[3] Brown DB, Zhang J (2022) On the strength of relaxations of weakly coupled stochastic dynamic programs. *Operations Research (articles in advance)* .

[4] de Farias DP, Van Roy B (2003) The linear programming approach to approximate dynamic programming. *Operations Research* 51(6):850–865.

[5] Nadarajah S, Cire AA (2025) Self-adapting network relaxations for weakly coupled Markov decision processes. *Management Science* 71(2):1779–1802.

[6] Whittle P (1988) Restless bandits: Activity allocation in a changing world. *Journal of Applied Probability* 25(A):287–298.

Research Highlight: A Tutorial on Branch-and-Bound Performance Estimation Programming

by Shuvomoy Das Gupta[1]
[1]Columbia University. sd3871@columbia.edu

This article serves as a step-by-step, hands-on tutorial for the Branch-and-Bound Performance Estimation Programming (BnB-PEP) methodology developed in the paper [8], which I co-authored with Professor Bart Van Parys and Professor Ernest Ryu during my PhD at the MIT Operations Research Center. I am honored to have received the 2024 INFORMS Computing Society (ICS) Student Paper Award for this paper, and I would like to thank the award committee members, Beste Basciftci, Austin Buchanan, Leonardo Lozano, and Hamed Rahimian, for recognizing the value of this work. Additionally, I thank Hamed Rahimian for his invitation to share this research with the ICS community through this tutorial.

1 Introduction

Need for faster optimization methods. Optimization problems pervade engineering, operations research, and machine learning, and as such, finding the provably fastest algorithm to solve them is of paramount importance. The efficiency of an optimization method can make a significant difference in the outcome of a problem, as well as the feasibility of solving it in practical applications. The ability to efficiently solve large-scale optimization problems has a direct impact on a wide range of areas, including operations research, management science, machine learning, and many others. In addition, optimizing the speed of an optimization method has far-reaching implications in terms of computational resources, cost-effectiveness, and energy consumption.

Importance of finding optimal first-order methods. For these aforementioned reasons, researchers in optimization continue to seek and develop faster methods with better convergence rates, lower computational complexity, and enhanced robustness. Among various types of optimization methods, first-order methods (FOMs), i.e., methods that rely solely on gradient or subgradient information are the most commonly used type of optimization algorithm today given their efficacy in solving high-dimensional problems, compatibility with large-scale datasets, and applicability to machine learning. The speed of an FOM depends on its stepsizes (also known as learning rates in machine learning). In this context, finding the optimal learning rates and the provably fastest FOM is of great importance. While other types of optimization methods can offer faster convergence in certain situations, they are computationally expensive and often impractical for large-scale optimization and machine learning. In contrast, FOMs such as gradient descent [6], accelerated gradient descent [20], heavy ball method [22], and their many variants can handle large-scale data sets and are amenable to distributed computing, making them well-suited for machine learning applications [2, 5, 11, 23]. As a result, improving the speed and efficiency of FOMs can have a significant impact on the feasibility and scalability of machine learning algorithms, as well as their ability to handle real-world applications in a timely and cost-effective manner.

Recently, renewed vitality was injected into this classical line of research by the emergence of computer-assisted methodologies following the Performance Estimation Problem (PEP) framework developed in [10, 28, 29]. Using the PEP framework, the celebrated accelerated gradient method by Nesterov was improved by a constant factor in [15, 27, 30], and entirely novel acceleration mechanisms, distinct from Nesterov's, have been discovered [14, 16, 17, 32]. These computer-assisted methodologies pose the problem of analyzing an efficient method as a convex semidefinite program, and the convexity provides certain algorithmic guarantees. However, the convexity in the formulation simultaneously serves as a limitation. The aforementioned works

presented several ingenious changes of variables, relaxations, and reformulations to retain convexity, but such efforts cover only a handful of setups. When these techniques do not apply, the prior methodologies become inapplicable.

Contribution of BnB-PEP. The Branch-and-Bound Performance Estimation Programming (BnB-PEP) proposed in [8] is a flexible methodology for constructing optimal first-order methods for convex and nonconvex optimization in a tractable and unified manner. In this methodology, we formulate the problem of finding the optimal FOM as a nonconvex yet practically tractable quadratically constrained quadratic problem (QCQP), and then apply a customized spatial branch-and-bound algorithm (open-source implementation available at: `https://github.com/Shuvomoy/BnB-PEP-code`) to solve such QCQPs to certifiable global optimality in a practical time scale. The customization speeds up the branch-and-bound algorithm, compared to the latest off-the-shelf solvers by orders of magnitude, reducing runtimes from hours to seconds and weeks to minutes. The BnB-PEP methodology is then applied to discover optimal FOMs for several practically relevant convex and nonconvex setups and obtain FOMs with bounds that improve upon prior state-of-the-art results. Furthermore, [8] demonstrates that BnB-PEP can also systematically generate analytical convergence proofs along with the optimized methods in the potential function setup.

Goal of this article. The objective of this article is to present a concise hands-on tutorial introducing the fundamental steps of the BnB-PEP methodology via a specific illustrative example. For this particular setup, we also provide a short prototype implementation in the `Julia` programming language that can be executed interactively by the reader in a single session, starting from scratch. This short prototype code can be used to compute an optimized FOM, certified to be locally optimal, and allows readers to quickly gain initial familiarity with the underlying concepts. For readers seeking a complete implementation of the full BnB-PEP algorithm—which computes a certifiably globally optimal FOM but involves significantly more development effort—we refer them to [8] and the associated open-source code.

Organization. This article is organized as follows. In Section 2, we provide the necessary background and describe the abstract optimization problem that BnB-PEP aims to solve to find the optimal FOM for a given problem setup. In Section 3, we discuss the steps to transform the abstract optimization problem into a nonconvex but practically tractable QCQP through a concrete example. In Section 4, we present a short prototype implementation of BnB-PEP in `Julia` that can quickly compute optimized stepsizes for the example under consideration. Finally, in Section 5, we present concluding remarks and provide references for those interested in learning about the topic in more detail.

