KKT-INFORMED NEURAL NETWORK

A PARALLEL SOLVER FOR PARAMETRIC CONVEX OPTIMIZATION PROBLEMS

A PREPRINT

Carmine Delle Femine
Department of Data Intelligence for Energy and Industrial Processes
Vicomtech Foundation
Donostia-San Sebastián, 20009
cdellefemine@vicomtech.org

September 11, 2024

ABSTRACT

A neural network-based approach for solving parametric convex optimization problems is presented, where the network estimates the optimal points given a batch of input parameters. The network is trained by penalizing violations of the Karush-Kuhn-Tucker (KKT) conditions, ensuring that its predictions adhere to these optimality criteria. Additionally, since the bounds of the parameter space are known, training batches can be randomly generated without requiring external data. This method trades guaranteed optimality for significant improvements in speed, enabling parallel solving of a class of optimization problems.

Keywords Optimization • Parametric Optimization • Convex Optimization • Karush-Kuhn-Tucker (KKT) Conditions • Neural Networks

1 Introduction

Solving convex optimization problems is essential across numerous fields, including optimal control, logistics, and finance. In many scenarios, such as the development of surrogate models, there is a need to solve a large set of related optimization problems defined by varying parameters. Achieving fast solutions, even at the cost of strict optimality guarantees, is often a priority.

Neural networks, with their inherent ability to process data in parallel and adapt to diverse problem structures, offer a promising solution. This work introduces the KKT-Informed Neural Network (KINN), a method designed to solve parametric convex optimization problems efficiently by integrating the KKT conditions into the network's learning process. This approach enables rapid, parallel problem-solving while balancing the trade-off between speed and guaranteed optimality.

2 Background

Consider a parametric convex optimization problem in the standard form:

$$\begin{aligned} \min_{x \in \mathcal{D} \subseteq \mathbb{R}^n} \quad & f(x, \theta) \\ \text{s.t.} \quad & g_i(x, \theta) \leq 0 \quad i = 1, \ldots, m \\ & A(\theta)x - b(\theta) = 0 \end{aligned}$$

where $x \in \mathcal{D} \subseteq \mathbb{R}^n$ is the optimization variable; $\theta \in \mathcal{D}_\theta \subseteq \mathbb{R}^k$ are the parameters defining the problem; $f : \mathcal{D}_f \subseteq \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}$ is the convex cost function; $g_i : \mathcal{D}_{g_i} \subseteq \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}$ are the convex inequality constraints, $A : \mathcal{D}_\theta \to \mathbb{R}^{p \times n}$ and $b : \mathcal{D}_\theta \to \mathbb{R}^p$ defines the affine equality constraints and $\mathcal{D} = \bigcap_{i=1}^m \mathcal{D}_{g_i} \cap \mathcal{D}_f$ is the domain of the optimization problem.

Assume differentiable cost and constraints functions and that g_i satisfies Slater's condition. Given a set of parameters θ, $x^* \in \mathcal{D}$ is optimal if and only if there are λ^* and ν^* that, with x^*, satisy the Karush-Kuhn-Tucker conditions (KKT) [2]:

$$A(\theta)x^* - b(\theta) = 0 \tag{1}$$
$$g_i(x^*, \theta) \leq 0 \quad i = 1, \ldots, m \tag{2}$$
$$\lambda_i^* \geq 0 \quad i = 1, \ldots, m \tag{3}$$
$$\lambda_i^* g_i(x^*, \theta) = 0 \quad i = 1, \ldots, m \tag{4}$$
$$\nabla_{x^*} f(x^*, \theta) + \sum_{i=1}^{m} \lambda_i^* \nabla_{x^*} g_i(x^*, \theta) + A(\theta)^T \nu^* = 0 \tag{5}$$

