RESEARCH ARTICLE

WILEY

# Massively parallelizable proximal algorithms for large-scale stochastic optimal control problems

Ajay K. Sampathirao[1] | Panagiotis Patrinos[2] | Alberto Bemporad[3] | Pantelis Sopasakis[4]

[1]Centrica Business Solutions, Antwerp, Belgium

[2]Department of Electrical Engineering (ESAT), KU Leuven, Leuven, Belgium

[3]IMT School for Advanced Studies, Lucca, Italy

[4]School of Electronics, Electrical Engineering and Computer Science (EEECS) and Centre for Intelligent Autonomous Manufacturing Systems (i-AMS), Queen's University Belfast, Belfast, UK

**Correspondence**
Pantelis Sopasakis, School of Electronics, Electrical Engineering and Computer Science (EEECS) and the Centre for Intelligent Autonomous Manufacturing Systems (i-AMS), Queen's University Belfast, Ashby Building, Stranmillis Road, Belfast, BT9 5AG, UK.
Email: p.sopasakis@qub.ac.uk

## Abstract
Scenario-based stochastic optimal control problems suffer from the curse of dimensionality as they can easily grow to six and seven figure sizes. First-order methods are suitable as they can deal with such large-scale problems, but may perform poorly and fail to converge within a reasonable number of iterations. To achieve a fast rate of convergence and high solution speeds, in this article, we propose the use of two proximal quasi-Newtonian limited-memory algorithms—MINFBE applied to the dual problem and the Newton-type alternating minimization algorithm (NAMA)—which can be massively parallelized on lockstep hardware such as graphics processing units. In particular, we use MINFBE and NAMA to solve scenario-based stochastic optimal control problems with affine dynamics, convex quadratic cost functions (with the stage cost functions being strongly convex in the control variable) and joint state-input convex constraints. We demonstrate the performance of these methods, in terms of convergence speed and parallelizability, on large-scale problems involving millions of variables.

**KEYWORDS**
graphics processing units (GPUs), parallelizable numerical optimization, scenario-based linear-quadratic constrained stochastic optimal control

## 1 | INTRODUCTION

### 1.1 | Background

Stochastic optimal control is the backbone of stochastic model predictive control (MPC), which is known for its appealing stability and constraint satisfaction properties[1,2] and has found several applications.[3-5] More specifically, scenario-based

stochastic MPC is gaining great popularity[6-9] due to its applicability to virtually any stochastic model of uncertainty that can be reasonably approximated by a discrete distribution. However, the limiting factor towards its industrial uptake is the computational time required to solve numerically the resulting large-scale optimization problem. Indeed, multistage scenario-based stochastic optimal control problems suffer from the curse of dimensionality and can lead to problems with millions of decision variables.[7]

Several approaches have been proposed for solving stochastic optimal control problems. Benders decomposition is a popular method for solving (mixed integer) linear stochastic problems,[10,11] including multistage formulations.[12] Stochastic dual dynamic programming is used for solving both linear[13] and general convex multistage stochastic problems.[14] The progressive hedging algorithm, the theoretical basis of which was laid by Rockafellar and Wets,[15,16] is an alternating direction of method multipliers (ADMM)-like algorithm[17] where the non-anticipativity constraints are relaxed; this can lead to parallelizable algorithms.[18] Other notable approaches include the progressive decoupling algorithm,[19] scenario aggregation methods,[20] and the most recent incremental proximal method.[21]

Graphics processing units (GPUs) have been used for their massive parallelization capabilities in applications as diverse as cryptocurrency mining,[22] cosmology,[23] medical image processing,[24] simulations of molecular dynamics,[25] machine learning,[26] and a lot more. GPUs are suitable for lockstep parallelization, where the same elementary operations are applied to different memory positions using dedicated functions known as *kernels*. Programming GPUs for general-purpose data-parallel computations is facilitated by programming languages and frameworks such as CUDA[27,28] (for NVIDIA GPUs, used by well-known software such as TensorFlow[29] and Caffe[30]), OpenCL, OpenACC, OpenGL and more.

Field programmable gate arrays (FPGAs) have also been used in MPC applications. Some of the algorithms that have been successfully implemented on FPGAs for solving MPC problems include interior point,[31,32] ADMM,[33,34] GPAD[35] and active set algorithms.[36] To date, FPGAs have led to solution times at the $\mu$ s range for problems of small to medium size.

In recent years, a number of papers have proposed parallelizable variants of numerical optimization methods such as the interior point method,[37] parallel quadratic programming,[38] ADMM[39-41] and other proximal algorithms.[42,43] In these approaches, GPUs are used to parallelize the involved algebraic operations and the solution of linear systems: the primal-dual optimality conditions in interior point algorithms and equality-constrained QPs in ADMM. Given the lockstep data parallelization paradigm of GPUs, numerical methods that aim at splitting the problem into smaller optimization problems that are to be executed in parallel (such as References 44 and 45) do not lend themselves to GPU implementations.

Scenario-based problems possess a certain structure that can be exploited to design very efficient *ad hoc* GPU-enabled implementations leading to a higher acceleration as discussed in Reference 7. It has been shown that first-order algorithms such as the accelerated proximal gradient method can be used to achieve significant speed-ups.[7,46,47] However, first-order methods tend to be prone to ill-conditioning as they disregard curvature information. This motivates the development of numerical methods that can exploit the underlying problem structure of scenario-based optimal control problems, come with good convergence characteristics, and are amenable to lockstep parallelization on GPUs.

## 1.2 | Contributions

In this article, we propose the use of two proximal quasi-Newtonian limited-memory algorithms for the parallelizable solution of scenario-based stochastic optimal control problems with linear or affine dynamics, convex quadratic cost functions (with the stage costs being strongly convex in the control variable) and joint state-input convex constraints. In particular, the key contributions of the article are as follows

1. We propose two massively parallelizable numerical methods that exploit the structure of scenario-based stochastic optimal control problems, building up on the MINFBE method[48] applied to the dual problem, the Newton-type alternating minimization algorithm (NAMA)[49] algorithms, as well as on our previous work on GPU-accelerated optimization.[50]
2. We demonstrate that all methods lend themselves to highly parallelizable implementations and lead to similar convergence speeds.

3. We show that NAMA allows a significantly higher parallelizability and lower computation times. In fact, MINFBE and NAMA involve only simple algebraic operations, use limited-memory BFGS directions, and can achieve better accuracy and significantly faster convergence than the accelerated proximal gradient method of Reference 7 (linear convergence rate instead of $\mathcal{O}(1/k^2)$).

## 1.3 | Notation

Let $\mathbb{N}, \mathbb{R}, \mathbb{R}^n$, and $\mathbb{R}^{m \times n}$ denote the sets of nonnegative integers, real numbers, $n$-dimensional vectors and $m$-by-$n$ matrices respectively. Let $\mathbb{N}_{[k_1,k_2]} := \{n \in \mathbb{N} : k_1 \le n \le k_2\}$. Let $\overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty\}$ be the set of extended-real numbers. Given a set $X \subseteq \mathbb{R}^n$ and $x \in \mathbb{R}^n$ we define the *indicator* of $X$ as the extended-real-valued function $\delta(\cdot \mid X) : \mathbb{R}^n \to \overline{\mathbb{R}}$ with $\delta(x \mid X) = 0$ for $x \in X$ and $\delta(x \mid X) = \infty$ otherwise. For $A \in \mathbb{R}^{m \times n}$, $A^\intercal$ denotes the transpose of $A$. For $A, B \in \mathbb{R}^{m \times n}$, we write $A \succ B$ $(A \succeq B)$ if $A - B$ is positive (semi)definite. For a convex function $f : \mathbb{R}^n \to \overline{\mathbb{R}}$, its convex conjugate function $f^*$ is defined as $f^*(y) = \sup_x \{x^\intercal y - f(x)\}$. Given a nonempty, closed, convex set $X \subseteq \mathbb{R}^n$, we define the projection operator onto $X$ as $\mathbf{proj}_X(x) = \mathbf{argmin}_{y \in X} \|y - x\|$.

The proximal operator of a proper, closed, convex function $g$ plays a major role in modern optimization theory and is defined as $\mathbf{prox}_{\lambda g}(v) = \mathbf{argmin}_z \{g(z) + 1/2\lambda \|v - z\|^2\}$, with $\lambda > 0$. Proximal operators of a great variety of functions including indicators of sets, distance-to-set functions and norms can be easily evaluated analytically and at a very low computational cost.[51] For example, the proximal operator of the indicator of a set $Y$ is the projection on $Y$, that is $\mathbf{prox}_{\lambda \delta(\cdot|Y)}(v) = \mathbf{proj}(v|Y)$.

## 2 | PROBLEM STATEMENT

We start by stating the stochastic optimal control problem we will study in this article.