2 Background and problem setup

Notation and notions. Write $\mathbb{R}^d$ for the underlying Euclidean space. Write $\langle \cdot \mid \cdot \rangle$ and $\|\cdot\| = \sqrt{\langle \cdot \mid \cdot \rangle}$ to denote the standard inner product and norm on $\mathbb{R}^d$, respectively. For $a, b \in \mathbb{N}$, denote $[a : b] = \{a, a+1, a+2, \ldots, b-1, b\}$. Write $\mathbb{R}^{m\times n}$ for the set of $m \times n$ matrices, $\mathbb{S}^n$ for the set of $n \times n$ symmetric matrices, and $\mathbb{S}^n_+$ for the set of $n \times n$ positive-semidefinite matrices. We follow standard convex-analytical definitions [4, 21, 24, 25]. A set $S \subseteq \mathbb{R}^d$ is *a convex set* if for any $x, y \in S$ and $\theta \in [0, 1]$, we have $\theta x + (1-\theta)y \in S$. A function $f \colon \mathbb{R}^d \to \mathbb{R}$ is *a convex function* if $f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$ for all $x, y \in \mathbb{R}^d$ and $\theta \in (0, 1)$. The *abstract subdifferential* of $f \colon \mathbb{R}^d \to \mathbb{R}$ at x, denoted by $\partial f(x)$, is defined to satisfy the following properties [1]: (i) If f is convex, then the abstract subdifferential is the usual convex subdifferential, i.e., $\partial f(x) = \{g \in \mathbb{R}^n \mid f(y) \geq f(x) + \langle g \mid y - x\rangle, \forall\, y \in \mathbb{R}^n\}$, (ii) If f is continuously differentiable at x, then its abstract subdifferential at x just contains the gradient $\nabla f(x)$, i.e., $\partial f(x) = \{\nabla f(x)\}$, (iii) If f attains a local minimum at x, then $0 \in \partial f(x)$, (iv) For all $y \in \mathbb{R}^d$ and $\beta \in \mathbb{R}$, $\partial\left(f(\cdot) + \frac{\beta}{2}\|\cdot - y\|^2\right) = \partial f(\cdot) + \beta(\cdot - y)$. The Clarke–Rockafellar subdifferential [7, §1.2] , Mordukhovich subdifferential [19, §1.3], and Fréchet subdifferential [3, page 132] are all instances of the abstract subdifferential [1, page 70]. Whenever we say *subdifferential* in this paper, we are referring to the abstract subdifferential. Because our analyses use only the properties of the abstract subdifferential, our results apply to all instances of the abstract subdifferential. We write $f'(x)$ to denote an element of $\partial f(x)$.

Optimization problem to be solved. In large-scale optimization and machine learning, the central problem is to find an efficient method to solve the problem

$$\underset{x\in\mathbb{R}^d}{\text{minimize}} \quad f(x), \tag{$\mathcal{P}$}$$

where $f : \mathbb{R}^d \to [-\infty, \infty]$ is the function to be minimized over the decision variable $x \in \mathbb{R}^d$. Because we are considering a large-scale setup, we will assume that the problem dimension d is much larger than the number of iterations of the method; we call this the *large-scale assumption*. We will further assume that f has a global minimizer $x_\star$ (not necessarily unique), although our methodology can be extended even when there is no minimizer. We assume that we know some properties of the function f that are independent of the problem data, i.e., f belongs to some function class that we can characterize mathematically, e.g., ($\mathcal{P}$) could represent smooth convex optimization problems, nonsmooth nonconvex optimization problems, and so on. Our goal is to design the provably fastest FOM to solve ($\mathcal{P}$) irrespective of the problem data; in other words, we are seeking the optimal FOM over the function class under consideration. We next define (i) the function class, (ii) compact representation of FOMs, (iii) performance measure, initial condition, and worst-case performance of an FOM that are required to evaluate its progress towards an optimal solution, and (iv) the optimal FOM.

Function class $\mathcal{F}$. When designing an optimization method, we need to recognize what are the properties of the function f that the method can exploit. In other words, it is necessary to characterize what function class $\mathcal{F}$ the function f belongs to. BnB-PEP is applicable to any function class that is *quadratically representable* (which strictly contains all the common function classes considered in classical optimization). We say $\mathcal{F}$ is *quadratically representable* if the membership $f \in \mathcal{F}$ is defined by an inequality of the form: $c_0 f(y) \geq c_1 f(x) + q(x, y, u, v)$, where $u \in \partial f(x)$, $v \in \partial f(y)$, and $x, y \in \mathbb{R}^d$, where

$$\begin{aligned} q(x,y,u,v) \triangleq\, & c_2 \langle x \mid x\rangle + c_3 \langle y \mid y\rangle + c_4 \langle u \mid u\rangle + c_5 \langle v \mid v\rangle + c_6 \langle x \mid y\rangle + c_7 \langle x \mid u\rangle \\ & + c_8 \langle x \mid v\rangle + c_9 \langle y \mid u\rangle + c_{10} \langle y \mid v\rangle + c_{11} \langle u \mid v\rangle + c_{12}, \end{aligned} \tag{1}$$

with $c_i \in \mathbb{R}$ for $i \in [0:12]$ along with an optional inequality of the form $\|u\| \leq M$, for $u \in \partial f(x)$, $x \in \mathbb{R}^d$ for some $M > 0$. Many commonly studied function classes in optimization literature are quadratically representable. For example,

- The class of L-smooth μ-strongly ($0 \leq \mu \leq L \leq \infty$) convex functions $\mathcal{F}_{\mu,L}$ satisfies [27, Theorem 1]:

$$\begin{aligned} f(y) \geq f(x) + \langle \nabla f(x) \mid y - x\rangle + \frac{1}{2(1-\mu/L)}\Big(& (1/L)\|\nabla f(x) - \nabla f(y)\|^2 \\ & + \mu\|x-y\|^2 - 2(\mu/L)\langle \nabla f(x) - \nabla f(y) \mid x - y\rangle \Big), \quad \forall x, y \in \mathbb{R}^d. \end{aligned} \tag{2}$$

- The class of nonsmooth nonconvex functions that are regularizable is known as ρ-weakly convex functions with L-bounded subgradients $\mathcal{W}_{\rho,L}$ satisfies [9, Lemma 2.1]:

$$\begin{aligned} & f(y) \geq f(x) + \langle u \mid y - x\rangle - \frac{\rho}{2}\|x-y\|^2, \quad \forall u \in \partial f(x),\, x, y \in \mathbb{R}^d, \\ & \|u\| \leq L, \quad \forall\, u \in \partial f(x),\, x \in \mathbb{R}^d. \end{aligned} \tag{3}$$

Compact representation of a general FOM. We now discuss the class of FOMs over which we seek the optimal FOM. To that goal, we work with the following definition of a FOM in a general form. Any FOM with N iterates in this general form can be represented with the following compact form. It takes in a function f and a starting point $x_0 \in \mathbb{R}^d$ as input and produces its iterates with:

$$x_i = x_{i-1} - \sum_{j=0}^{i-1} s_{i,j} f'(x_j) \tag{FOM}$$

for $i \in [1:N]$, where $f'(x_j) \in \partial f(x_j)$ is a subgradient of f at x_j for $j \in [0:N-1]$. We call $\{s_{i,j}\}_{0 \leq j < i \leq N}$ the learning rates or stepsizes of the method; these learning rates control the speed of the method. Different learning rates lead to different FOMs: popular FOMs such as gradient descent [5], Nesterov's accelerated gradient descent [20], Polyak's heavy ball method [22], all can be equivalently represented in the form (FOM) using different choices of the learning rates [26]. Write $\mathcal{M}_N$ to denote the set of all FOMs with N steps.

