \mathbb{R}^{d_\eta} \rightarrow \mathbb{R}^{d_v} \times \mathbb{R}^{d_\eta}$  as:

$$F(\mathbf{v}, \boldsymbol{\eta}) = - \begin{bmatrix} \mathbf{Q} + \rho \mathbf{I}_v & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{p} - \rho(2\Pi(\mathbf{v}) - \mathbf{v}) \\ -\mathbf{b} \end{bmatrix} + \begin{bmatrix} \mathbf{v} \\ \boldsymbol{\eta} \end{bmatrix} - \begin{bmatrix} \Pi(\mathbf{v}) \\ \boldsymbol{\eta} \end{bmatrix}, \quad (47)$$

and let

$$\mathbf{M} = \begin{bmatrix} \mathbf{Q} + \rho \mathbf{I}_v & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix}. \quad (48)$$

Therefore we have

$$\mathbf{M}F(\mathbf{v}, \boldsymbol{\eta}) = - \begin{bmatrix} \mathbf{p} - \rho(2\Pi(\mathbf{v}) - \mathbf{v}) \\ -\mathbf{b} \end{bmatrix} + \mathbf{M} \begin{bmatrix} \mathbf{v} \\ \boldsymbol{\eta} \end{bmatrix} - \mathbf{M} \begin{bmatrix} \Pi(\mathbf{v}) \\ \boldsymbol{\eta} \end{bmatrix}. \quad (49)$$

Taking the partial differentials of Equation (49) with respect to the relevant problem variables therefore gives:

$$\begin{aligned} \mathbf{M}dF(\mathbf{v}, \boldsymbol{\eta}) &= - \begin{bmatrix} d\mathbf{p} \\ -d\mathbf{b} \end{bmatrix} + d\mathbf{M} \begin{bmatrix} \mathbf{v} \\ \boldsymbol{\eta} \end{bmatrix} - d\mathbf{M} \begin{bmatrix} \Pi(\mathbf{v}) \\ \boldsymbol{\eta} \end{bmatrix} - d\mathbf{M}F(\mathbf{v}, \boldsymbol{\eta}) \\ &= - \begin{bmatrix} d\mathbf{p} \\ -d\mathbf{b} \end{bmatrix} - d\mathbf{M} \left[ -\mathbf{M}^{-1} \begin{bmatrix} \mathbf{p} - \rho(2\Pi(\mathbf{v}) - \mathbf{v}) \\ -\mathbf{b} \end{bmatrix} \right] = - \begin{bmatrix} d\mathbf{p} \\ -d\mathbf{b} \end{bmatrix} - d\mathbf{M} \begin{bmatrix} \mathbf{x}^* \\ \boldsymbol{\eta}^* \end{bmatrix} \\ &= - \begin{bmatrix} d\mathbf{p} + \frac{1}{2}(\partial \mathbf{Q} \mathbf{x}^* + \partial \mathbf{Q}^T \mathbf{x}^*) + \partial \mathbf{A}^T \boldsymbol{\eta}^* \\ -d\mathbf{b} + \partial \mathbf{A} \mathbf{x}^* \end{bmatrix}. \end{aligned} \quad (50)$$

From Equation (49) we have that the differential  $\partial F(\mathbf{v}, \boldsymbol{\eta})$  is given by:

$$\partial F(\mathbf{v}, \boldsymbol{\eta}) = -\mathbf{M}^{-1} \begin{bmatrix} \partial \mathbf{p} + \frac{1}{2}(\partial \mathbf{Q} \mathbf{x}^* + \partial \mathbf{Q}^T \mathbf{x}^*) + \partial \mathbf{A}^T \boldsymbol{\eta}^* \\ -\partial \mathbf{b} + \partial \mathbf{A} \mathbf{x}^* \end{bmatrix}. \quad (51)$$

Substituting the gradient action of Equation (51) into Equation (26) and taking the left matrix-vector product of the transposed Jacobian with the previous backward-pass gradient,  $\frac{\partial \ell}{\partial \mathbf{z}^*}$ , gives the desired result.