## 2.1 | Stochastic dynamics on scenario trees

Consider a discrete-time stochastic dynamical system of the form

$$x_{t+1} = A_{w_t} x_t + B_{w_t} u_t + c_{w_t}, \tag{1}$$

with state $x_t \in \mathbb{R}^{n_x}$ and input $u_t \in \mathbb{R}^{n_u}$, which is driven by the stochastic process $w_t$. For example, Markov jump affine systems fall into this category.[52] The evolution of this system over a finite sequence of time instants, $t \in \mathbb{N}_{[0,N]}$, can be described using a *scenario tree*: a directed graph of the form shown in Figure 1. The scenario tree structure is essentially the representation of a discrete multistage probability distribution. A scenario tree represents the evolution of the system states as more information becomes available: at every stage $t$, we assume that the state, $x_t$, can be measured and a control action $u_t$ can be decided based on that measurement, thus modeling an entire feedback policy.

The nodes of the scenario tree are organized in *stages*, $t \in \mathbb{N}_{[0,N]}$, and indexed by a unique integer $i$. At stage $t = 0$ we assume that the state—which is the current state in an MPC setting—is known; this corresponds to the *root* node of the tree, which is indexed by $i = 0$. The nodes at a stage $t$ are denoted by $\mathbf{nodes}(t)$ and the nodes at stage $t = N$ are called the *leaf nodes* of the tree. For notational convenience, we will denote the nodes at stages $t \in \mathbb{N}_{[t_1,t_2]}$, with $0 \le t_1 \le t_2 \le N$, by $\mathbf{nodes}(t_1, t_2) = \bigcup_{t=t_1}^{t_2} \mathbf{nodes}(t)$. The set $\mathbf{nodes}(t)$ is a probability space: every node $i \in \mathbf{nodes}(t)$ is assigned a nonzero probability value $\pi^i$. Naturally $\pi^0 = 1$ and $\sum_{i \in \mathbf{nodes}(t)} \pi^i = 1$ for all $t \in \mathbb{N}_{[0,N]}$.

Every node $i$ at a stage $t \in \mathbb{N}_{[1,N]}$ has an *ancestor*, $\mathbf{anc}(i) \in \mathbf{nodes}(t - 1)$, and all nodes at a stage $t \in \mathbb{N}_{[0,N-1]}$ have a set of *children*, $\mathbf{child}(i) \subseteq \mathbf{nodes}(t + 1)$. The set $\mathbf{child}(i)$ is a probability space with probability vector $\pi^{[i]} \in \mathbb{R}^{|\mathbf{child}(i)|}$. This is a vector whose $i_+$th element is equal to $\pi^{i_+}/\pi^i$—for short $\pi^{[i]} = \frac{1}{\pi^i}(\pi^{i_+})_{i_+ \in \mathbf{child}(i)}$.
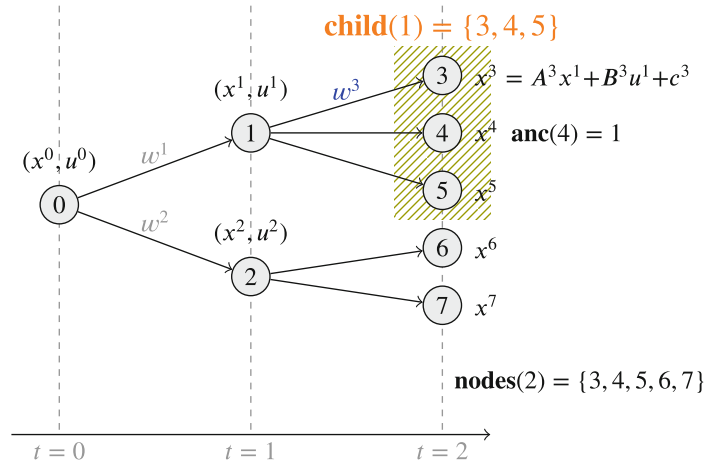
**FIGURE 1** Scenario tree structure with three stages and the system dynamics on its nodes. The cost associated with node $i = 1$ is $\ell_1(x^0, u^0, w^1)$.

The system dynamics, (1), across the nodes of the scenario tree can be stated as

$$x^{i_+} = A^{i_+}x^i + B^{i_+}u^i + c^{i_+}, \tag{2}$$

for $i \in \mathbf{nodes}(0, N-1)$, $i_+ \in \mathbf{child}(i)$. Note that the total number of scenarios coincides with the number of leaf nodes, and the number of non-leaf nodes with the number of free input variables (see Figure 1).

## 2.2 | Stochastic optimal control problem

A *multistage stochastic optimal control problem* for (1) with horizon $N$ can be formulated as

$$\mathbb{P}(p) : \underset{\{u_t\}_{t=0}^{N-1}, \{x_t\}_{t=0}^{N}}{\textbf{minimize}} \mathbb{E}\left[V_f(x_N) + \sum_{t=0}^{N-1} \ell_t(x_t, u_t, w_t)\right], \tag{3}$$

subject to (1) and the condition $x_0 = p$, where $x_t$ is the state at stage $t$ starting from $x_0$ and with the application of the control actions $u_0, u_1, \ldots, u_{t-1}$. Note that in this formulation, $\{u_t\}_{t=0}^{N-1}$ and $\{x_t\}_{t=0}^{N}$ are random variables. The *stage cost* at stage $t \in \mathbb{N}_{[1,N]}$ is a random variable which admits the values $\ell^i(x^{\mathbf{anc}(i)}, u^{\mathbf{anc}(i)}) := \ell_t(x^{\mathbf{anc}(i)}, u^{\mathbf{anc}(i)}, w^i)$, for $i \in \mathbf{nodes}(t)$, with probability $\pi^i$. The *terminal cost* function is also a random variable which admits the values $V_f(x^i)$ for $i \in \mathbf{nodes}(N)$ with probability $\pi^i$. That said, the optimal control problem can be written as

$$\mathbb{P}(p) : \underset{\substack{\{u^i\}_{i \in \mathbf{nodes}(0,N-1)} \\ \{x^i\}_{i \in \mathbf{nodes}(0,N)}}}{\textbf{minimize}} \sum_{i \in \mathbf{nodes}(1,N)} \pi^i \ell^i(x^{\mathbf{anc}(i)}, u^{\mathbf{anc}(i)}) + \sum_{i \in \mathbf{nodes}(N)} \pi^i V_f^i(x^i), \tag{4}$$

subject to the system dynamics (2) and the condition $x_0 = p$.

The stage cost function, $\ell^i : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \overline{\mathbb{R}}$, at node $i \in \mathbf{nodes}(1, N)$, is an extended-real-valued function which can be decomposed as follows

$$\ell^i(x, u) = \phi^i(x, u) + \overline{\phi}^i(F^i x + G^i u), \tag{5}$$

where $\phi^i : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ is a smooth convex function and $\overline{\phi}^i : \mathbb{R}^{m_i} \to \overline{\mathbb{R}}$ is a proper, extended-real-valued, possibly nonsmooth, convex, lower semicontinuous function and $F^i \in \mathbb{R}^{m_i \times n_x}$, $G^i \in \mathbb{R}^{m_i \times n_u}$. Functions $\overline{\phi}^i$ can be taken to be indicator functions so as to model constraints on inputs and states.

We can also decompose the terminal cost function, $V_f^i : \mathbb{R}^{n_x} \to \overline{\mathbb{R}}$, as follows

$$V_f^i(x) = \phi_N^i(x) + \overline{\phi}_N^i(F_N^i x), \tag{6}$$

where $F_N^i \in \mathbb{R}^{m_{N,i} \times n}$, $\phi_N^i : \mathbb{R}^{n_x} \to \mathbb{R}$ is real valued, smooth, convex function and $\overline{\phi}_N^i : \mathbb{R}^{m_{N,i}} \to \overline{\mathbb{R}}$ is a proper extended-real-valued, convex, lower semicontinuous function.

Functions $\overline{\phi}^i$ need not be smooth. They can be used to describe hard joint state-input constraints of the form $F^i x + G^i u \in Y^i$ by taking $\overline{\phi}^i(\cdot) = \delta(\cdot|Y^i)$. Similarly, $\overline{\phi}^i$ can describe soft constraints simply by replacing the indicator function $\delta(\cdot|Y^i)$ by a distance-to-set function. On the other hand, functions $\phi^i$ and $\phi_N^i$ are typically taken to be convex quadratic (and $\phi^i$ are assumed to be strongly convex with respect to $u$ and jointly convex in $(x, u)$). Hereafter, we consider the quadratic cost functions

$$\phi^i(x, u) = \begin{bmatrix} x \\ u \end{bmatrix}^\mathsf{T} \begin{bmatrix} Q_i & S_i^\mathsf{T} \\ S_i & R_i \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + q_i^\mathsf{T} x + r_i^\mathsf{T} u, \tag{7}$$

for $x \in \mathbb{R}^{n_x}$ and $u \in \mathbb{R}^{n_u}$, with $Q_i = Q_i^\mathsf{T} \succcurlyeq 0$, and $R_i = R_i^\mathsf{T} \succ 0$, and

$$\begin{bmatrix} Q_i & S_i^\mathsf{T} \\ S_i & R_i \end{bmatrix} \succcurlyeq 0, \tag{8}$$

for all $i \in \mathbf{nodes}(0, N-1)$. Lastly, $V_f(x) = x^\mathsf{T} P_N x + p_N^\mathsf{T} x$ for $x \in \mathbb{R}^{n_x}$ with $P_N = P_N^\mathsf{T} \succ 0$.