Performance measure $\mathcal{E}$ and initial condition $\mathcal{C}$. For notational convenience, define the index set $I_N^\star = \{0, 1, \ldots, N, \star\}$. Throughout this paper, we will use $\star$ as the index corresponding to the optimal point. Write $\mathcal{E}$ to denote the performance measure that evaluates a method $M \in \mathcal{M}_N$ on a specific function $f \in \mathcal{F}$ with a starting point x_0. We require that $\mathcal{E}$ depends only on iterates $\{x_0, \ldots, x_N\}$, a globally optimal solution $x_\star$ to ($\mathcal{P}$), and the function values and (sub)gradients of f at the points $x_0, x_1, \ldots, x_N, x_\star$. Commonly considered performance measures include $\mathcal{E} := f(x_N) - f(x_\star)$ or $\mathcal{E} := \|\nabla f(x_N)\|^2$ when f is differentiable and so on. To obtain a meaningful rate on the methods, we impose a suitable condition on the initial iterate x_0, which we abstractly express as $\mathcal{C} \leq 0$. Common initial conditions are $\mathcal{C} := \|x_0 - x_\star\|^2 - R^2$ or $\mathcal{C} := f(x_0) - f(x_\star) - R^2$, where $R > 0$.

Worst-case performance. The speed of over the entire function class $\mathcal{F}$ is measured by its *worst-case performance*. The worst-case performance $\mathcal{R}$ of a given method $M \in \mathcal{M}_N$ is obtained by maximizing $\mathcal{E}$ over functions in $\mathcal{F}$ for initial condition $\mathcal{C}$, i.e.,

$$\mathcal{R}(M, \mathcal{E}, \mathcal{F}, \mathcal{C}) = \left(\begin{array}{l} \text{maximize} \quad \mathcal{E}\left(\{x_i, f'(x_i), f(x_i)\}_{i \in I_N^\star}\right) \\ \text{subject to} \\ f \in \mathcal{F}, \\ x_\star \text{ is a globally optimal solution to } (\mathcal{P}), \\ \{x_i\}_{i \in [1:N]} \text{ is generated by } M \text{ with initial point } x_0, \\ \mathcal{C}\left(\{x_i, f'(x_i), f(x_i)\}_{i \in I_N^\star}\right) \leq 0, \end{array} \right) \tag{$\mathcal{O}^{\text{inner}}$}$$

where f, $x_0, \ldots, x_N$, and $x_\star$ are the decision variables.

Optimal FOM. The optimal FOM $M_N^\star \in \mathcal{M}_N$ for a given performance measure $\mathcal{E}$ over function class $\mathcal{F}$ subject to the initial condition $\mathcal{C}$ is a solution to the following minimax optimization problem:

$$\mathcal{R}^\star(\mathcal{M}_N, \mathcal{E}, \mathcal{F}, \mathcal{C}) = \underset{M \in \mathcal{M}_N}{\text{minimize}} \quad \mathcal{R}(M, \mathcal{E}, \mathcal{F}, \mathcal{C}). \tag{$\mathcal{O}^{\text{outer}}$}$$

Note that ($\mathcal{O}^{\text{outer}}$) is an infinite-dimensional and nonconvex problem and hence it is intractable in the present form. The BnB-PEP methodology formulates ($\mathcal{O}^{\text{outer}}$) as a nonconvex but practically tractable QCQP by exploiting problem-specific structures and solves it to certifiable global optimality using a spatial branch-and-bound algorithm in a practically tractable manner. In §3, we illustrate our methodology by describing it for a concrete problem instance.

3 Illustrative example: finding the optimal FOM for gradient minimization in strongly convex smooth problems

To demonstrate how BnB-PEP works, we pick a specific problem setup, where our goal is to find the optimal FOM for reducing the gradient of a μ-strongly convex L-smooth function, with $0 \leq \mu < L \leq \infty$, i.e., we choose the function class $\mathcal{F} := \mathcal{F}_{\mu,L}$, performance measure $\mathcal{E} := \|\nabla f(x_N)\|^2$. Furthermore, we choose the initial condition $\mathcal{C} := \|x_0 - x_\star\|^2 - R^2 \leq 0$ with $R > 0$. We assume that the stepsizes $\{s_{i,j}\}_{0 \leq j < i \leq N}$ of method M in (FOM) are nonnegative. We pick this setup specifically as an illustration of our methodology, because the resulting problem contains all the steps necessary for a complete demonstration.

3.1 Construction process of the optimal FOM

In this subsection, we discuss how to reformulate the infinite-dimensional nonconvex problem ($\mathcal{O}^{\text{outer}}$) with $\mathcal{F} := \mathcal{F}_{\mu,L}$, $\mathcal{E} := \|\nabla f(x_N)\|^2$, and $\mathcal{C} := \|x_0 - x_\star\|^2 - R^2 \leq 0$ into a tractable nonconvex QCQP In §3.1.1, we formulate the inner problem ($\mathcal{O}^{\text{inner}}$) as a convex semidefinite programming problem (SDP), follows the approach of [10, 28, 29]. Then in §3.1.2 we formulate the outer problem ($\mathcal{O}^{\text{outer}}$) as a QCQP, which is novel.

3.1.1 Formulating the inner problem ($\mathcal{O}^{\text{inner}}$) as a convex SDP

Step 1. Transform the inner optimization problem into a finite-dimensional form. For the inner problem, the method M, which is of the form (FOM), is fixed. By setting $s_{i,j} = h_{i,j}/L$, and then defining $\alpha_{i,j} = h_{i,i-1}$ for $j = i-1$ and $\alpha_{i,j} = \alpha_{i-1,j} + h_{i,j}$ for $j \in [0 : i-2]$ in (FOM), we can parameterize the FOMs in $\mathcal{M}_N$ as:

$$x_i = x_0 - (1/L) \sum_{j=0}^{i-1} \alpha_{i,j} f'(x_j), \tag{4}$$

for $i \in [1 : N]$. Because the relationship between α and h is an invertible linear system, we can compute h given α and vice versa. Next, to convert the infinite-dimensional constraint $f \in \mathcal{F}_{\mu,L}$ (that is equivalent to (2)) into a finite dimensional form, we use a discretization result [29, Theorem 4], which states that in the inner problem we can discretize f at the points seen by the method without any loss of generality. In other words, we can discretize (2) at the points $x_0, x_1, \ldots, x_N, x_\star$. Using (4), the discretization result, and finally introducing the notation $g_i \triangleq \nabla f(x_i)$ and $f_i \triangleq f(x_i)$, we can cast ($\mathcal{O}^{\text{inner}}$) as the following finite-dimensional

but nonconvex problem:

$$\mathcal{R}(M,\mathcal{E},\mathcal{F},\mathcal{C}) = \left(\begin{array}{ll} \text{maximize} & \|\nabla f(x_N)\|^2 \\ \text{subject to} & f_i \geq f_j + \langle g_j \mid x_i - x_j\rangle + \\ & \quad \frac{1}{2(1-\frac{\mu}{L})}\left(\frac{1}{L}\|g_i-g_j\|^2 + \mu\|x_i-x_j\|^2 - 2\frac{\mu}{L}\left\langle g_i-g_j \mid x_i-x_j,\right\rangle\right), \quad i,j\in I_N^\star, \\ & \nabla f(x_\star) = 0, \\ & x_i = x_0 - \sum_{j=0}^{i-1}\frac{\alpha_{i,j}}{L}\nabla f(x_j), \quad i\in[1:N], \\ & \|x_0 - x_\star\|^2 \leq R^2, \\ & x_\star = 0,\ f(x_\star) = 0, \end{array}\right) \tag{5}$$

where the decision variables are $\{x_i, g_i, f_i\}_{i\in I_N^\star} \subseteq \mathbb{R}^d\times\mathbb{R}^d\times\mathbb{R}$, where $I_N^\star = \{0,1,\dots,N,\star\}$. We set $x_\star = 0$ and $f(x_\star) = 0$, which incurs no loss of generality because any quadratically representable function class is closed and invariant under shifting variables and function values. While the problem above is finite-dimensional, the problem is still intractable as d can be arbitrarily large.

Step 2. Construct convex SDP in maximization form via Grammian formulation. Next, we formulate (5) as a convex SDP. Let $P = [x_0 \mid g_0 \mid g_1 \mid \dots \mid g_N] \in \mathbb{R}^{d\times(N+2)}$, $G = P^\top P \in \mathbb{S}_+^{N+2}$, and $F = [f_0 \mid f_1 \mid \dots \mid f_N] \in \mathbb{R}^{1\times(N+1)}$. Note that $\operatorname{rank}(G) \leq d$. Define the following notation for selecting columns and elements of P and F:

$$\begin{aligned} &\mathbf{g}_\star = 0\in\mathbb{R}^{N+2},\ \mathbf{g}_i = e_{i+2}\in\mathbb{R}^{N+2}, \quad i\in[0:N] \\ &\mathbf{x}_0 = e_1\in\mathbb{R}^{N+2},\ \mathbf{x}_\star = 0\in\mathbb{R}^{N+2},\ \mathbf{x}_i = \mathbf{x}_0 - \frac{1}{L}\sum_{j=0}^{i-1}\alpha_{i,j}\mathbf{g}_j\in\mathbb{R}^{N+2}, \quad i\in[1:N] \\ &\mathbf{f}_\star = 0\in\mathbb{R}^{N+1},\ \mathbf{f}_i = e_{i+1}\in\mathbb{R}^{N+1}, \quad i\in[0:N]. \end{aligned} \tag{6}$$

This notation is defined so that $x_i = P\mathbf{x}_i$, $g_i = P\mathbf{g}_i$, $f_i = F\mathbf{f}_i$ for $i\in I_N^\star$. Note that $\mathbf{x}_i$ depends on $\{\alpha_{i,j}\}_{j\in[0:i-1]}$ linearly for $i\in[1:N]$. Write $(\cdot\odot\cdot):\mathbb{R}^d\times\mathbb{R}^d\to\mathbb{R}^{d\times d}$ to denote the symmetric outer product, that is, for any $x,y\in\mathbb{R}^d$: $x\odot y = (xy^\top + yx^\top)/2$. For $i,j\in I_N^\star$, now define

$$\begin{aligned} &A_{i,j}(\alpha) = \mathbf{g}_j\odot(\mathbf{x}_i-\mathbf{x}_j),\ B_{i,j}(\alpha) = (\mathbf{x}_i-\mathbf{x}_j)\odot(\mathbf{x}_i-\mathbf{x}_j),\ C_{i,j} = (\mathbf{g}_i-\mathbf{g}_j)\odot(\mathbf{g}_i-\mathbf{g}_j) \\ &D_{i,j} = \mathbf{g}_i\odot\mathbf{g}_j\ E_{i,j}(\alpha) = (\mathbf{g}_i-\mathbf{g}_j)\odot(\mathbf{x}_i-\mathbf{x}_j)\ a_{i,j} = \mathbf{f}_j-\mathbf{f}_i. \end{aligned} \tag{7}$$

Note that $A_{i,j}(\alpha)$ and $E_{i,j}(\alpha)$ are affine and $B_{i,j}(\alpha)$ is quadratic in $\{\alpha_{i,j}\}_{i\in[1:N],j\in[0,i-1]}$.

Under the large-scale assumption we have $d\gg N$, hence we can assume $d\geq N+2$. As a result the $\operatorname{rank}(G)\leq d$ becomes vacuous, since $G\in\mathbb{S}_+^{(N+2)\times(N+2)}$. Using this notation, we can formulate (5) as a convex maximization SDP:

$$\mathcal{R}(M,\mathcal{E},\mathcal{F},\mathcal{C}) = \left(\begin{array}{ll} \text{maximize} & \operatorname{tr} G\left(C_{N,\star}\right) \\ \text{subject to} & \\ & Fa_{i,j} + \operatorname{tr} G\Big[A_{i,j} + \\ & \frac{1}{2(1-\frac{\mu}{L})}\left(\frac{1}{L}C_{i,j} + \mu B_{i,j} - 2\frac{\mu}{L}E_{i,j}\right)\Big] \leq 0, \quad i,j\in I_N^\star : i\neq j, \quad \triangleright \text{dual var. } \lambda_{i,j}\geq 0 \\ & -G\preceq 0, \quad \triangleright \text{dual var. } Z\succeq 0 \\ & \operatorname{tr} GB_{0,\star}\leq R^2, \quad \triangleright \text{dual var. } \nu\geq 0. \end{array}\right) \tag{8}$$

where $F\in\mathbb{R}^{N+2}$, $G\in\mathbb{S}^{N+2}$ are the decision variables. We denote the corresponding dual variables on the right hand side of the constraints for later use. Note that (8) is free from problem dimension d.