$$\begin{bmatrix} \hat{\mathbf{d}}_{\mathbf{x}} \\ \hat{\mathbf{d}}_{\boldsymbol{\eta}} \end{bmatrix} = \begin{bmatrix} \mathbf{Q} + \rho \mathbf{I}_{\mathbf{v}} & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix}^{-1} \left[ \mathbf{I}_{\tilde{\mathbf{v}}} - \nabla_{\tilde{\mathbf{v}}} F(\tilde{\mathbf{v}}(\boldsymbol{\theta}), \boldsymbol{\theta}) \right]^{-T} \begin{bmatrix} D\Pi(\mathbf{v}) & 0 \\ 0 & \mathbf{I}_{\boldsymbol{\eta}} \end{bmatrix} \begin{bmatrix} \left(-\frac{\partial \ell}{\partial \mathbf{z}^*}\right)^T \\ 0 \end{bmatrix}. \quad (52)$$

From Equation (24) we have:

$$\mathbf{I}_{\tilde{\mathbf{v}}} - \nabla_{\tilde{\mathbf{v}}} F(\tilde{\mathbf{v}}(\boldsymbol{\theta}), \boldsymbol{\theta}) = \begin{bmatrix} \mathbf{Q} + \rho \mathbf{I}_{\mathbf{v}} & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix}^{-1} \begin{bmatrix} -\rho(2D\Pi(\mathbf{v}) - \mathbf{I}_{\mathbf{v}}) & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} D\Pi(\mathbf{v}) & 0 \\ 0 & \mathbf{I}_{\boldsymbol{\eta}} \end{bmatrix}. \quad (53)$$

Simplifying Equation (52) with Equation (53) yields the final expression:

$$\begin{bmatrix} \hat{\mathbf{d}}_{\mathbf{x}} \\ \hat{\mathbf{d}}_{\boldsymbol{\eta}} \end{bmatrix} = \left[ \begin{bmatrix} D\Pi(\mathbf{v}) & 0 \\ 0 & \mathbf{I}_{\boldsymbol{\eta}} \end{bmatrix} \begin{bmatrix} \mathbf{Q} + \rho \mathbf{I}_{\mathbf{v}} & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix} + \begin{bmatrix} -\rho(2D\Pi(\mathbf{v}) - \mathbf{I}_{\mathbf{v}}) & 0 \\ 0 & 0 \end{bmatrix} \right]^{-1} \begin{bmatrix} D\Pi(\mathbf{v}) & 0 \\ 0 & \mathbf{I}_{\boldsymbol{\eta}} \end{bmatrix} \begin{bmatrix} \left(-\frac{\partial \ell}{\partial \mathbf{z}^*}\right)^T \\ 0 \end{bmatrix}. \quad (54)$$

### A.3 Proof of Proposition 3

From the KKT system of equations (20) we have:

$$\mathbf{G}^T \text{diag}(\tilde{\lambda}^*) \hat{\mathbf{d}}_{\lambda} = \text{diag}(\rho \boldsymbol{\mu}^*) \hat{\mathbf{d}}_{\lambda} = \left( - \left( \frac{\partial \ell}{\partial \mathbf{z}^*} \right)^T - \mathbf{Q} \hat{\mathbf{d}}_{\mathbf{x}} - \mathbf{A}^T \hat{\mathbf{d}}_{\boldsymbol{\eta}} \right). \quad (55)$$

From Equation (21) it follows that:

$$\frac{\partial \ell}{\partial \mathbf{l}} \neq 0 \iff \lambda_{-}^* > 0 \quad \text{and} \quad \frac{\partial \ell}{\partial \mathbf{u}} \neq 0 \iff \lambda_{+}^* > 0, \quad (56)$$

and therefore Equation (55) uniquely determines the relevant non-zero gradients. Let  $\tilde{\boldsymbol{\mu}}^*$  be as defined by Equation (29), then it follows that:

$$\hat{\mathbf{d}}_{\lambda} = \text{diag}(\rho \tilde{\boldsymbol{\mu}}^*)^{-1} \left( - \left( \frac{\partial \ell}{\partial \mathbf{z}^*} \right)^T - \mathbf{Q} \hat{\mathbf{d}}_{\mathbf{x}} - \mathbf{A}^T \hat{\mathbf{d}}_{\boldsymbol{\eta}} \right). \quad (57)$$

Substituting  $\hat{\mathbf{d}}_{\lambda}$  into Equation (21) gives the desired gradients.

## A.4 Data Summary

See Table 3.