## 2.3 | Formulation of optimization problem

The decision variable of $\mathbb{P}(p)$ is the vector $x = \left( (u^i)_{i \in \mathbf{nodes}(0,N-1)}, (x^i)_{i \in \mathbf{nodes}(1,N)} \right) \in \mathbb{R}^n$, where $n = |\mathbf{nodes}(0, N-1)| n_u + |\mathbf{nodes}(1, N)| n_x$. Let us define the affine space

$$\mathcal{X}(p) = \left\{ x \,\middle|\, \begin{matrix} x^0 = p, x^{i_+} = A^{i_+} x^i + B^{i_+} u^i + c^{i_+}, \\ i \in \mathbf{nodes}(0, N-1), i_+ \in \mathbf{child}(i). \end{matrix} \right\}, \tag{9}$$

which describes the system dynamics. Let us also define the function $f(\cdot; p) : \mathbb{R}^n \to \overline{\mathbb{R}}$ as

$$f(x; p) = \sum_{i \in \mathbf{nodes}(1,N)} \pi^i \phi^i(x^{\mathbf{anc}(i)}, u^{\mathbf{anc}(i)}) + \sum_{i \in \mathbf{nodes}(N)} \pi^i \phi_N(x^i) + \delta(x|\mathcal{X}(p)). \tag{10a}$$

Next, for every $i \in \mathbf{nodes}(0, N)$ we define $z^i = F^i x^{\mathbf{anc}(i)} + G^i u^{\mathbf{anc}(i)}$, and for every $i \in \mathbf{nodes}(N)$ we define $z_N^i = F_N^i x^i$. Let $z = ((z^i)_{i \in \mathbf{nodes}(0,N)}, (z_N^i)_{i \in \mathbf{nodes}(N)})$. We then define the function $g : \mathbb{R}^m \to \overline{\mathbb{R}}$ as

$$g(z) = \sum_{i \in \mathbf{nodes}(1,N)} \pi^i \overline{\phi}^i(z^i) + \sum_{i \in \mathbf{nodes}(N)} \pi^i \overline{\phi}_N(z_N^i), \tag{10b}$$

where $z = ((z^i)_{i \in \mathbf{nodes}(0,N)}, (z_N^i)_{i \in \mathbf{nodes}(N)})$ and define $H : \mathbb{R}^n \to \mathbb{R}^m$ as a linear operator that maps $x$ to a vector $z \in \mathbb{R}^m$ as above with $z^i = F^i x^{\mathbf{anc}(i)} + G^i u^{\mathbf{anc}(i)}$ for $i \in \mathbf{nodes}(0, N)$ and $z_N^i = F_N^i x^i$ for $i \in \mathbf{nodes}(N)$.

Given that functions $\phi^i$ are quadratic as described in the previous section, function $f$ is strongly convex (as it follows from Reference 53 [Prop. 6]) with a strong convexity modulus $\mu_f$, therefore the convex conjugate of $f$, $f^*$, is differentiable with $L$-Lipschitz gradient, with $L = 1/\mu_f$, because of Reference 54 (Prop. 12.60).

Problem $\mathbb{P}(p)$ can be written as

$$\mathbb{P}(p) : \underset{x \in \mathbb{R}^n}{\textbf{minimize}}\, f(x; p) + g(Hx). \tag{11}$$

Hereafter, we assume that $\mathbb{P}(p)$ is feasible. The Fenchel dual of Problem $\mathbb{P}(p)$ in Equation (11) is

$$\mathbb{D}(p) : \underset{y \in \mathbb{R}^m}{\textbf{minimize}}\, f^*(-H^\intercal y; p) + g^*(y). \tag{12}$$

Let us define the function $\hat{f} : \mathbb{R}^m \to \mathbb{R}$ as

$$\hat{f}(y; p) := f^*(-H^\intercal y; p). \tag{13}$$

Then, problem $\mathbb{D}(p)$ can be written as

$$\mathbb{D}(p) : \underset{y \in \mathbb{R}^m}{\textbf{minimize}}\, \hat{f}(y; p) + g^*(y). \tag{14}$$

For given $p$, strong duality holds if there is an $x \in \mathcal{X}(p)$ such that $Hx \in \textbf{relint dom}\,g$ Reference 55 (Theorem 15.23)—we will hereafter assume that this assumption is satisfied.

## 2.4 | Optimality conditions

A simple optimality condition for the dual optimization problem of Equation (12) is

$$y - \textbf{prox}_{\lambda g^*}(y - \lambda \nabla \hat{f}(y)) = 0, \tag{15}$$

for some $\lambda > 0$.[56] By virtue of the Moreau decomposition formula, Equation (15) is equivalently written as

$$\nabla \hat{f}(y) + \textbf{prox}_{\lambda^{-1}g}(\lambda^{-1}y - \nabla \hat{f}(y)) = 0. \tag{16}$$

We define the *forward–backward mapping*

$$T_\lambda(y) := \textbf{prox}_{\lambda g^*}(y - \lambda \nabla \hat{f}(y)), \tag{17}$$

which, using the Moreau decomposition property, becomes

$$T_\lambda(y) = y - \lambda \nabla \hat{f}(y) - \lambda \textbf{prox}_{\lambda^{-1}g}(\lambda^{-1}y - \nabla \hat{f}(y)), \tag{18}$$

and we also define the *fixed-point residual* mapping

$$R_\lambda(y) := \lambda^{-1}(y - T_\lambda(y)) \tag{19}$$

$$= z_\lambda(y) - Hx(y), \tag{20}$$

where $x(y)$ and $z_\lambda(y)$ are defined as

$$x(y) := \nabla f^*(-H^\intercal y), \tag{21a}$$

$$z_\lambda(y) := \textbf{prox}_{\lambda^{-1}g}\left(\lambda^{-1}y + Hx(y)\right), \tag{21b}$$

therefore,

$$x(y) = \underset{z}{\textbf{argmin}}\{\langle z, H^{\mathsf{T}}y \rangle + f(z)\}. \tag{22}$$

Note also that $T_\lambda(y)$ can be computed from Equation (18) as

$$T_\lambda(y) = y - \lambda\Big(\nabla\hat{f}(y) - z_\lambda(y)\Big). \tag{23}$$

The aforementioned optimality condition in Equation (15) is equivalently written as $R_\lambda(y) = 0$, that is, solving the dual optimization problem (12) becomes equivalent to finding a zero of the operator $R_\lambda$.

# 3 | NUMERICAL OPTIMIZATION

## 3.1 | Forward–backward envelope

The *forward–backward envelope* (FBE) of (12) is a real-valued function $\varphi_\lambda$ given by[57,58]

$$\varphi_\lambda(y) = \hat{f}(y) + g^*(T_\lambda(y)) - \lambda\langle\nabla\hat{f}(y), R_\lambda(y)\rangle + \frac{\lambda}{2}\|R_\lambda(y)\|^2. \tag{24}$$

If $\hat{f}$ is twice continuously differentiable—conditions under which this is the case can be found in Reference 59—then $\varphi_\lambda$ is continuously differentiable with

$$\nabla\varphi_\lambda(y) = (I - \lambda\nabla^2\hat{f}(y))R_\lambda(y). \tag{25}$$

Note that in practice it is not necessary to compute or store the Hessian matrix $\nabla^2\hat{f}(y)$. Instead, it suffices to implement an algorithm that returns Hessian-vector products of the form $\nabla^2\hat{f}(y) \cdot z$. The most important property of the FBE is that for $\lambda \in (0, 1/L)$, the set of minimizers of (12) coincides with

$$\begin{aligned}
\textbf{argmin } \varphi_\lambda \equiv \textbf{zer } \nabla\varphi_\lambda &:= \{y : \nabla\varphi_\lambda(y) = 0\} \\
&= \textbf{argmin } \hat{f}(y) + g^*(y) = \textbf{zer } R_\lambda.
\end{aligned}$$

Essentially, the problem of solving the dual optimization problem (12) is equivalent to the unconstrained minimization of the continuously differentiable function $\varphi_\lambda$, that is

$$\textbf{inf } \hat{f}(y) + g^*(y) = \textbf{inf } \varphi_\lambda, \tag{26a}$$

$$\textbf{argmin } \hat{f}(y) + g^*(y) = \textbf{argmin } \varphi_\lambda. \tag{26b}$$

Moreover, the above is equivalent to finding a zero of the fixed-point residual operator. In the case where $\hat{f}$ is strongly convex quadratic, $\phi_\lambda$ is both continuously differentiable and convex.

## 3.2 | Dual MINFBE method

If $\hat{f}$ is twice differentiable, according to (26) the original (dual) optimization problem can be cast as an *unconstrained* optimization problem with a smooth cost function. As a result we can use an appropriate unconstrained optimization method to solve such problems, such as limited-memory BFGS,[50] however, convergence is only guaranteed under restrictive requirements (such as twice differentiability and uniform convexity of the FBE[60]).