Step 3. Construct convex SDP in minimization form via duality. Next we use convex duality to formulate the SDP (8) in the previous step, originally a maximization problem, as a minimization problem. In our setup, strong duality will hold due to the existence of a Slater point in (8) [29, Theorem 6]. The dualized minimization problem to (8) is:

$$\mathcal{R}(M,\mathcal{E},\mathcal{F},\mathcal{C}) = \left(\begin{array}{ll} \text{minimize} & \nu R^2 \\ \text{subject to} & \\ & \sum_{i,j\in I_N^\star:i\neq j}\lambda_{i,j}a_{i,j} = 0, \\ & \nu B_{0,\star} - C_{N,\star} + \\ & \quad \sum_{i,j\in I_N^\star:i\neq j}\lambda_{i,j}\left[A_{i,j} + \frac{1}{2(1-\frac{\mu}{L})}\left(\frac{1}{L}C_{i,j} + \mu B_{i,j} - 2\frac{\mu}{L}E_{i,j}\right)\right] = Z, \\ & Z\succeq 0, \\ & \nu\geq 0,\ \lambda_{i,j}\geq 0, \quad i,j\in I_N^\star : i\neq j, \end{array}\right) \tag{9}$$

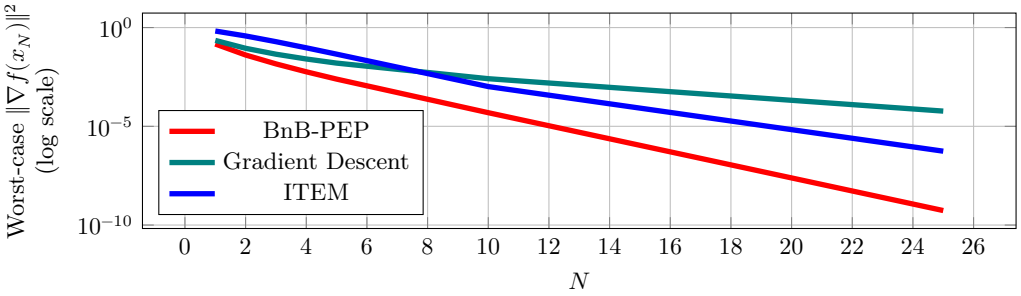

Figure 1. Comparison of the optimal method obtained by solving (10) with the BnB-PEP Algorithm against other known methods. Note that the optimal method finds a high-precision solution in less than 20 iterations.

where $\nu \in \mathbb{R}$, $\lambda = \{\lambda_{i,j}\}_{i,j \in I_N^\star : i \neq j}$, and $Z \in \mathbb{S}_+^{N+2}$ are the decision variables. We call λ, ν, and Z the *inner-dual variables*.

3.1.2 Formulating the outer problem ($\mathcal{O}^{\text{outer}}$) as a QCQP

With ($\mathcal{O}^{\text{inner}}$) formulated as a minimization problem, the outer optimization problem ($\mathcal{O}^{\text{outer}}$) becomes a joint minimization over the inner dual variables and the learning rate vector α. However, the outer minimization problem is not convex in all of the variables, even though the inner problem is. The nonconvex outer optimization problem ($\mathcal{O}^{\text{outer}}$) minimizes over α in addition to the inner dual variables of (9). We confront the nonconvexity directly by formulating ($\mathcal{O}^{\text{outer}}$) as a (nonconvex) QCQP and solving it with custom spatial branch-and-bound algorithms. We do not discard constraints or use any relaxation. To this end, we replace the semidefinite constraint with a quadratic constraint via the Cholesky factorization.

Lemma 1 ([13, Corollary 7.2.9])**.** *A matrix $Z \in \mathbb{S}^n$ is positive semidefinite if and only if it has a Cholesky factorization $PP^\top = Z$, where $P \in \mathbb{R}^{n \times n}$ is lower triangular with nonnegative diagonals.*

In raw index form, the conditions of Lemma 1, applied to the present setup, have the following equivalent representations: $P_{j,j} \geq 0$, for $j \in [1 : N+2]$, $P_{i,j} = 0$, for $1 \leq i < j \leq N+2$, and $\sum_{k=1}^{j} P_{i,k} P_{j,k} = Z_{i,j}$, for $1 \leq j \leq i \leq N+2$. Here the first two are linear constraints, and the last one is a quadratic constraint. We now formulate ($\mathcal{O}^{\text{outer}}$), the problem of finding the optimal FOM, as the following QCQP:

$$\mathcal{R}^\star(\mathcal{M}_N, \mathcal{E}, \mathcal{F}, \mathcal{C}) = \left(\begin{array}{l} \text{minimize} \quad \nu R^2 \\ \text{subject to} \\ \sum_{i,j \in I_N^\star : i \neq j} \lambda_{i,j} a_{i,j} = 0, \\ \nu B_{0,\star} - C_{N,\star} + \\ \qquad \sum_{i,j \in I_N^\star : i \neq j} \lambda_{i,j} \left[A_{i,j} + \frac{1}{2(1-\frac{\mu}{L})} \left(\frac{1}{L} C_{i,j} + \mu \Theta_{i,j} - 2\frac{\mu}{L} E_{i,j} \right) \right] = Z, \\ P \text{ is lower triangular with nonnegative diagonals}, \\ Z = PP^\top, \\ \Theta_{i,j} = B_{i,j}, \quad i,j \in I_N^\star : i \neq j, \\ \nu \geq 0, \ \lambda_{i,j} \geq 0, \quad i,j \in I_N^\star : i \neq j, \\ \alpha_{i,j} \geq 0, \quad 0 \leq j < i \leq N, \end{array} \right) \tag{10}$$

where $\lambda = \{\lambda_{i,j}\}_{i,j \in I_N^\star : i \neq j}$, ν, Z, P, $\{\alpha_{i,j}\}_{0 \leq j < i \leq N}$, and $\{\Theta_{i,j}\}_{i,j \in I_N^\star : i \neq j}$ are the decision variables. Note that $\{\Theta_{i,j}\}_{i,j \in I_N^\star : i \neq j}$ is introduced as a separate decision variable to formulate the cubic constraints arising from $\lambda_{i,j} B_{i,j}$ terms in the second constraint as quadratic constraints.where λ, ν, Z, P, and α are the decision variables. We can solve (10) to global optimality using a custom spatial branch-and-bound algorithm such as the BnB-PEP algorithm in [8], and the resultant optimal $\alpha^\star$ will give us the optimal learning rate vector $h^\star$ which corresponds to the optimal FOM for any large-scale setup $d \geq N+2$. Furthermore, due to the scale invariance discussed in [29, §3.5], it suffices to solve the nonconvex QCQP for $L = 1$ and $R = 1$ and find the corresponding optimal learning rate $\alpha^\star$ (or $h^\star$) and the associated optimal worst-case performance measure $\|\nabla f(x_N)\|^2$. More specifically, for any other $L > 0$ and $R > 0$, the new optimal learning rate will be scaled as $\alpha^\star / L$ (or $h^\star / L$) with corresponding performance measure scaled as $L^2 R^2 \|\nabla f(x_N)\|^2$. Figure 1 shows how the optimal FOM computed by BnB-PEP compares with other FOMs.

Figure 2. `Julia` started for the first time

4 A short prototype implementation in `Julia` to compute optimized stepsizes

A worked example is worth thousands of words, so we present a prototype implementation in `Julia` that can compute optimized stepsizes for problem (10). The main advantage of this prototype is its simplicity and ease of implementation; we estimate that implementing this code from scratch would require less than one hour. However, its primary limitation is that it can only certify local optimality. In contrast, the full BnB-PEP algorithm in [8] employs a custom spatial branch-and-bound approach capable of solving this problem to global optimality, albeit at the expense of substantial development effort. For a comprehensive implementation of this approach, we refer the reader to our paper [8].