3 Proposed method

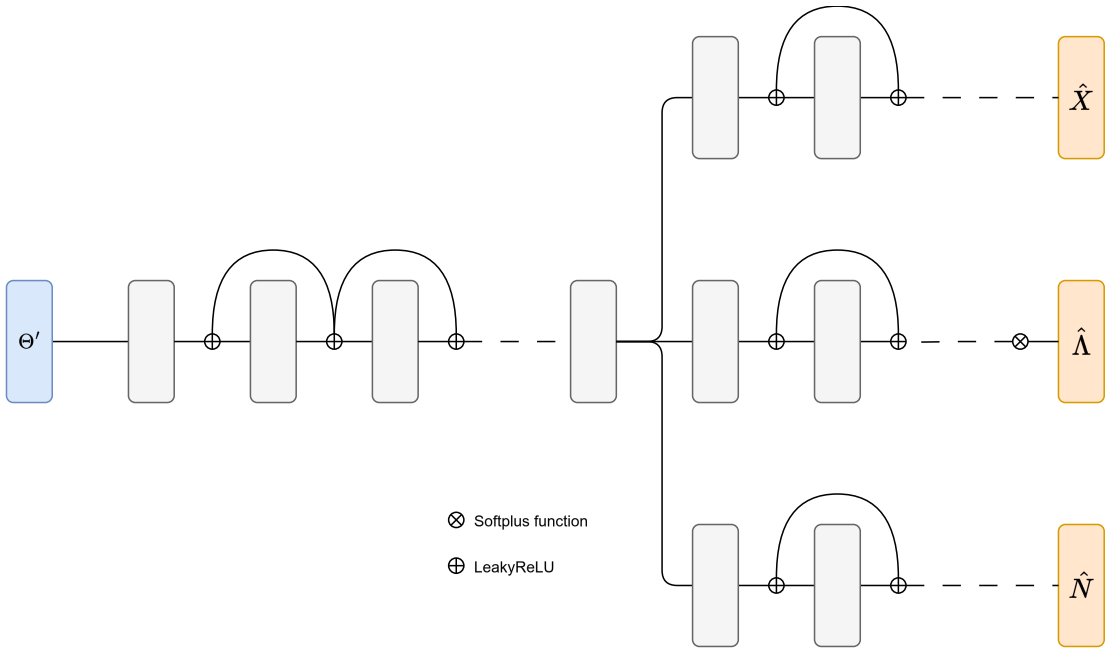


Figure 1: KINN architecture

KKT-Informed Neural Network (KINN) builds upon the principles of Physics-Informed Neural Networks (PINNs) [5], inducing compliance with Karush-Kuhn-Tucker (KKT) through a learning bias, directly coding their violation into the loss function that will be minimized in the training phase.

The objective is to processes a batch of B problem parameters $\Theta \in \mathbb{R}^{B \times k}$, $\Theta_i = \theta^{(i)}$ and have as output $\hat{X}$, $\hat{\Lambda}$, $\hat{N}$. Assuming to know the range of feasibility of each parameter, Θ will be scaled in order to obtain $\Theta' \in [-1, 1]^{B \times k}$ which will be used to fed the netowrk, aiding the convergence. Netwrok architecture is shown in Figure 1. An embedding for Θ is learned through a MLP, where each layer has a skip connection to the previous one. This embedding is then used as input to three MLPs, predicting respectively $\hat{X}$, $\hat{\Lambda}$, $\hat{N}$. A softplus function is applied to the network predicting $\hat{\Lambda}$ to ensure its feasibility.

$$[\hat{X}, \hat{\Lambda}, \hat{N}] = \text{KINN}(\Theta) \tag{6}$$
$$\hat{X} \in \mathbb{R}^{B \times n}, \quad \hat{X}_i = \hat{x}^{(i)} \tag{7}$$
$$\hat{\Lambda} \in \mathbb{R}_+^{0^{B \times m}}, \quad \hat{\Lambda}_i = \hat{\lambda}^{(i)} \tag{8}$$
$$\hat{N} \in \mathbb{R}^{B \times p}, \quad \hat{N}_i = \hat{\nu}^{(i)} \tag{9}$$