---

**Algorithm 1.** Dual MINFBE with L-BFGS directions

---

**Require:** $\lambda \in (0, 1/L)$, $y^0$, $m$ (memory), $\epsilon$ (tolerance)
**Ensure:** Primal-dual solution triple $(x, z, y)$
1: Initialize an L-BFGS buffer with memory $m$
2: **while** $\|R_\lambda(y^k)\|_\infty > \epsilon$ **do**
3:     $d^k = -B^k \nabla \varphi_\lambda(y^k)$ (Compute an L-BFGS direction using the L-BFGS buffer)
4:     Choose the largest $\tau_k \in \{2^{-\nu}\}_{\nu \in \mathbb{N}}$ so that

$$\varphi_\lambda(w^k) \le \varphi_\lambda(y^k), \tag{27}$$

    where $w^k = y^k + \tau_k d^k$
5:     $x^{k+1} = T_\lambda(w^k)$
6:     Compute the L-BFGS-related quantities $s^k = y^{k+1} - y^k$, $q^k = \nabla \varphi_\lambda(y^{k+1}) - \nabla \varphi_\lambda(y^k)$ and $\rho_k = \langle s^k, q^k \rangle$
7:     **if** $\langle s^k, q^k \rangle > \epsilon' \|s^k\|^2 \|\nabla \varphi_\lambda(y^k)\|^2$ **then**
8:         Push $(s^k, q^k, \rho^k)$ into the L-BFGS buffer
9:     **end if**
10:     $k = k + 1$
11: **end while**
12: **return** $(x, z, y) = (x(y^k), z(y^k), y^k)$

---

Instead, MINFBE is a method that can be applied to problems with nonsmooth cost functions using the FBE as a merit function using a simple line search.[48] MINFBE involves simple and computationally inexpensive iterations, and exhibits superior global convergence properties. The application of MINFBE to the dual optimization problem, $\mathbb{D}(p)$, leads to Algorithm 1.

MINFBE consists in applying the forward–backward mapping on the extrapolated vector $w^k = y^k + \tau_k d^k$ which satisfies the decrease condition (27). The L-BFGS buffer is updated with the vectors $s^k$, $q^k$, and their inner product $\rho_k$, provided that the minimum-curvature condition in line 7 is satisfied for a small tolerance $\epsilon' > 0$, following.[61]

The algorithm iterates on the dual vectors $y^k$ and returns a triple $(x, z, y)$ which satisfies the termination condition $\|R_\lambda(y^k)\|_\infty \le \epsilon$, which, in light of Equation (20) means that

$$\|z - Hx\|_\infty \le \epsilon, \tag{28a}$$

$$-H^\mathsf{T} y \in \partial f(x), \tag{28b}$$

$$\mathbf{dist}_{\|\cdot\|_\infty}(y, \partial g(z)) \le \lambda \epsilon, \tag{28c}$$

where $\mathbf{dist}_{\|\cdot\|_\infty}$ denotes the point-to-set distance with respect to the $\infty$-norm.

We should highlight that the line search in line 4 of Algorithm 1 is a simple descent condition on the FBE, which is simpler than the Wolfe conditions used in Reference 50. Moreover, although in Algorithm 1 we use L-BFGS directions, the method works with any direction of descent $d^k$ with respect to the FBE, that is, if $\langle d^k, \nabla \varphi_\lambda(y^k) \rangle \le 0$.

If $f$ is quadratic plus the indicator of an affine subspace, $x(y)$ turns out to be linear, that is

$$x(w) = x(y + \tau d) = x(y) + \tau x(d), \tag{29}$$

and $\hat{f}$ is a quadratic function, that is $\nabla \hat{f}$ is linear and $\hat{f}(y) = \langle y, \nabla \hat{f}(y) \rangle$, from which we can see that

$$\begin{aligned} \hat{f}(y + \tau d) &= \langle y + \tau d, \nabla \hat{f}(y + \tau d) \rangle \\ &= \hat{f}(y) + \tau^2 \hat{f}(d) + 2\tau \langle y, \nabla \hat{f}(d) \rangle. \end{aligned} \tag{30}$$

By virtue of the last two properties and after some algebraic manipulations, we find that the line search condition $\varphi_\lambda(w^k) - \varphi_\lambda(y^k) \leq 0$ is equivalent to

$$\alpha_2(y, d)\tau^2 + \alpha_1(d)\tau + \alpha_0(\tau; y, d) \leq 0, \tag{31}$$

where

$$\alpha_0(\tau; y, d) = g^*(T_\lambda(y + \tau d)) - g^*(T_\lambda(y)) + \frac{\lambda}{2}[\|z_\lambda(y + \tau d)\|^2 - \|z_\lambda(y)\|^2], \tag{32a}$$

$$\alpha_1(y, d) = \langle Hx(d), 2y - \lambda Hx(y) \rangle, \tag{32b}$$

$$\alpha_2(d) = \hat{f}(d) - \frac{\lambda}{2}\|Hx(d)\|^2, \tag{32c}$$

and $\hat{f}(d)$ can be computed by invoking Theorem 23.5 from Reference 62, from which

$$\hat{f}(d) = -\langle Hx(d), d \rangle - f(x(d)). \tag{33}$$

Note that $\alpha_1$ and $\alpha_2$ do not depend on $\tau$, therefore, can be computed once per iteration. This leads to a significant reduction of the involved floating point operations per iteration. The most computationally demanding parts of MINFBE are (i) the computation of $x(y)$ and $x(d)$, and (ii) the computation of the Hessian-vector product $\nabla \hat{f}(y^k)R_\lambda(y^k)$ that is required to determine $\nabla \varphi_\lambda(y^k)$ in line 3 of Algorithm 1. The involved operations can be parallelized on a GPU as we shall discuss in Section 3.4, but the computations of $x(y)$, $x(d)$ and $\nabla \hat{f}(y^k)R_\lambda(y^k)$ cannot be parallelized.

Often, the Lipschitz constant of the gradient of $\hat{f}$ is not known and needs to be estimated with a backtracking procedure. The original backtracking proposed in Reference 48 halves the value of $\lambda$ after the line search in line 4 if the following condition is satisfied

$$\hat{f}(T_\lambda(w^k)) > \hat{f}(y^k) - \lambda\langle \nabla \hat{f}(y^k), R_\lambda(y^k) \rangle + \frac{(1 - \beta)\lambda}{2}\|R_\lambda(y^k)\|^2, \tag{34}$$

for some $\beta \in [0, 1)$. The values $\hat{f}(y^k)$, $\langle \nabla \hat{f}(y^k), R_\lambda(y^k) \rangle$ and $\|R_\lambda(y)\|^2$ are known from the preceding line search, so the cost of the backtracking is that of computing $\hat{f}(T_\lambda(w^k))$. Alternatively, we may use the backtracking method proposed in Reference 63 (Linesearch 1) which halves $\lambda$ if

$$\lambda\|\nabla \hat{f}(T_\lambda(y^k)) - \nabla \hat{f}(y^k)\| > \epsilon''\|T_\lambda(y^k) - y^y\|, \tag{35}$$

where $\epsilon'' \in (0, 1/2)$. This backtracking procedure has a lower computational cost compared to Equation (34). In both cases, the L-BFGS buffer is emptied when the value of $\lambda$ is updated.

## 3.3 | Parallelizable Newton-type alternating minimization algorithm

The NAMA can be used to solve the dual optimization problem $\mathbb{D}(p)$ in Equation (12) without the need to compute the gradient of the FBE.[49] NAMA, applied to the dual optimization problem is given in Algorithm 2.

NAMA involves a simple line search which consists in determining a $\tau_k$ so that the dual vector defined as $w^k = y^k + \tau_k d^k + (1 - \tau_k)r^k$ satisfies the descent condition $\varphi_\lambda(w^k) \leq \varphi_\lambda(y^k)$. Again, if $\hat{f}$ is a quadratic function, we can precompute certain quantities in a fashion akin to Equation (31). In particular, before the line search in line 7 of Algorithm 2 we need to compute $x(r)$ and $x(d)$.