The steps for the prototype implementation to compute optimized stepsizes for (10) are described below.

Install `Julia`. The first step is to install `Julia`, the programming language that we will use for the prototyping. If the operating system is Windows, then we open the command prompt, type the following and press enter:

```
winget install --name Julia --id 9NJNWW8PVKMN -e -s msstore
```

If the operating system is Linux or MacOS, then we can open the terminal, type the following and press enter:

```
curl -fsSL https://install.julialang.org | sh
```

Once installed, `Julia` will be available via the command prompt or terminal by typing `julia` and then pressing enter. We should see something like Figure 2, which is called the `Julia REPL`.

Now, open the `Julia REPL` and run the following code to install some necessary packages that we will use later.

```
using Pkg;
Pkg.add(["JuMP", "Ipopt", "OffsetArrays"])
```

Here, `JuMP` is a domain-specific modeling language for mathematical optimization embedded in Julia [18], `Ipopt` is an open-source nonlinear interior point solver [31], and `OffsetArrays` is used to define arrays with arbitrary indices.

Load the `Julia` packages. In the `Julia REPL`, we run the following code step-by-step (it suffices just to copy the code, paste into `Julia REPL`, and then press enter). We load the packages that we just installed along with two helper functions that we will use later in our code.

```
# Load the packages
using JuMP, Gurobi, Ipopt, OffsetArrays

# Define e_i ∈ ℝ^n
function e_i(n, i)

    e_i_vec = zeros(n, 1)
```

```
    e_i_vec[i] = 1
    return e_i_vec

end

# This function computes a ⊙ b = 1/2(ab^⊤ + ba^⊤)
function ⊙(a,b)

    return ((a*b') .+ transpose(a*b')) ./ 2

end
```

Load the parameters. We will work with the following parameters that works as the input data for (10). Run the following code in the `Julia REPL` to load the parameter values.

```
# Parameter values
N = 2 # Iteration budget
μ = 1/10 # Strong convexity parameter
L = 1 # Smoothness parameter
R = 1 # Initial condition distance
dim_Z = N+2 # dimension of the matrix Z

# Define the index set I_N^⋆ = {⋆, 0, 1, ..., N},
# where we  use −1 as the index for ⋆, i.e., x_⋆ = x_{-1}, g_⋆ = g_{-1}, and f_⋆ = f_{-1}
I_N_star = -1:N
```

Create $\{\mathbf{x}_i, \mathbf{g}_i, \mathbf{f}_i\}_{i\in I_N^\star}$ and matrix functions. Next, run the `Julia` code to create $\{\mathbf{x}_i, \mathbf{g}_i, \mathbf{f}_i\}_{i\in I_N^\star}$ that we defined in (6) are as follows:

```
# Data generator function to create {x_i, g_i, f_i}_{i∈I_N^⋆}
# It will output bold_x, bold_g, and bold_f where for every i ∈ I_N^⋆
# bold_x[:,i] corresponds to x_i
# bold_g[:, i] corresponds to g_i
# bold_f[:, i] corresponds to f_i

function data_generator_function(N, α, μ, L)

    dim_bold_x = N+2
    dim_bold_g = N+2
    dim_bold_f = N+1
    N_pts = N+2 # number of points corresponding to [x_⋆=x_{-1} x_0 ... x_N]
    bold_x_0 = e_i(dim_bold_x, 1)
    bold_x_star = zeros(dim_bold_x, 1)

    # initialize bold_g and bold_f vectors
    # bold_g = [bold_g_{-1}=bold_g_⋆ | bold_g_0 | bold_g_1 |... | bold_g_N]
    bold_g =  OffsetArray(zeros(dim_bold_g, N_pts), 1:dim_bold_g, -1:N)

    # bold_f  = [bold_f_{-1}=bold_f_⋆ | bold_f_0 | bold_f_1 |... | bold_f_N]
    bold_f = OffsetArray(zeros(dim_bold_f, N_pts), 1:dim_bold_f, -1:N)

    # construct bold_g vectors, note that bold_g_⋆  is already constructed zero
    for k in 0:N
        bold_g[:,k] = e_i(dim_bold_g, k+2)
    end
```

```
    # construct bold_f vectors, note that bold_f_⋆ is already constructed zero
    for k in 0:N
        bold_f[:,k] = e_i(dim_bold_f, k+1)
    end

    # time to define the bold_x vectors, which requires more care
    # caution ⋆: keep in mind that this matrix bold_x is not 0 indexed yet, so while
     ↪ constructing its elements,
    # ensure to use the full formula for bold_x_i

    bold_x = [bold_x_star bold_x_0]

    # construct part of bold_x corresponding to the x iterates: x_1, ..., x_N
    for i in 1:N
        bold_x_i =  bold_x_0  - ( (1/L)*sum( α[i,j] * bold_g[:,j] for j in 0:i-1) )
        bold_x = [bold_x bold_x_i]
    end

    # make bold_x an offset array to make our life comfortable
    bold_x = OffsetArray(bold_x, 1:dim_bold_x, -1:N)

    # time to return
    return bold_x, bold_g, bold_f

end
```

The `Julia` code that defines the matrix functions in (7) is as follows.

```
A_mat(i,j,α,bold_g,bold_x) = ⊙(bold_g[:,j], bold_x[:,i]-bold_x[:,j])

B_mat(i,j,α,bold_x) = ⊙(bold_x[:,i]-bold_x[:,j], bold_x[:,i]-bold_x[:,j])

C_mat(i,j,bold_g) = ⊙(bold_g[:,i]-bold_g[:,j], bold_g[:,i]-bold_g[:,j])

D_mat(i,j,bold_g) = ⊙(bold_g[:,i], bold_g[:,j])

E_mat(i, j, α, bold_g, bold_x) = ⊙(bold_g[:,i] - bold_g[:,j], bold_x[:,i]-bold_x[:,j])

a_vec(i,j,bold_f) = bold_f[:, j] - bold_f[:, i]
```

Model (10) in JuMP. Now we are going to solve (10) step by step. `JuMP` builds problems incrementally in a `JuMP` Model object, so first, we create a model by passing the `Ipopt` optimizer to the Model function:

```
# Load the model
BnB_PEP_model =  Model(Ipopt.Optimizer)
```