Vector-valued loss function consists of four terms that correspond to each KKT conditions:

$$\mathcal{L} = \frac{1}{B}\left[\sum_{i=1}^{B}\mathcal{L}_S^{(i)}, \sum_{i=1}^{B}\mathcal{L}_I^{(i)}, \sum_{i=1}^{B}\mathcal{L}_E^{(i)}, \sum_{i=1}^{B}\mathcal{L}_C^{(i)}\right] \tag{10}$$

where:

$$\mathcal{L}_S^{(i)} = \|\nabla_{\hat{x}^{(i)}} f(\hat{x}^{(i)}, \theta^{(i)}) + \sum\nolimits_{j=1}^{m} \hat{\lambda}_j^{(i)} \nabla_{\hat{x}^{(i)}} g_j(\hat{x}^{(i)}, \theta^{(i)}) + A(\theta^{(i)})^T \hat{\nu}^{(i)}\|_2 \tag{11}$$

$$\mathcal{L}_I^{(i)} = \|(\max(0, g_1(\hat{x}^{(i)}, \theta^{(i)})), \ldots, \max(0, g_m(\hat{x}^{(i)}, \theta^{(i)})))\|_2 \tag{12}$$

$$\mathcal{L}_E^{(i)} = \|A(\theta^{(i)})\hat{x}^{(i)} - b(\theta^{(i)})\|_2 \tag{13}$$

$$\mathcal{L}_C^{(i)} = \|(\hat{\lambda}_1^{(i)} g_1(\hat{x}^{(i)}, \theta^{(i)}), \ldots, \hat{\lambda}_m^{(i)} g_m(\hat{x}^{(i)}, \theta^{(i)}))\|_2 \tag{14}$$

$$\tag{15}$$

This vector-valued loss function is minimized through a Jacobian descent [4]. Let $\mathcal{J} \in \mathbb{R}^{P\times 4}$ the Jacobian matrix of $\mathcal{L}$, with P the number of parameters of the KINN, $\mathcal{A} : \mathbb{R}^{P\times 4} \to \mathbb{R}^P$ is called aggregator. The "direction" of the update of networks parameters will be $\mathcal{A}(\mathcal{J})$. The aggregator chosen is $\mathcal{A}_{\text{UPGrad}}$, described in [4].

4 Case studies

4.1 Projection onto a convex polythope

A renewable energy generator in a power grid is used as a test case for this approach.. The generator's active and reactive power injections (P, Q) are controllable, but they must adhere to physical constraints. As such, the desired setpoints (a_P, a_Q) must be projected onto the feasible set defined by these constraints. This problem is taken from [3].

4.1.1 Problem description

The feasible set $\mathcal{D}$ (shown in Figure 2) is defined by the physical parameters of the generator $\overline{P}_g \in \mathbb{R}_0^+$, $P_g^+ \in]0, \overline{P}_g]$, $\overline{Q}_g \in \mathbb{R}_0^+$, $Q_g^+ \in]0, \overline{Q}_g]$, characterizing the minimum and maximum possible values and the relationships between active and reactive power, and the dynamic value $P_{g,t}^{(\max)}$ which indicates the maximum power that can be generated at that time given the external conditions (e.g. wind speed, solar radiation, etc.):

$$\mathcal{D} = \{(P, Q) \in \mathbb{R}\check{s} | 0 \le P \le P_{g,t}^{(\max)}, -\overline{Q}_g \le Q \le \overline{Q}_g, Q \le \tau_g^{(1)} P + \rho_g^{(1)}, Q \ge \tau_g^{(2)} P + \rho_g^{(2)}\} \tag{16}$$

where:

$$\tau_g^{(1)} = \frac{Q_g^+ - \overline{Q}_g}{\overline{P}_g - P_g^+} \tag{17}$$

$$\rho_g^{(1)} = \overline{Q}_g - \tau_g^{(1)} P_g^+ \tag{18}$$

$$\tau_g^{(2)} = \frac{\overline{Q}_g - Q_g^+}{\overline{P}_g - P_g^+} \tag{19}$$

$$\rho_g^{(2)} = -\overline{Q}_g - \tau_g^{(2)} P_g^+ \tag{20}$$

$$\tag{21}$$

The problem could be stated in standard form as:

$$\begin{aligned} \min_{x \in \mathcal{D} \subseteq \mathbb{R}^2} \quad & \frac{1}{2}\|a - x\|_2^2 \\ \text{s.t.} \quad & Gx - h \le 0 \end{aligned}$$

with $a = (a_P, a_Q)$, $x = (P, Q)$ and:

$$G = \begin{pmatrix} -1 & 1 & 1 & 0 & 0 & -\tau_g^{(1)} & \tau_g^{(2)} \\ 0 & 0 & 0 & -1 & 1 & 1 & 1 \end{pmatrix}^T \tag{22}$$