**Table 3** U.S. stock data, sorted by GICS Sector. Data provided by Quandl

GICS Sector	Stock symbols							
Communication Services	CBB	CMCSA	DIS	FOX	IPG	LUMN	MDP	NYT
	T	VOD	VZ					
Consumer Discretionary	BBY	CBRL	CCL	F	GPC	GPS	GT	HAS
	HD	HOG	HRB	JWN	LB	LEG	LEN	LOW
	MCD	NKE	NVR	NWL	PHM	PVH	ROST	TGT
	TJX	VFC	WHR	WWW				
Consumer Staples	ADM	ALCO	CAG	CASY	CHD	CL	CLX	COST
	CPB	FLO	GIS	HSY	K	KMB	KO	KR
	MO	PEP	PG	SYY	TAP	TR	TSN	UVV
	WBA	WMK	WMT					
Energy	AE	APA	BKR	BP	COP	CVX	EOG	HAL
	HES	MRO	OKE	OXY	SLB	VLO	WMB	XOM
Financials	AFG	AFL	AIG	AJG	AON	AXP	BAC	BEN
	BK	BXS	C	GL	JPM	L	LNC	MMC
	PGR	PNC	RJF	SCHW	STT	TROW	TRV	UNM
	USB	WFC	WRB	WTM				
Health Care	ABMD	ABT	AMGN	BAX	BDX	BIO	BMJ	CAH
	CI	COO	CVS	DHR	HUM	JNJ	LLY	MDT
	MRK	OMI	PFE	PKI	SYK	TFX	TMO	VTRS
	WST							
Industrials	ABM	AIR	ALK	AME	AOS	BA	CAT	CMI
	CSL	CSX	DE	DOV	EFX	EMR	ETN	FDX
	GD	GE	GWG	HON	IEX	ITW	JCI	KSU
	LMT	LUV	MAS	MMM	NOC	NPK	NSC	PCA
	RPH	PNR	ROK	ROL	RTX	SNA	SWK	TXT
	UNP							
Information Technology	AAPL	ADBE	ADI	ADP	ADSK	AMAT	AMD	GLW
	HPQ	IBM	INTC	MSFT	MSI	MU	ORCL	ROG
	SWKS	TER	TXN	TYL	WDC	XRX		
Materials	APD	AVY	BLL	CCK	CRS	ECL	FMC	GLT
	IFF	IP	MOS	NEM	NUE	OLN	PPG	SEE
	SHW	SON	VMC					
Real Estate	ALX	FRT	GTJ	HST	PEAK	PSA	VNO	WRI
	WY							
Utilities	AEP	ATO	BKH	CMS	CNP	D	DTE	DUK
	ED	EIX	ETR	EVRG	EXC	LNT	NEE	NFG
	NI	NJR	OGE	PEG	PNM	PNW	PPL	SJW
	SO	SWX	UGI	WEC	XEL			



## A.5 Experiment 1: relative performance

See Table 4.

**Table 4** Computational performance of ADMM KKT, ADMM Unroll, Optnet and SCS relative to ADMM FP for various problem sizes,  $d_z$ , and stopping tolerances. Batch size = 128