Next, we define all the decision variables. Recall that our decision varaibles in (10) are $\{\lambda_{i,j}\}_{i,j\in I_N^\star: i\neq j} \geq 0$, $\nu \geq 0$, Z, P, α, and $\{\Theta_{i,j}\}_{i,j\in I_N^\star: i\neq j}$.

```
# Declare the variables λ_{i,j} for  i,j ∈ I_N^⋆, i ≠ j
struct i_j_idx
    i::Int64 # corresponds to index i
    j::Int64 # corresponds to index j
end
```

```
idx_set_λ = i_j_idx[]

for i in I_N_star
    for j in I_N_star
        if i!=j
            push!(idx_set_λ, i_j_idx(i,j))
        end
    end
end

@variable(BnB_PEP_model, λ[idx_set_λ] >= 0)

# Declare ν as a decision variable
@variable(BnB_PEP_model, ν >= 0)

# Declare    Z as a decision variable
@variable(BnB_PEP_model, Z[1:dim_Z, 1:dim_Z], Symmetric)

# Define the cholesky matrix P as a decision variable
@variable(BnB_PEP_model, P[1:dim_Z, 1:dim_Z])

# Declare the stepsize matrix α as a decision variable
@variable(BnB_PEP_model, α[i = 1:N, j= 0:i-1] >= 0)

# Declare Θ_{i,j} as decision variables for i,j ∈ I_N^⋆
Θ = BnB_PEP_model[:Θ] = reshape(
hcat([
@variable(BnB_PEP_model, [1:N+2, 1:N+2], Symmetric, base_name = "Θ[$i_j_λ]")
for i_j_λ in idx_set_λ]...), N+2, N+2, length(idx_set_λ))
```

Time to create the $\{\mathbf{x}_i, \mathbf{g}_i, \mathbf{f}_i\}_{i \in I_N^\star}$ for our problem setup by running the following code.

```
# Create the data generator function
dim_bold_x = N+2
bold_x_0 = e_i(dim_bold_x, 1)
bold_x_star = zeros(dim_bold_x, 1)
bold_x, bold_g, bold_f = data_generator_function(N, α, μ, L)
```

We are in a position to add the objective and the constraints one by one. Let us start with defining the objective νR^2 that we want to minimize.

```
# Minimize νR^2
@objective(BnB_PEP_model, Min, ν*R^2)
```

Let us now model the first constraint $\sum_{i,j \in I_N^\star : i \neq j} \lambda_{i,j} a_{i,j} = 0$.

```
# Constraint ∑_{i,j∈I_N^⋆:i≠j} λ_{i,j} a_{i,j} = 0
@constraint(BnB_PEP_model, sum(λ[i_j_λ]*a_vec(i_j_λ.i,i_j_λ.j,bold_f) for i_j_λ in idx_set_λ)
 ↪  .== 0)
```

Next, model the second constraint

$$\nu B_{0,\star} - C_{N,\star} + \sum_{i,j \in I_N^\star : i \neq j} \lambda_{i,j} \left[A_{i,j} + \frac{1}{2(1-\frac{\mu}{L})} \left(\frac{1}{L} C_{i,j} + \mu \Theta_{i,j} - 2\frac{\mu}{L} E_{i,j} \right) \right] = Z$$

as follows.

```
# A helper function
index_finder_Θ(i,j,idx_set_λ) = findfirst(isequal(i_j_idx(i,j)), idx_set_λ)

# Constraint νB_{0,⋆} − C_{N,⋆} + Σ_{i,j∈I⋆_N:i≠j} λ_{i,j} [A_{i,j} + 1/(2(1−μ/L)) (1/L C_{i,j} + μΘ_{i,j} − 2μ/L E_{i,j})] = Z
@constraint(BnB_PEP_model,
    vectorize(
        -C_mat(N, -1, bold_g) +
        ν * ⊙(bold_x_0 - bold_x_star, bold_x_0 - bold_x_star)
        + sum(λ[i_j_λ] * (
            A_mat(i_j_λ.i, i_j_λ.j, α, bold_g, bold_x)
            +
            (1 / (2 * (1 - (μ / L)))) * (
                (1 / (L)) * C_mat(i_j_λ.i, i_j_λ.j, bold_g)
                +
                (μ * Θ[:, :, index_finder_Θ(i_j_λ.i, i_j_λ.j, idx_set_λ)])
                -
                (2 * (μ / L) * E_mat(i_j_λ.i, i_j_λ.j, α, bold_g, bold_x))
            )
        ) for i_j_λ in idx_set_λ) - Z,
        SymmetricMatrixShape(dim_Z)
    ) .== 0
)
```

Now define the constraints related to P being lower triangular with nonnegative diagonal entries.

```
# P is lower-triangular, i.e., upper off-diagonal terms of P are zero
for i in 1:dim_Z
    for j in 1:dim_Z
        if i < j
            fix(P[i,j], 0; force = true)
        end
    end
end

# The diagonal components of P are non-negative
for i in 1:dim_Z
    @constraint(BnB_PEP_model, P[i,i] >= 0)
end
```

We next model the constraint $Z = PP^\top$:

```
# Constraint Z = PP^⊤
@constraint(BnB_PEP_model, vectorize(Z - (P * P'), SymmetricMatrixShape(dim_Z)) .== 0)
```

The final constraint is $\Theta_{i,j} = B_{i,j} = (\mathbf{x}_i - \mathbf{x}_j) \odot (\mathbf{x}_i - \mathbf{x}_j)$ for $i, j \in I_N^\star : i \neq j$.

```
# Constraint Θ_{i,j} = B_{i,j} = (x_i − x_j) ⊙ (x_i − x_j) for i,j ∈ I⋆_N : i ≠ j
conΘ = map(1:length(idx_set_λ)) do ℓ
    i_j_λ = idx_set_λ[ℓ]
    @constraint(BnB_PEP_model, vectorize(
    Θ[:,:,ℓ] - ⊙(bold_x[:,i_j_λ.i]-bold_x[:,i_j_λ.j], bold_x[:,i_j_λ.i]-bold_x[:,i_j_λ.j]),
    SymmetricMatrixShape(dim_bold_x)) .== 0)
end
```

Optimize the JuMP model. Okay, we have modeled the problem, time to solve it now and store the data!

```
# Optimize the model
optimize!(BnB_PEP_model)

# Store λ_opt
λ_opt = value.(λ)

# Store ν_opt
ν_opt = value.(ν)

# Store α_opt
α_opt = value.(α)

# Store Z_opt
Z_opt = value.(Z)
```

We will see an output like the following in `Julia REPL`:

```
This is Ipopt version 3.14.17, running with linear solver MUMPS 5.7.3.

Number of nonzeros in equality constraint Jacobian...:      573
Number of nonzeros in inequality constraint Jacobian.:        4
Number of nonzeros in Lagrangian Hessian.............:      198

Total number of variables............................:      156
                     variables with only lower bounds:       16
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:      143
Total number of inequality constraints...............:        4
        inequality constraints with only lower bounds:        4
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  9.9999900e-03 1.00e+00 5.00e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   :   :   :  :   :  :   :  :   :  :   :  :   :  :   :  :   :  :   :  :   :  :  :
 197  4.0944374e-02 9.62e-09 2.52e-12  -9.0 8.31e-05    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 197

                                   (scaled)                 (unscaled)
Objective...............:   4.0944374041740512e-02    4.0944374041740512e-02
Dual infeasibility......:   2.5218825085263378e-12    2.5218825085263378e-12
Constraint violation....:   9.6153423703463827e-09    9.6153423703463827e-09
Variable bound violation:   3.0147036799467637e-09    3.0147036799467637e-09
Complementarity.........:   1.4749187665451383e-09    1.4749187665451383e-09
Overall NLP error.......:   9.6153423703463827e-09    9.6153423703463827e-09


Number of objective function evaluations             = 360
Number of objective gradient evaluations             = 195
Number of equality constraint evaluations            = 360
Number of inequality constraint evaluations          = 360
Number of equality constraint Jacobian evaluations   = 200
Number of inequality constraint Jacobian evaluations = 200
Number of Lagrangian Hessian evaluations             = 197
Total seconds in IPOPT                               = 0.526
```

```
EXIT: Optimal Solution Found.
```