The main computational cost involved in Algorithm 2 comes from the evaluation of $x(y)$, $x(r)$, and $x(d)$. Note that if $x$ is linear, $x(w)$ can be computed at a very low computational cost. In particular, the extrapolated vector $w^k$ can be written

---

**Algorithm 2.** NAMA method for the dual optimization problem

---

**Require:** $\lambda \in (0, 1/L)$, $y^0$, $\epsilon > 0$ (tolerance)

**Ensure:** Primal-dual solution triple $(x, z, y)$

1: $k = 0$

2: **while** $\|R_\lambda(y^k)\| > \epsilon$ **do**

3: $\quad x^k = x(y^k), z^k = z_\lambda(y^k)$

4: $\quad r^k = z^k - Hx^k$

5: $\quad d^k = -B^k r^k$ (Compute an L-BFGS direction using the L-BFGS buffer)

6: $\quad$ Choose the largest $\tau_k \in \{2^{-\nu}\}_{\nu \in \mathbb{N}}$ so that

$$\varphi_\lambda(w^k) \leq \varphi_\lambda(y^k), \tag{36}$$

$\quad$ where $w^k = y^k + \tau_k d^k + (1 - \tau_k)r^k$

7: $\quad \tilde{x}^k = x(w^k), \tilde{z}^k = z_\lambda(w^k)$

8: $\quad y^{k+1} = y^k + \lambda(H\tilde{x}^k - \tilde{z}^k)$

9: $\quad$ Compute the L-BFGS-related quantities $s^k = y^{k+1} - y^k$, $q^k = R_\lambda(y^{k+1}) - r^k$ and $\rho_k = \langle s^k, q^k \rangle$

10: $\quad$ **if** $\langle s^k, q^k \rangle > \epsilon' \|s^k\|^2 \|r^k\|^2$ **then**

11: $\quad\quad$ Push $(s^k, q^k, \rho^k)$ into the L-BFGS buffer

12: $\quad$ **end if**

13: $\quad k = k + 1$

14: **end while**

15: **return** $(x, z, y) = (x(y^k), z(y^k), y^k)$

---

as $w^k = \tilde{y}^k + \tau_k \tilde{d}^k$, where $\tilde{y}^k = y^k + r^k$ and $\tilde{d}^k = d^k - r^k$, therefore the decrease condition of NAMA in Equation (36) is equivalent to Equation (31) with $\tilde{y}^k$ and $\tilde{d}^k$ in lieu of $y^k$ and $d^k$ respectively, that is,

$$\alpha_2(\tilde{y}, \tilde{d})\tau^2 + \alpha_1(\tilde{d})\tau + \alpha_0(\tau; \tilde{y}, \tilde{d}) \leq 0. \tag{37}$$

Again, the Lipschitz constant $L$ may not be known, but it can be determined using the line search method we described in the previous section.

Overall, given that the computation of Hessian-vector products in MINFBE comes at approximately the same cost as computing the dual gradient, and given that the computation of $x(r)$ and $x(d)$ can be carried out in parallel, NAMA has a lower per-iteration computation cost. Although MINFBE and NAMA exhibit similar convergence properties, with NAMA we can afford a greater parallelizability that leads to superior performance in practice as we shall show in Section 4.

## 3.4 | Efficient parallel computations

GPUs have a hardware architecture that allows the execution of the same set of instructions on different memory positions. GPUs are equipped with a set of SIMD stream processors, each having its own computing resources, that execute "compute kernels," that is, functions that are executed simultaneously on different data.

NVIDIA's GPUs use the CUDA programming interface where kernels are executed in parallel threads, which are organized in *blocks* which can share memory and which are in turn organized in grids. At a hardware level, threads are executed in parallel in *warps* of 32 threads. Threads in the same block have asynchronous read/write access to a local shared memory and can synchronize. Each thread has its own local memory, and all threads have access to the device's global memory. Modern GPUs count several streaming multiprocessors with hundreds of cores, possess a computing throughput of several Tera-FLOPs, and have a significant memory capacity of several GBs. The hardware architecture and programming model of GPUs necessitates a fresh look at parallelization approaches for numerical optimization. Kernels are best suited for the parallel execution of simple numerical operations.

The efficient computation of the dual gradient is of crucial importance for the performance of the algorithm we are about to describe. By virtue of the conjugate subgradient theorem (see Theorem 23.5 in Reference 62), we have that

$$
x(y) = \underset{x \in \mathcal{X}(p)}{\textbf{argmin}} \left\{ \sum_{i \in \textbf{nodes}(1,N)} \pi^i \hat{\phi}^i(x^{\textbf{anc}(i)}, u^{\textbf{anc}(i)}) + \sum_{i \in \textbf{nodes}(N)} \pi^i \hat{\phi}_N(x^i) \right\}, \tag{38}
$$

where $\hat{\phi}^i(x^{\textbf{anc}(i)}, u^{\textbf{anc}(i)}) = \phi^i(x^{\textbf{anc}(i)}, u^{\textbf{anc}(i)}) + \langle y^i, F^i x^{\textbf{anc}(i)} + G^i u^{\textbf{anc}(i)} \rangle$ for $i \in \textbf{nodes}(1,N)$, and $\hat{\phi}_N(x^i) = \phi_N(x^i) + \langle y^i, F_N^i x^i \rangle$, for $i \in \textbf{nodes}(1,N)$. The solution of the minimization problem in Equation (38) can be determined via dynamic programming leading to Algorithms 3 and 4.[47] Algorithm 3 calculates the matrices $\Phi^i$, $\Theta^i$, $D^i$, $\Lambda^i$, $K^i$, $\sigma^i$, and $\hat{c}^i$ following a Riccati-type recursion. These matrices are computed once offline and are used in Algorithm 4. Note that Algorithm 3 is also amenable to parallelization. In cases where the data of the optimal control problem need to be updated (e.g., if the dynamical system is time varying, or the parameters of the cost must be updated in real time), the computation of these matrices can be carried out on a GPU and in fact the time for their computation is negligible compared to that of solving the problem.

The computation of Hessian-vector products of the form $\nabla^2 \hat{f}(y) r$ that is required for the computation of the gradient of the FBE. This can be rewritten as Gâteux directional derivative of the gradient $\nabla \hat{f}$ at $y$ along the direction $r$, which is given by

$$
\nabla^2 \hat{f}(y) r = \lim_{\epsilon \to 0} \frac{\nabla \hat{f}(y + \epsilon r) - \nabla \hat{f}(y)}{\epsilon}. \tag{39}
$$

---

**Algorithm 3.** Offline computation of matrices used in Algorithms 4 and 5

---

**Require:** $N$ (prediction horizon), $A^i$, $B^i$, $c^i$ (system data), $Q^i$, $R^i$, $S^i$, $q^i$, $r^i$, $P^N$, $p^N$ (weight matrices), $\pi^i$ (probabilities), $F^i$, $G^i$, $F_N^i$

**Ensure:** $\Phi^i$, $\Theta^i$, $D^i$, $\Lambda^i$, $K^i$, $\sigma^i$, $\bar{c}^i$

1: $P^i \leftarrow P^N$, for all $i \in \textbf{nodes}(N)$
2: **for** $k = N-1, \ldots, 0$ **do**
3:     **for** [ **do** in parallel ] $i \in \textbf{nodes}(k)$
4:         $\bar{P}^i \leftarrow \sum_{i_+ \in \textbf{child}(i)} B^{i_+ \top} P^{i_+} B^{i_+}$
5:         $\bar{R}^i \leftarrow 2(\pi^i R^i + \bar{P}^i)$, $\Phi^i = -(\bar{R}^i)^{-1} G^{i\top}$
6:         $K^i \leftarrow -2(\bar{R}^i)^{-1} \left( \pi^i S^i + \sum_{i_+ \in \textbf{child}(i)} B^{i_+ \top} P^{i_+} A^{i_+} \right)$
7:         $\sigma^i \leftarrow r^i - 2(\bar{R}^i)^{-1} \sum_{i_+ \in \textbf{child}(i)} B^{i_+ \top} P^{i_+} c^{i_+}$
8:         $\bar{A}^{i_+} \leftarrow A^{i_+} + B^{i_+} K^i$, $D^{i_+} = F^{i_+} + G^{i_+} K^i + S^{i\top} K^i \Phi^i$, $\forall i_+ \in \textbf{child}(i)$
9:         $\bar{c}^i \leftarrow K^{i\top} r^i + q^i + S^{i\top} K^i \sigma^i + 2 \sum_{i_+ \in \textbf{child}(i)} \bar{A}^{i_+ \top} P^{i_+} c^{i_+}$
10:         **if** $k = N - 1$ **then**
11:             $\Theta^{i_+} \leftarrow -(\bar{R}^{i_+})^{-1} B^{i_+ \top} F_N^{i_+ \top}$, $\forall i_+ \in \textbf{child}(N-1)$
12:             $\Lambda^{i_+} \leftarrow F^{i_+} \bar{A}^{i_+ \top}$, $\forall i_+ \in \textbf{child}(N-1)$
13:             $\bar{c}^i \leftarrow \bar{c}^i + \sum_{i_+ \in \textbf{child}(i)} A^{i_+ \top} p^N$
14:             $\sigma^i \leftarrow \sigma^i + \sum_{i_+ \in \textbf{child}(i)} B^{i_+ \top} p^N$
15:         **else**
16:             $\Theta^{i_+} \leftarrow -(\bar{R}^{i_+})^{-1} B^{i_+ \top}$, $\forall i_+ \in \textbf{child}(i)$
17:             $\Lambda^{i_+} \leftarrow \bar{A}^{i_+ \top}$, $\forall i_+ \in \textbf{child}(i)$
18:         **end if**
19:         $P^i \leftarrow \pi^i (Q^i + K^{i\top} R^i K^i + S^{i\top} K^i) + \sum_{i_+ \in \textbf{child}(i)} \bar{A}^{i_+ \top} P^{i_+} \bar{A}^{i_+ \top}$
20:     **end for**
21: **end for**
22: **return** $\Phi^i$, $\Theta^i$, $D^i$, $\Lambda^i$, $K^i$, $\sigma^i$, $\bar{c}^i$