		ADMM FP	ADMM KKT	ADMM Unroll	OptNet	SCS
$\epsilon = 10^{-1} d_z = 10$	Backward	$1.0 \times$	$0.8 \times$	$0.4 \times$	$0.8 \times$	$0.7 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.0 \times$	$1.4 \times$	$0.9 \times$
	Total	$1.0 \times$	$0.9 \times$	$0.8 \times$	$1.3 \times$	$0.9 \times$
$\epsilon = 10^{-3} d_z = 10$	Backward	$1.0 \times$	$0.8 \times$	$1.0 \times$	$0.6 \times$	$0.7 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.0 \times$	$1.0 \times$	$0.4 \times$
	Total	$1.0 \times$	$1.0 \times$	$1.0 \times$	$0.9 \times$	$0.5 \times$
$\epsilon = 10^{-5} d_z = 10$	Backward	$1.0 \times$	$0.8 \times$	$1.9 \times$	$0.6 \times$	$0.7 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.0 \times$	$0.6 \times$	$0.2 \times$
	Total	$1.0 \times$	$1.0 \times$	$1.1 \times$	$0.6 \times$	$0.3 \times$
$\epsilon = 10^{-1} d_z = 50$	Backward	$1.0 \times$	$1.9 \times$	$0.9 \times$	$1.2 \times$	$1.9 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.0 \times$	$2.2 \times$	$2.4 \times$
	Total	$1.0 \times$	$1.2 \times$	$1.0 \times$	$2.0 \times$	$2.3 \times$
$\epsilon = 10^{-3} d_z = 50$	Backward	$1.0 \times$	$1.9 \times$	$1.8 \times$	$1.2 \times$	$1.9 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.0 \times$	$1.7 \times$	$1.3 \times$
	Total	$1.0 \times$	$1.1 \times$	$1.1 \times$	$1.6 \times$	$1.4 \times$
$\epsilon = 10^{-5} d_z = 50$	Backward	$1.0 \times$	$1.8 \times$	$2.9 \times$	$1.2 \times$	$1.9 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.0 \times$	$1.1 \times$	$0.9 \times$
	Total	$1.0 \times$	$1.0 \times$	$1.1 \times$	$1.1 \times$	$1.0 \times$
$\epsilon = 10^{-1} d_z = 100$	Backward	$1.0 \times$	$3.3 \times$	$1.7 \times$	$2.1 \times$	$3.8 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.2 \times$	$4.2 \times$	$5.7 \times$
	Total	$1.0 \times$	$1.8 \times$	$1.3 \times$	$3.5 \times$	$5.1 \times$
$\epsilon = 10^{-3} d_z = 100$	Backward	$1.0 \times$	$3.3 \times$	$3.0 \times$	$2.1 \times$	$3.8 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.1 \times$	$3.1 \times$	$3.5 \times$
	Total	$1.0 \times$	$1.5 \times$	$1.5 \times$	$2.9 \times$	$3.6 \times$
$\epsilon = 10^{-5} d_z = 100$	Backward	$1.0 \times$	$3.2 \times$	$4.5 \times$	$2.3 \times$	$3.8 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.1 \times$	$2.8 \times$	$2.6 \times$
	Total	$1.0 \times$	$1.3 \times$	$1.6 \times$	$2.7 \times$	$2.8 \times$
$\epsilon = 10^{-1} d_z = 250$	Backward	$1.0 \times$	$6.9 \times$	$1.6 \times$	$3.7 \times$	$7.4 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.0 \times$	$13.1 \times$	$12.9 \times$
	Total	$1.0 \times$	$3.6 \times$	$1.3 \times$	$9.0 \times$	$10.5 \times$
$\epsilon = 10^{-3} d_z = 250$	Backward	$1.0 \times$	$7.0 \times$	$3.6 \times$	$3.6 \times$	$7.4 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.1 \times$	$11.3 \times$	$8.9 \times$
	Total	$1.0 \times$	$3.0 \times$	$1.9 \times$	$8.7 \times$	$8.4 \times$

**Table 4** (continued)

		ADMM FP	ADMM KKT	ADMM Unroll	OptNet	SCS
$\epsilon = 10^{-5} d_z = 250$	Backward	$1.0 \times$	$6.9 \times$	$6.3 \times$	$4.5 \times$	$7.7 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.1 \times$	$11.1 \times$	$6.7 \times$
	Total	$1.0 \times$	$2.4 \times$	$2.3 \times$	$9.5 \times$	$7.0 \times$
$\epsilon = 10^{-1} d_z = 500$	Backward	$1.0 \times$	$8.6 \times$	$1.4 \times$	$3.9 \times$	$9.0 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.0 \times$	$18.9 \times$	$15.8 \times$
	Total	$1.0 \times$	$5.1 \times$	$1.2 \times$	$10.8 \times$	$12.2 \times$
$\epsilon = 10^{-3} d_z = 500$	Backward	$1.0 \times$	$8.5 \times$	$2.6 \times$	$3.6 \times$	$9.0 \times$
	Forward	$1.0 \times$	$0.9 \times$	$0.9 \times$	$17.5 \times$	$11.1 \times$
	Total	$1.0 \times$	$4.3 \times$	$1.7 \times$	$11.3 \times$	$10.2 \times$
$\epsilon = 10^{-5} d_z = 500$	Backward	$1.0 \times$	$8.6 \times$	$8.2 \times$	$4.1 \times$	$9.0 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.0 \times$	$17.4 \times$	$6.0 \times$
	Total	$1.0 \times$	$3.0 \times$	$2.9 \times$	$13.9 \times$	$6.8 \times$
$\epsilon = 10^{-1} d_z = 1000$	Backward	$1.0 \times$	$13.5 \times$	$1.1 \times$	$6.6 \times$	$15.0 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.0 \times$	$23.9 \times$	$16.4 \times$
	Total	$1.0 \times$	$7.3 \times$	$1.0 \times$	$15.1 \times$	$15.7 \times$
$\epsilon = 10^{-3} d_z = 1000$	Backward	$1.0 \times$	$13.4 \times$	$2.1 \times$	$6.6 \times$	$15.0 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.0 \times$	$27.3 \times$	$14.5 \times$
	Total	$1.0 \times$	$6.8 \times$	$1.5 \times$	$17.6 \times$	$14.7 \times$
$\epsilon = 10^{-5} d_z = 1000$	Backward	$1.0 \times$	$13.4 \times$	$11.3 \times$	$6.7 \times$	$14.5 \times$
	Forward	$1.0 \times$	$1.0 \times$	$1.0 \times$	$19.7 \times$	$7.4 \times$
	Total	$1.0 \times$	$4.3 \times$	$3.8 \times$	$16.2 \times$	$9.3 \times$