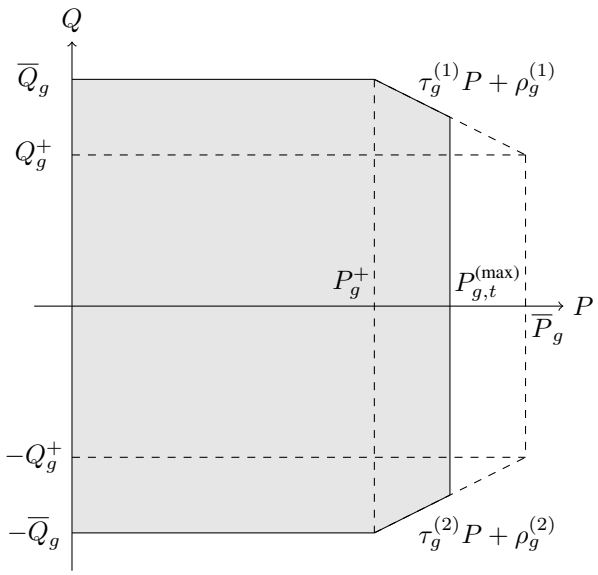


Figure 2: Feasibile set $\mathcal{D}$

$$h = \begin{pmatrix} 0 & \overline{P}_g & P_{g,t}^{(\max)} & \overline{Q}_g & \overline{Q}_g & \rho_g^{(1)} & -\rho_g^{(2)} \end{pmatrix}^T \tag{23}$$

With associated KKT conditions:

$$Gx^* - h \leq 0 \tag{24}$$
$$\lambda_i^* \geq 0 \quad i = 1, \ldots, 7 \tag{25}$$
$$G^T \lambda^* = 0 \tag{26}$$
$$(a - x^*) + G^T \lambda^* = 0 \tag{27}$$

4.1.2 Experimental results

The problem described has a two-dimensional optimization variable, seven scalar parameters and seven constraints:

$$[\hat{X}, \hat{\Lambda}] = \text{KINN}(\Theta) \tag{28}$$

with:

$$\Theta \in \mathbb{R}^{B \times 7}, \quad \Theta_i = (a_P^{(i)}, a_Q^{(i)}, \overline{P}_g^{(i)}, P_g^{+^{(i)}}, \overline{Q}_g^{(i)}, Q_g^{+^{(i)}}, P_{g,t}^{(\max)^{(i)}}) \tag{29}$$
$$\hat{X} \in \mathbb{R}^{B \times 2}, \quad \hat{X}_i = \hat{x}^{(i)} = (\hat{P}^{(i)}, \hat{Q}^{(i)}) \tag{30}$$
$$\hat{\Lambda} \in \mathbb{R}_+^{0^{B \times 7}}, \quad \hat{\Lambda}_i = \hat{\lambda}^{(i)} \tag{31}$$

The network is composed by three hidden layers of 512 neurons each, with a LeakyReLU (negative slope of 0.01) as activation function and a skip connection around each hidden layer.