---

---

**Algorithm 4.** Computation of the dual gradient, $x(y)$

---

**Require:** Dual vector $y \in \mathbb{R}^m$
**Ensure:** $x(y)$
1: $\hat{q}^i \leftarrow y^i$, for all $i \in \mathbf{nodes}(N)$
2: **for** [ **do** in parallel ] $i \in \mathbf{nodes}(0, N-1)$
3:      $u^i \leftarrow \Phi^i y^i + \sigma^i$
4:      $\hat{q}^i \leftarrow D^{i\intercal} y^i + \bar{c}^i$
5: **end for**
6: **for** $k = N-1, \dots, 0$ **do**
7:      **for** [ **do** in parallel ] $i \in \mathbf{nodes}(k)$
8:          $u^i \leftarrow \sum_{i_+ \in \mathbf{child}(i)} \Theta^{i_+} \hat{q}^{i_+}$
9:          $\hat{q}^i \leftarrow \sum_{i_+ \in \mathbf{child}(i)} \Lambda^{i_+ \intercal} \hat{q}^{i_+}$
10:      **end for**
11: **end for**
12: $x^0 = p$
13: **for** $i \in \mathbf{nodes}(0, N-1)$ **do**
14:      $u^i \leftarrow K^i x^i + u^i$
15:      **for** [ **do** in parallel ] $i_+ \in \mathbf{child}(i)$
16:          $x^{i_+} \leftarrow A^{i_+} x^i + B^{i_+} u^i + c^{i_+}$
17:      **end for**
18: **end for**
19: **return** $x(y) = (\{x^i\}, \{u^i\})$.

---

As the gradient $\nabla \hat{f}(y)$ is affine—meaning that $\nabla \hat{f}(y) - \nabla \hat{f}(0)$ is linear—we have that $\nabla \hat{f}(y + \epsilon r) - \nabla \hat{f}(y) = \epsilon (\nabla \hat{f}(r) - \nabla \hat{f}(0))$, so

$$\nabla^2 \hat{f}(y) r = \nabla \hat{f}(r) - \nabla \hat{f}(0). \tag{40}$$

This means that Hessian-vector products can be determined via gradient computations. In particular, Hessian-vector products can be computed via Algorithm 5. It should be noted that Algorithms 4 and 5 incur roughly the same computation cost.

Lastly, most proximal operations can be massively parallelized. For example, if $\bar{\phi}^i(z) = \delta(z|Y^i)$ and $Y^i = \{z : z_{\mathbf{min}}^i \leq z \leq z_{\mathbf{max}}^i\}$, then the computation of $\mathbf{prox}_{\lambda \bar{\phi}^i} = \mathbf{proj}_{Y^i}$ is element-wise independent and can be easily parallelized. Likewise, a great many proximal operators, such as those of the indicators of rectangles and common norm-balls, and functions such as $\| \cdot \|_1$, the Huber loss function and more, lend themselves to high parallelizability.[56]

In general, the total memory that needs to be allocated on the GPU grows linearly with the length of the L-BFGS buffer, linearly with the prediction horizon, and linearly with the number of nodes of the tree, and quadratically with the system states and inputs. The additional parallelization in NAMA requires the allocation of additional memory on the GPU, but leads to a higher throughput and occupancy of the device.

## 3.5 | Preconditioning

Stochastic optimal control problems tend to be ill conditioned because of the presence of generally small probability values. As first-order methods are known to be affected by the problem being ill conditioned, here we make use of a simple diagonal preconditioning heuristic where we scale the original dual variables $y = ((y^i)_{i \in \mathbf{nodes}(0,N)}, (y_N^i)_{i \in \mathbf{nodes}(N)})$ by introducing the scaled dual variables $\bar{y} = ((\bar{y}^i)_{i \in \mathbf{nodes}(0,N)}, (\bar{y}_N^i)_{i \in \mathbf{nodes}(N)})$ with

$$\bar{y}^i = \frac{y^i}{\sqrt{\pi^i}}, \tag{41}$$

---

**Algorithm 5.** Computation of Hessian-vector products required for the computation of $\nabla\varphi_\lambda$

---

**Require:** Vector $r$

**Ensure:** Hessian-vector product, $\nabla^2\hat{f}(y) \cdot r$

1: $\hat{q}^i \leftarrow r^i$, for all $i \in \mathbf{nodes}(N)$
2: **for** $k = N-1, \dots, 0$ **do**
3:     **for** [ **do** in parallel ] $i \in \mathbf{nodes}(k)$
4:         $\hat{u}^i \leftarrow \Phi^i r^i + \sum_{i_+ \in \mathbf{child}(i)} \Theta^{i_+} \hat{q}^{i_+}$
5:         $\hat{q}^i \leftarrow D^{i\top} r^i + \sum_{i_+ \in \mathbf{child}(i)} \Lambda^{i_+\top} \hat{q}^{i_+}$
6:     **end for**
7: **end for**
8: $\hat{x}^0 = 0$
9: **for** $i \in \mathbf{nodes}(0, N-1)$ **do**
10:     $\hat{u}^i \leftarrow K^i \hat{x}^i + \hat{u}^i$
11:     **for** $i_+ \in \mathbf{child}(i)$ **do**
12:         $\hat{x}^{i_+} \leftarrow A^{i_+} \hat{x}^i + B^{i_+} \hat{u}^i$
13:     **end for**
14: **end for**
15: **return** $\nabla^2\hat{f}(y) \cdot r = (\{\hat{x}^i\}, \{\hat{u}^i\})$.

---

for $i \in \mathbf{nodes}(0, N)$ and

$$\bar{y}_N^i = \frac{y_N^i}{\sqrt{\pi^i}}, \tag{42}$$

for $i \in \mathbf{nodes}(N)$. This scaling is a heuristic similar to the Jacobi preconditioning discussed in Reference 64.

## 3.6 | Warm start

Generally, the accelerated projected gradient method converges at a rate $\mathcal{O}(1/k^2)$ and although it may exhibit slow convergence, its iterations are computationally cheap, so it can be used to warm start MINFBE and NAMA. We have observed that running as few as five iterations of GPAD[7,65] can provide a good warm starting point for MINFBE and NAMA.

## 4 | NUMERICAL SIMULATIONS

This section is organized in two parts: in Section 4.1 we compare MINFBE and NAMA with the accelerated projected gradient method and discuss the convergence rate of each method. In particular, we demonstrate that a serial implementation of NAMA and MINFBE leads to superior performance compared to the accelerated proximal gradient method. The two methods exhibit comparable convergence speed. Next, in Section 4.2 we apply MINFBE and NAMA to solve a large-scale stochastic optimal control problem for the operating management of the drinking water network of Barcelona taken from Reference 7. We show that NAMA affords a higher parallelization leading to a significant performance improvement.

## 4.1 | Spring-mass-damper array

Consider an array of $M$ consecutive point particles of mass $m$ connected to each other through elastic springs of stiffness $k_s$ and linear dampers with viscous damping coefficients $b_d$ illustrated in Figure 2.

In between the successive masses $j$ and $j+1$, for $j = 1, \dots, M-1$, there is an actuator that can apply a force $u_j \in [u_{\min}, u_{\max}]$. The state variable of this system comprises the positions $p_j$ of the masses and their velocities $v_j$, which are
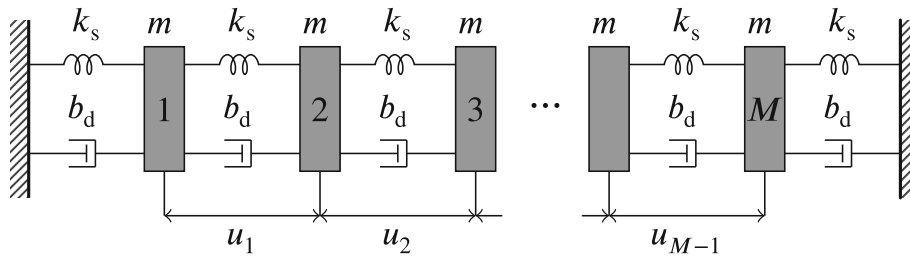
**FIGURE 2** Array of $M$ consecutive interconnected masses, $M + 1$ elastic springs and dampers and $M - 1$ actuators.
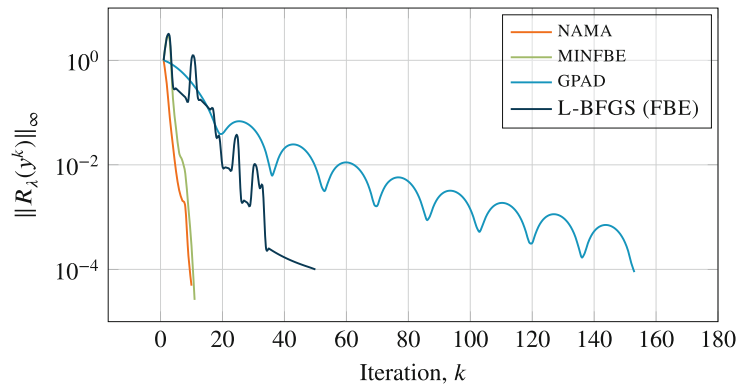


**FIGURE 3** Infinity norm of the fixed point residual against the iterations; comparison between NAMA, MINFBE, GPAD, and the L-BFGS method applied to $\varphi_\lambda$ with a memory of 5.

constrained in $[p_{min}, p_{max}]$ and $[v_{min}, v_{max}]$, respectively. The system is described by a set of linear differential equations which can be obtained by the application of Newton's second law of motion, which, after discretization with sampling time $T_s$ and a zero-order hold, yields a discrete-time linear time invariant system. Furthermore, we assume that there is an external additive disturbance $c_{w_k}$, as in Equation (1), which is driven by a discrete Markov process, $w_k$, with two modes.