**Data availability** The U.S stock data used for computational experiments 3 and 4 was obtained from Quandl <https://data.nasdaq.com>. The Fama-French factor data was obtained from the Kenneth R. French data library [https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data\\_library.html](https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html).

## Declarations

**Conflict of interest** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. Agrawal, A., Brandon, A., Barratt, S., Boyd, S., Diamond, S., Kolter, J.Z.: Differentiable convex optimization layers. In: Advances in Neural Information Processing Systems, volume 32, pages 9562–9574. Curran Associates, Inc., (2019)
2. Agrawal, A., Barratt, S., Boyd, S., Busseti, E., Moursi, W.M.: Differentiating through a cone program, (2019). [arXiv:1904.09043](https://arxiv.org/abs/1904.09043)
3. Amos, B., Kolter, Z.J.: Optnet: Differentiable optimization as a layer in neural networks, (2017). [arXiv:1703.00443](https://arxiv.org/abs/1703.00443)
4. Amos, B., Rodriguez, Jimenez, I.D Sacks, J., Boots, B., Kolter, J.Z: Differentiable mpc for end-to-end planning and control (2019). [arXiv:1810.13400](https://arxiv.org/abs/1810.13400)

5. Anderson, D.G.M.: Iterative procedures for nonlinear integral equations. *J. ACM* **12**, 547–560 (1965)
6. Black, F., Litterman, R.: Asset allocation combining investor views with market equilibrium. *J. Fixed Income* **1**(2), 7–18 (1991)
7. Blondel, M., Berthet, Q., Cuturi, M., Frostig, R., Hoyer, S., Llinares-Lopez, F., Pedregosa, F., Vert, J.-P.: Efficient and modular implicit differentiation, (2021). [arXiv:2105.15183](https://arxiv.org/abs/2105.15183)
8. Boyd, S., Vandenberghe, L.: *Convex Optimization*. Cambridge University Press, Cambridge (2004). <https://doi.org/10.1017/CBO9780511804441>
9. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.* **3**, 1–122 (2011). <https://doi.org/10.1561/22000000016>
10. Busseti, E., Moursi, W.M., Boyd, S.: Solution refinement at regular points of conic problems. *Comput. Optim. Appl.* **74**(3), 627–643 (2019). <https://doi.org/10.1007/s10589-019-00122-2>
11. Butler, A., Kwon, R.: Covariance estimation for risk-based portfolio optimization: an integrated approach. *J. Risk*, **24**(2), (2021)
12. Butler, A., Kwon, R.H.: Integrating prediction in mean-variance portfolio optimization, (2021). [arXiv:2102.09287](https://arxiv.org/abs/2102.09287)
13. Cornuejols, G., Tutuncu, R.: *Optimization Methods in Finance*. Cambridge University Press, Cambridge (2007). <https://doi.org/10.1017/CBO9780511753886>
14. Diamond, S., Sitzmann, V., Heide, F., Wetzstein, G.: Unrolled optimization with deep priors, (2017). [arXiv:1705.08041](https://arxiv.org/abs/1705.08041)
15. Domke, J.: Generic methods for optimization-based modeling. In: Lawrence, Neil D., Girolami, M. (Eds), *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 318–326, La Palma, Canary Islands, 21–23 Apr (2012). PMLR. URL <https://proceedings.mlr.press/v22/domke12.html>
16. Dontchev, A., Rockafellar, R.: *Implicit Functions and Solution Mappings: A View from Variational Analysis*. Springer, New York (2009)
17. Donti, P., Amos, B., Zico Kolter, J.: Task-based end-to-end model learning in stochastic optimization. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) *Advances in Neural Information Processing Systems*, vol. 30, pp. 5484–5494. Curran Associates Inc., New York (2017)
18. Fama, Eugene F., French, K.R.: A five-factor asset pricing model. *J. Financ. Econ.* **116**(1), 1–22 (2015)