At each training step, a random batch of parameters Θ was sampled:

$$a_P^{(i)} \sim U(0 \text{ p.u.}, 1 \text{ p.u.}) \tag{32}$$
$$a_Q^{(i)} \sim U(-1 \text{ p.u.}, 1 \text{ p.u.}) \tag{33}$$
$$\overline{P}_g^{(i)} \sim U(0.2 \text{ p.u.}, 0.8 \text{ p.u.}) \tag{34}$$
$$P_g^{+^{(i)}} \sim U(0 \text{ p.u.}, \overline{P}_g^{(i)}) \tag{35}$$
$$\overline{Q}_g^{(i)} \sim U(0.2 \text{ p.u.}, 0.8 \text{ p.u.}) \tag{36}$$
$$Q_g^{+^{(i)}} \sim U(0 \text{ p.u.}, \overline{Q}_g^{(i)}) \tag{37}$$
$$P_{g,t}^{(\text{max})^{(i)}} \sim U(0 \text{ p.u.}, \overline{P}_g^{(i)}) \tag{38}$$

Models parameters were update to minimize the following vector-valued loss function:

$$\mathcal{L} = \frac{1}{B}\left[\sum_{i=1}^{B} \mathcal{L}_S^{(i)}, \sum_{i=1}^{B} \mathcal{L}_I^{(i)}, \sum_{i=1}^{B} \mathcal{L}_C^{(i)}\right] \tag{39}$$

where:

$$\mathcal{L}_S^{(i)} = \|(a^{(i)} - \hat{x}^{(i)}) + G^{(i)^T}\hat{\lambda}^{(i)}\|_2 \tag{40}$$
$$\mathcal{L}_I^{(i)} = \|\max(0, G^{(i)}\hat{x} - h^{(i)})\|_2 \tag{41}$$
$$\mathcal{L}_C^{(i)} = \|G^{(i)^T}\hat{\lambda}^{(i)}\|_2 \tag{42}$$
$$\tag{43}$$

4.1.2.1 Training Training was performed with the Adam optimization algorithm with an intial learning rate of 10^{-3} and an exponential scheduler for the latter with a γ of 0.99986. An early stopping condition occurred when no progress occurred on any of the constituent terms of the loss for 5000 steps.

Finally, the training lasted 3583 steps reaching final values shown in Table 1, while the evolution along the various steps is in Figure 3.

Table 1: Final values

Loss	Value
$\mathcal{L}_S$	0.3519
$\mathcal{L}_I$	0.0020
$\mathcal{L}_C$	0.0000

4.1.2.2 Evaluation To evaluate the approach presented here, the "cvxpylayers" library [1], which implements a batched solver via multi-threading, was used as a baseline. The validation set consists of 1000 samples, generated by taking the physical parameters of the two generators present in the use case presented in [3] and, for each of them, simulating 500 random inputs (a_P, a_Q) and external condition $P_{g,t}^{(\text{max})}$. Specifically these parameters are:

$$[\overline{P}_g^{(i)}, P_g^{+^{(i)}}, \overline{Q}_g^{(i)}, Q_g^{+^{(i)}}]_1 = [0.3, 0.2, 0.3, 0.15] \tag{44}$$
$$[\overline{P}_g^{(i)}, P_g^{+^{(i)}}, \overline{Q}_g^{(i)}, Q_g^{+^{(i)}}]_2 = [0.5, 0.35, 0.5, 0.2] \tag{45}$$

The metrics for validation were the mean absolute error (MAE) and R^2. Last values are shown in Table 2, , while the evolution along the various steps is in Figure 4.

By increasing the number of points, an inference time comparison was performed on an Apple M2 Pro processor with backend for PyTorch's MPS. The difference is remarkable, about two orders of magnitude (Figure 5): with a batch size of 1000, cvxpylayers took 2.35 s as opposed to KINN's 0.06 s.