In this example, we consider a stochastic optimal control problem with prediction horizon $N$, quadratic stage cost functions $\phi^i(x, u) = x^\mathsf{T}Qx + u^\mathsf{T}Ru$, and quadratic terminal costs $\phi^i_N(x) = x^\mathsf{T}P_Nx$. Moreover, we have $M = 5$ masses with $m = 5$ kg, $k_s = 1$ N/m, $b_d = 0.1$ Ns/m, $u_{max} = -u_{min} = 2$ N and the maximum allowed velocity is 5 m/s. The prediction horizon is $N = 11$ and the external disturbance $c_k$ is driven by a Markov chain with two modes with initial probability distribution $p_c = (0.5, 0.5)$ and probability transition matrix $P_c = \begin{bmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{bmatrix}$; at mode 1 the value of $c$ is zero and at mode 2, $c$ takes the value 0.1. The sampling time is $T_s = 0.5$ s. Lastly, the weights of the stage and terminal cost functions are $Q = 5I_{10}$, $R = 2I_4$, and $P_N = 100I_{10}$. No warm starting is used in any of the algorithms.

To solve this problem we apply MINFBE and NAMA and we compare with GPAD and with the L-BFGS method for the minimization of the FBE with a memory of 5.[50] The progress of these methods, in terms of the infinity norm of the fixed point residual, against their iterations is shown in Figure 3. Although L-BFGS seems to perform well in practice, note that according to Reference 60 (Thm. 5.7.4) uniform convexity of $\varphi_\lambda$ is necessary for the L-BFGS method to converge, but this condition may not be satisfied (it is not satisfied in this example).

In order to assess the how MINFBE and NAMA perform when faced with problems of different parameters—and, in particular, ill-conditioned ones—we tested with weight matrices $Q$ of different condition numbers. Figure 4 shows the total number of calls of Algorithms 4 and 5 required for convergence. Apart from the fact that MINFBE and NAMA require fewer such calls, we observe that for high condition numbers, GPAD and L-BFGS fail to converge within 5000 calls.

GPAD is known to converge at a rate of $\mathcal{O}(1/k^2)$, which can be observed in Figure 3; clearly, GPAD can only achieve low to medium accuracy solutions within a few hundred iterations. On the other hand, MINFBE and NAMA exhibit a significantly faster convergence rate and require fewer iterations to achieve solutions of higher accuracy.

We ran the stochastic optimal control problem for 300 initial states $x_0 = p$, sampled uniformly from the problem's domain. These problems were solved with NAMA, MINFBE and the accelerated projected gradient method applied to the
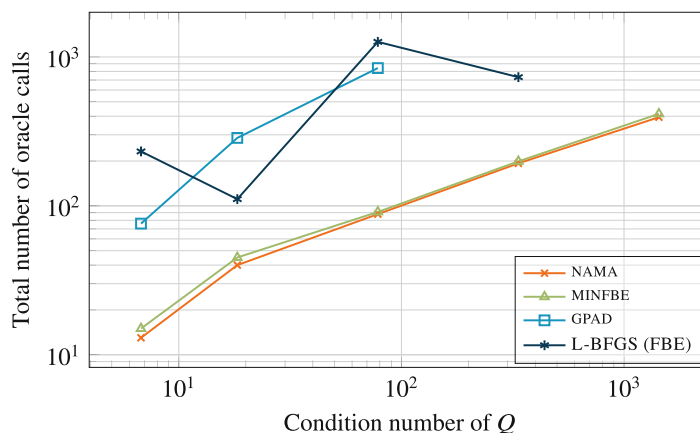
**FIGURE 4** Total number of oracle calls of Algorithms 4 and 5 necessary for convergence ($\epsilon = 10^{-4}$) against the condition number of $Q$ for NAMA, MINFBE, GPAD, and the LBFGS method applied to $\varphi_\lambda$ with a memory of 5.
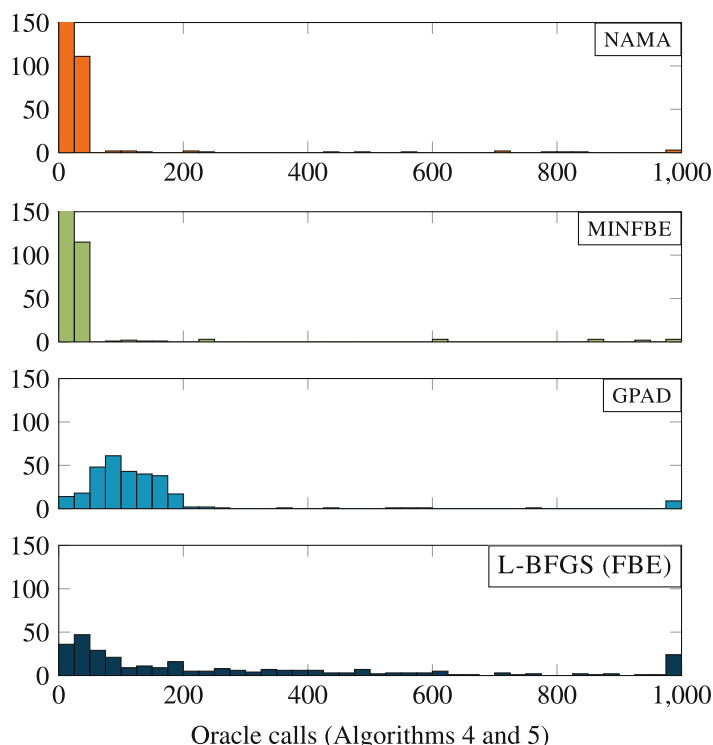


**FIGURE 5** Distribution of the number of oracle calls required for the computation of the dual gradient (Algorithm 4) and Hessian-vector products (Algorithm 5) for GPAD, MINFBE, GPAD, and the L-BFGS algorithm with memory 5 applied for the minimization of the FBE.

dual problem (GPAD) following Reference 7. In NAMA and MINFBE we used L-BFGS directions with a memory of 5. We used the same termination condition in all methods with $\epsilon = 10^{-4}$.

In Figure 5, we show the number of calls of Algorithms 4 and 5 required to solve the aforementioned collection of 300 random problems up to the desired accuracy. We may observe that in the majority of cases (84%), MINFBE and NAMA can solve the problems with no more than 50 calls, whereas the median of the number of calls corresponding to GPAD is 188. We should note that, in all 300 cases, the first control action, $u_0^\star$, obtained by the above algorithms is element-wise within 0.011% of the solution obtained using the interior point solver of Gurobi.

Note that NAMA and MINFBE appear to perform on a par. However, in the next section we will demonstrate that NAMA allows for greater parallelizability leading to superior performance on a GPU.

## 4.2 | Large-scale drinking water network

In this section, we apply the proposed numerical optimization methods for the solution of an MPC problem for a drinking water network, whose transportation dynamics is described by

$$x_{t+1} = Ax_t + Bu_t + Gd_t, \tag{43a}$$

$$0 = E_u u_t + E_d d_t, \tag{43b}$$

where $x_t$ is the vector of the volume of water in the reservoirs of the network, $u_t$ is the vector of pumping set points and $d_t$ is the vector of water demands from the various distribution nodes. The value of $d_t$ is measured at time $t$ and future demand values are predicted by a model that returns estimates $\hat{d}_{t+t'|t}$, for $t' \geq t$, while $d_{t+t'} = \hat{d}_{t+t'|t} + \epsilon_{t'}$, where $\epsilon_{t'}$ is a random process that can be described by a scenario tree.[7]