19. Gabay, D., Mercier, B.: A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Comput. Math. Appl.* **2**, 17–40 (1976)
20. Glowinski, R., Marroco, A.: Sur l’approximation, par éléments finis d’ordre un, et la résolution, par pénalisation-dualité d’une classe de problèmes de dirichlet non linéaires. *ESAIM: Math. Model. Numer. Anal. Modélisation Mathématique et Analyse Numérique* **9**(2), 41–76 (1975)
21. Goldfarb, D., Liu, S.: An  $\mathcal{O}(n^3)$  primal interior point algorithm for convex quadratic programming. *Math. Program.* **49**, 325–340 (1991)
22. Ho, M., Sun, Z., Xin, J.: Weighted elastic net penalized mean-variance portfolio design and computation. *SIAM J. Financ. Math.* **6**(1), 1220–1244 (2015)
23. Kim, S.-J., Koh, K., Lustig, M., Boyd, S., Gorinevsky, D.: An interior-point method for large-scale  $\ell_1$ -regularized least squares. *Sel. Top. Signal Process. IEEE J.* **1**, 606–617 (2008). <https://doi.org/10.1109/JSTSP.2007.910971>
24. Mahapatruni, R.S.G., Gray, A.: Cake: convex adaptive kernel density estimation. In: Gordon, G., Dunson, D., Dudik, M. (Eds), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 498–506, Fort Lauderdale, FL, USA, 11–13 (Apr 2011). PMLR. URL <https://proceedings.mlr.press/v15/mahapatruni11a.html>
25. Mandi, J., Guns, T.: Interior point solving for lp-based prediction+optimisation, (2020). [arXiv:2010.13943](https://arxiv.org/abs/2010.13943)
26. Mandi, J., Demirovic, E., Stuckey, P.J., Guns, T.: Smart predict-and-optimize for hard combinatorial optimization problems, (2019). [arXiv:1911.10092](https://arxiv.org/abs/1911.10092)
27. Markowitz, H.: Portfolio selection. *J. Financ.* **7**(1), 77–91 (1952)
28. Michaud, R., Michaud, R.: Estimation error and portfolio optimization: a resampling solution. *J. Invest. Manag.* **6**(1), 8–28 (2008)

29. O'Donoghue, B., Chu, E., Parikh, N., Boyd, S.: Conic optimization via operator splitting and homogeneous self-dual embedding, (2016)
30. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986)
31. Schubiger, M., Banjac, G., Lygeros, J.: Gpu acceleration of admm for large-scale quadratic programming. *J. Parallel Distrib. Comput.* **144**, 55–67 (2020)
32. Sotasakis, P., Menounou, K., Patrinos, P.: Superscs: fast and accurate large-scale conic optimization, (2019). [arXiv:1903.06477](https://arxiv.org/abs/1903.06477)
33. Stellato, B., Banjac, G., Goulart, P., Bemporad, A., Boyd, S.: OSQP: An operator splitting solver for quadratic programs. *Math. Program. Comput.* **12**(4), 637–672 (2020)
34. Tibshirani, Robert: Regression shrinkage and selection via the lasso. *J. R. Stat. Soc.* **58**(1), 267 (1996)
35. Tikhonov, A.N.: Solution of incorrectly formulated problems and the regularization method. *Soviet Math.* pp 1035–1038 (1963)
36. Uysal, A.S, Li, X., Mulvey, J.M.: End-to-end risk budgeting portfolio optimization with neural networks, (2021). [arXiv:2107.04636](https://arxiv.org/abs/2107.04636)
37. Walker, H., Ni, P.: Anderson acceleration for fixed-point iterations. *SIAM J. Numer. Anal.* **49**, 1715–1735 (2011). <https://doi.org/10.2307/23074353>
38. Xie, X., Wu, J., Zhong, Z., Liu, G., Lin, Z.: Differentiable linearized ADMM, (2019). [arXiv:1905.06179](https://arxiv.org/abs/1905.06179)
39. Yang, Y., Sun, J., Li, H., Xu, Z.: Admm-net: A deep learning approach for compressive sensing MRI (2017). [arXiv:1705.06869](https://arxiv.org/abs/1705.06869)

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.