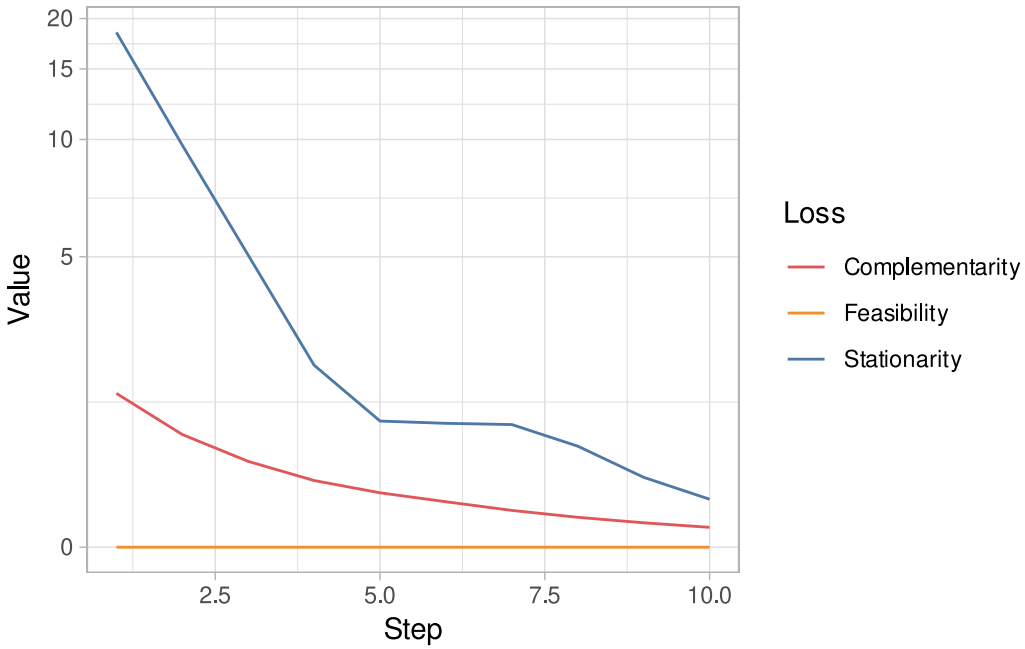
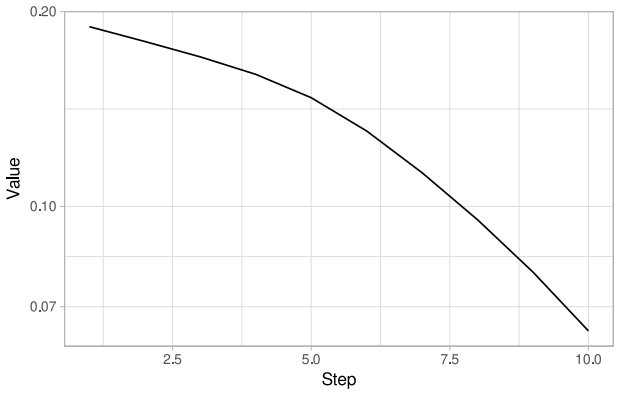