The optimized stepsizes turn out to be $\alpha^\star_{1,0} = 1.50177$, $\alpha^\star_{2,0} = 1.55119$, and $\alpha^\star_{2,1} = 1.50177$ with performance measure $\nu^\star = 0.0409$. In other words, it says that for the optimized stepsizes we have: $\|\nabla f(x_2)\|^2 \leq 0.0409 L^2 \|x_0 - x_\star\|^2$.

5 Conclusion

In this article, we have provided a step-by-step, hands-on tutorial on the BnB-PEP methodology proposed in [8]. However, this tutorial is meant only as an introductory overview of a much broader and deeper topic. For readers interested in gaining a comprehensive understanding of this research area, we recommend the foundational papers that established the performance estimation framework [10, 28, 29]. Another important resource is the PhD thesis of Adrien Taylor [26], which, besides being a landmark dissertation, serves as an excellent textbook-like reference on PEP due to its clarity, rigor, and readability. Lastly, for cases where the stepsizes of the FOM are already known, the open-source `Python` package `PEPit` is a very user-friendly tool for computing its worst-case performance measures [12].

References

[1] Heinz H Bauschke, Walaa M Moursi, and Xianfu Wang. Generalized monotone operators and their averaged resolvents. *Mathematical Programming*, 189:55–74, 2021.

[2] Amir Beck. *First-Order Methods in Optimization*. SIAM, 2017.

[3] Jonathan Borwein and Adrian S Lewis. *Convex Analysis and Nonlinear Optimization, 2nd Edition*. Springer, New York, 2006.

[4] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[5] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[6] Louis Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comptes Rendus de l'Académie Des Sciences*, 25(1847):536–538, 1847.

[7] Frank H Clarke. *Optimization and Nonsmooth Analysis*. SIAM, 1990.

[8] Shuvomoy Das Gupta, Bart PG Van Parys, and Ernest K Ryu. Branch-and-bound performance estimation programming: A unified methodology for constructing optimal optimization methods. *Mathematical Programming*, 204(1):567–639, 2024.

[9] Damek Davis and Dmitriy Drusvyatskiy. Stochastic model-based minimization of weakly convex functions. *SIAM Journal on Optimization*, 29(1):207–239, 2019.

[10] Yoel Drori and Marc Teboulle. Performance of first-order methods for smooth convex minimization: A novel approach. *Mathematical Programming*, 145(1-2):451–482, 2014.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT press, 2016.

[12] Baptiste Goujaud, Céline Moucer, François Glineur, Julien M Hendrickx, Adrien B Taylor, and Aymeric Dieuleveut. Pepit: computer-assisted worst-case analyses of first-order optimization methods in python. *Mathematical Programming Computation*, 16(3):337–367, 2024.

[13] Roger A Horn and Charles R Johnson. *Matrix Analysis*. Cambridge University Press, 2012.

[14] Donghwan Kim. Accelerated proximal point method for maximally monotone operators. *Mathematical Programming*, 190(1–2):57–87, 2021.

[15] Donghwan Kim and Jeffrey A Fessler. Optimized first-order methods for smooth convex minimization. *Mathematical Programming*, 159(1):81–107, 2016.

[16] Donghwan Kim and Jeffrey A Fessler. Optimizing the efficiency of first-order methods for decreasing the gradient of smooth convex functions. *Journal of Optimization Theory and Applications*, 188(1):192–219, 2021.

[17] Felix Lieder. On the convergence rate of the halpern-iteration. *Optimization Letters*, 15(2):405–418, 2021.

[18] Miles Lubin, Oscar Dowson, Joaquim Dias Garcia, Joey Huchette, Benoît Legat, and Juan Pablo Vielma. JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, 2023.

[19] Boris S Mordukhovich. *Variational Analysis and Generalized Differentiation I: Basic Theory*. Springer, 2006.

[20] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[21] Yurii Nesterov. *Lectures on Convex Optimization*, volume 137. second edition, 2018.

[22] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[23] Simon JD Prince. *Understanding Deep Learning*. MIT press, 2023.

[24] Ralph T Rockafellar and Roger J-B Wets. *Variational Analysis*. Springer Science & Business Media, 2009.

[25] Ernest K Ryu and Wotao Yin. *Large-Scale Convex Optimization via Monotone Operators*. Cambridge University Press, 2022.

[26] Adrien B Taylor. *Convex Interpolation and Performance Estimation of First-Order Methods for Convex Optimization*. PhD thesis, Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2017.

[27] Adrien B Taylor and Yoel Drori. An optimal gradient method for smooth strongly convex minimization. *Mathematical Programming*, pages 1–38, 2022.

[28] Adrien B Taylor, Julien M Hendrickx, and François Glineur. Exact worst-case performance of first-order methods for composite convex optimization. *SIAM Journal on Optimization*, 27(3):1283–1313, 2017.

[29] Adrien B Taylor, Julien M Hendrickx, and François Glineur. Smooth strongly convex interpolation and exact worst-case performance of first-order methods. *Mathematical Programming*, 161(1-2):307–345, 2017.

[30] Bryan Van Scoy, Randy A Freeman, and Kevin M Lynch. The fastest known globally convergent first-order method for minimizing strongly convex functions. *IEEE Control Systems Letters*, 2(1):49–54, 2017.

[31] Andreas Wächter and Lorenz T Biegler. Line search filter methods for nonlinear programming: Local convergence. *SIAM Journal on Optimization*, 16(1):32–48, 2005.

[32] TaeHo Yoon and Ernest K Ryu. Accelerated algorithms for smooth convex-concave minimax problems with $\mathcal{O}(1/k^2)$ rate on squared gradient norm. *International Conference on Machine Learning*, 2021.

Research Highlight: Variable Selection for Kernel Two-Sample Tests

by Jie Wang, Santanu S. Dey, Yao Xie
School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA

We would like to thank the 2024 ICS Student Paper Award Committee members: Beste Basciftci, Austin Buchanan, Leonardo Lozano, and Hamed Rahimian for the honor of being selected as the runner-up for this award. We also extend our gratitude to the ICS Newsletter editor, Hamed Rahimian, for the invitation to share this research highlight with the ICS community. This summary is based on our work [WDX23].