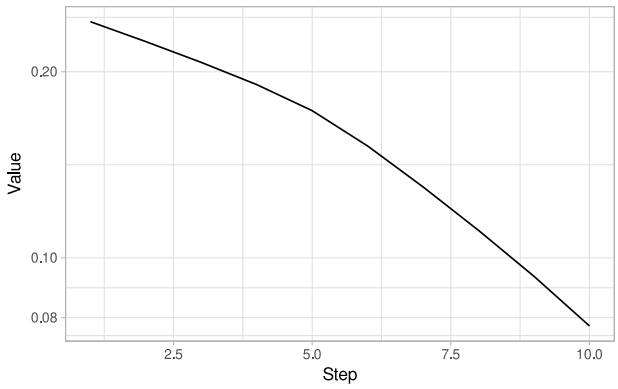
Figure 3: Loss terms during training

Table 2: Evaluation metrics

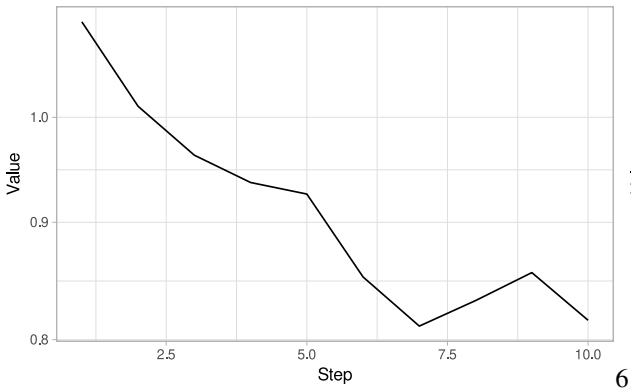
Metric	Value
MAE	0.0056 p.u.
R^2	0.9972



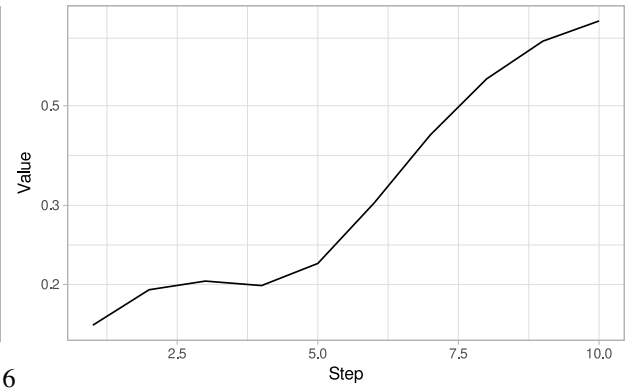
(a) MAE



(b) RMSE



(c) MAPE



(d) R^2

Figure 4: Evaluation metrics

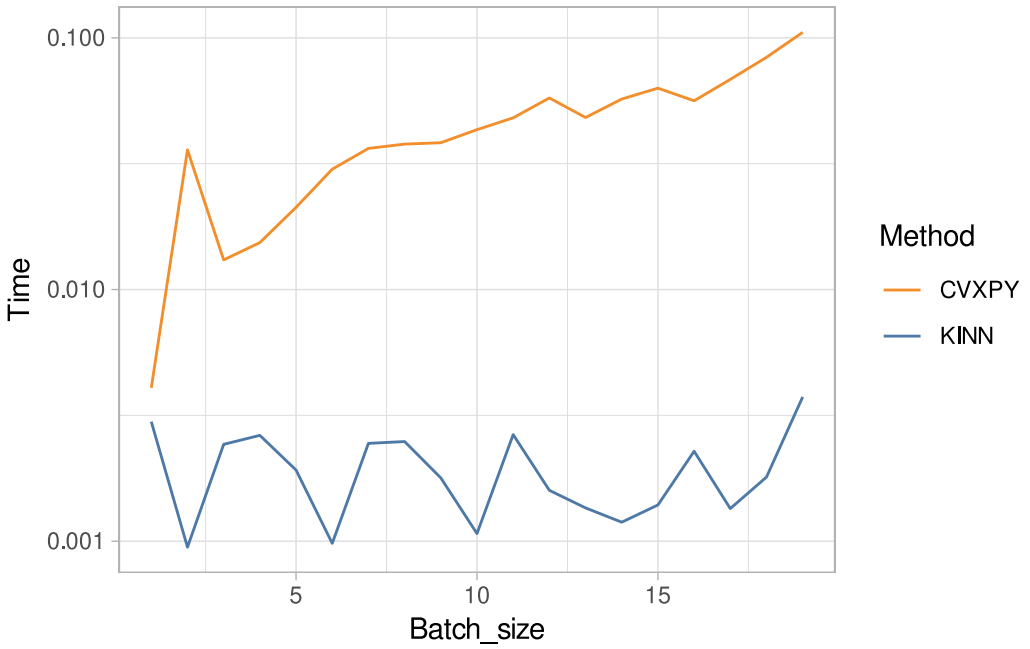


Figure 5: Computation time comparison

References

[1] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and Zico Kolter. Differentiable Convex Optimization Layers. URL https://arxiv.org/abs/1910.12430v1.

[2] Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press. ISBN 978-0-521-83378-3.

[3] Robin Henry and Damien Ernst. Gym-ANM: Reinforcement learning environments for active network management tasks in electricity distribution systems. 5:100092. ISSN 2666-5468. doi: 10.1016/j.egyai.2021.100092. URL https://www.sciencedirect.com/science/article/pii/S266654682100046X.

[4] Pierre Quinton and Valérian Rey. Jacobian Descent for Multi-Objective Optimization. URL https://arxiv.org/abs/2406.16232v1.

[5] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. 378:686–707. ISSN 0021-9991. doi: 10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.