The water network model (43) comprises 63 states corresponding to water level in the tanks, 114 inputs corresponding to flow control devices (pumps and valves), 88 disturbance variables corresponding to the demand sectors and input-disturbance relationship corresponding to the 17 mixing nodes. The detailed stochastic optimal control problem and the formulation of the optimization problem is discussed in Reference 7. The operation of the water network is subject to uncertainty in water demand and electricity prices.
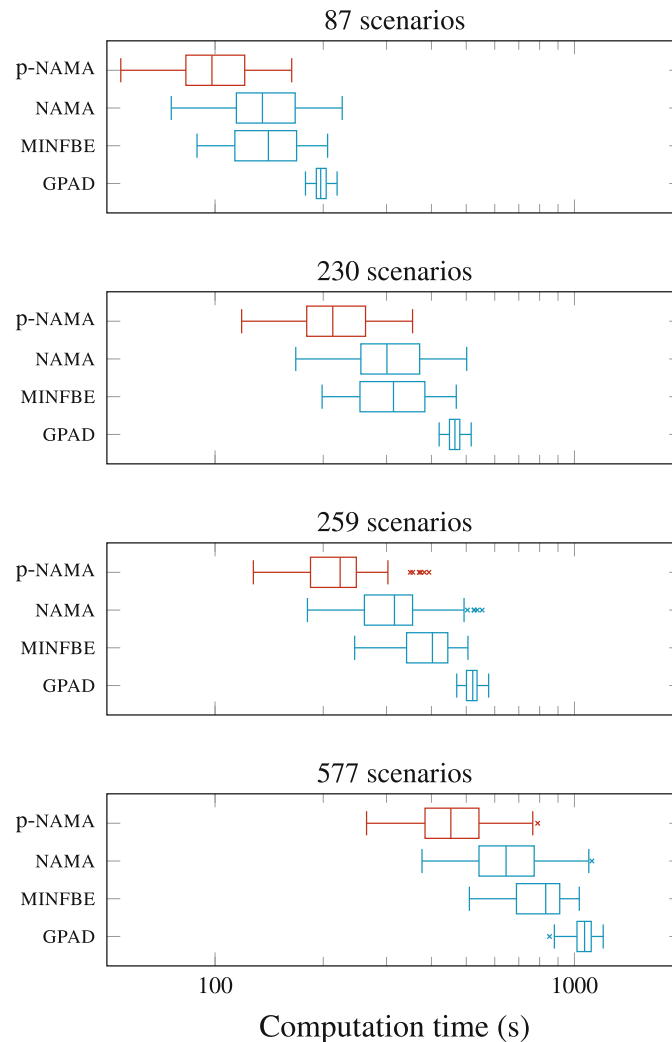


**FIGURE 6** Box plots of the computational time with different scenario tree sizes for the GPU implementation of the algorithms: (parallel) NAMA, MINFBE, and GPAD.

The NAMA and MINFBE algorithms are implemented in the `RapidNet`[*] software package that is developed for the operational control of water network problems. All simulations presented in this section were carried out on an NVIDIA Tesla C2075 GPU which counts 448 CUDA cores running at 1.15 GHz and 6 GB of dedicated memory.

In order to demonstrate the effect of the additional parallelization in NAMA that we discussed in Section 3.3, we provide results for the method with that additional parallelization in the computation of the line search (p-NAMA) and NAMA without that additional parallelization.

The parallel computations involved in Algorithms 4 and 5 are carried out using cuBLAS's `cublasSgemmBatched` and `cublasSgemm`. In this example, matrices $A$, $B$, $G_d$, $E_u$, and $E_d$ are sparse, and this has been used to tailor the implementations of Algorithms 4 and 5 to be more efficient.

The L-BFGS memory is set to 15. In the case with 577 scenarios, the problem involves 2.1 million primal and 3.8 million dual variables and NAMA and MINFBE algorithms require an excess of 2.9% (172 MB) of memory and p-NAMA requires an excess of 4.1% (242 MB) of memory than dual accelerated proximal gradient (GPAD) algorithm. The solve times of p-NAMA, NAMA, MINFBE and GPAD are shown in Figure 6, where note that the horizontal axis is logarithmic. It can be observed that p-NAMA is noticeably faster compared to NAMA, MINFBE and GPAD. Lastly, the (element-wise) difference between the solutions obtained by the above algorithms is in all cases within 0.01% of the solution that was obtained using Gurobi.

## 5 | CONCLUSIONS

In this article, we proposed the use of MINFBE and NAMA for solving large-scale scenario-based convex stochastic optimal control problems. Both methods use limited-memory quasi-Newtonian, L-BFGS, directions and exhibit a very fast convergence rate. They are both suitable for parallelization on GPUs, but NAMA lends itself to a significantly higher parallelization. We presented compelling results on two stochastic optimal control problems, namely a spring-mass-damper array and the drinking water network of Barcelona, demonstrating that the two methods significantly outperform GPAD, whose parallelizable implementation on a GPU has been previously shown to outperform Gurobi's interior point solver.[7] Future work will focus on the development of parallelizable methods for large-scale scenario-based risk-averse optimal control problems.[67]

### AUTHOR CONTRIBUTIONS
**All authors** Contributed equally to the development of the methodology, revised and proofread the article. **Ajay K. Sampathirao** and **Pantelis Sopasakis:** Software development of RapidNet and simulations. **Pantelis Sopasakis** and **Ajay Sampathirao:** Contributed to writing this article.

### DATA AVAILABILITY STATEMENT
The data that support the findings of this study are openly available on the GitHub repository of `RapidNet`[66] at https://doi.org/10.5281/zenodo.7072936.

### ENDNOTE
[*]https://github.com/GPUEngineering/RapidNet[66]

### ORCID
*Alberto Bemporad* https://orcid.org/0000-0001-6761-0856
*Pantelis Sopasakis* https://orcid.org/0000-0002-4584-2354

### REFERENCES
1. Patrinos P, Sopasakis P, Sarimveis H, Bemporad A. Stochastic model predictive control for constrained discrete-time Markovian switching systems. *Automatica*. 2014;50(10):2504-2514.

2. Chatterjee D, Lygeros J. On stability and performance of stochastic predictive control techniques. *IEEE Trans Automat Contr*. 2015;60(2):509-514.

3. Li L, You S, Yang C, Yan B, Song J, Chen Z. Driving-behavior-aware stochastic model predictive control for plug-in hybrid electric buses. *Appl Energy*. 2016;162:868-879.

4. Cairano SD, Bernardini D, Bemporad A, Kolmanovsky IV. Stochastic MPC with learning for driver-predictive vehicle control and its application to HEV energy management. *IEEE Trans Control Syst Technol*. 2014;22(3):1018-1031.

5. Darivianakis G, Georghiou A, Smith RS, Lygeros J. The power of diversity: data-driven robust predictive control for energy-efficient buildings and districts. *IEEE Trans Control Syst Technol*. 2019;27(1):132-145.

6. Hans C, Sopasakis P, Bemporad A, Raisch J, Reincke-Collon C. Scenario-based model predictive operation control of islanded microgrids. *2015 54th IEEE Conference on Decision and Control*. IEEE; 2015:3272-3277.

7. Sampathirao AK, Sopasakis P, Bemporad A, Patrinos P. GPU-accelerated stochastic predictive control of drinking water networks. *IEEE Trans Control Syst Technol*. 2018;26(2):551-562.

8. Zidek RA, Kolmanovsky IV, Bemporad A. Model predictive control for drift counteraction of stochastic constrained linear systems. *Automatica*. 2021;123:109304.

9. Freitas Virgilio Pereira M, Kolmanovsky IV, Cesnik CE, Vetrano F. *Time-distributed scenario-based model predictive control approach for flexible aircraft*. American Institute of Aeronautics and Astronautics; 2021.

10. de la Penad D, Bemporad A, Alamo T. Stochastic programming applied to model predictive control. *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE; 2005:1361-1366.

11. Zverovich V, Fábián CI, Ellison EFD, Mitra G. A computational study of a solver system for processing two-stage stochastic LPs with enhanced benders decomposition. *Math Program Comput*. 2012;4(3):211-238.

12. Egging R. Benders decomposition for multi-stage stochastic mixed complementarity problems—applied to a global natural gas market model. *Eur J Oper Res*. 2013;226(2):341-353.

13. Shapiro A. Analysis of stochastic dual dynamic programming method. *Eur J Oper Res*. 2011;209(1):63-72.

14. Akian M, Chancelier JP, Tran B. A min-plus-SDDP algorithm for deterministic multistage convex programming. *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE; 2019:3334-3339.

15. Rockafellar RT, Wets RJB. Stochastic variational inequalities: single-stage to multistage. *Math Program*. 2016;165(1):331-360.

16. Rockafellar RT, Wets RJB. Scenarios and policy aggregation in optimization under uncertainty. *Math Oper Res*. 1991;16(1):119-147.

17. Sun J, Xu H, Zhang M. A new interpretation of the progressive hedging algorithm for multistage stochastic minimization problems. *J Ind Manag Optim*. 2020;16(4):1655-1662.

18. Veliz FB, Watson JP, Weintraub A, Wets R, Woodruff D. Stochastic optimization models in forest planning: a progressive hedging solution approach. *Ann Oper Res*. 2014;232:259-274. doi:10.1007/s10479-014-1608-4

19. Rockafellar RT. Progressive decoupling of linkages in optimization and variational inequalities with elicitable convexity or monotonicity. *Set Valued Var Anal*. 2018;27(4):863-893.

20. Medina-Gonázlez S, Gkioulekas I, Dua V, Papageorgiou LG. A graph theory approach for scenario aggregation for stochastic optimisation. *Comput Chem Eng*. 2020;137:106810.

21. Shen X, Boyd S. Incremental proximal multi-forecast model predictive control; 2021. https://arxiv.org/abs/2111.14728

22. Anish DJ. Bitcoin mining acceleration and performance quantification. *2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE; 2014:1-6.

23. Aubert D. Numerical cosmology powered by GPUs. *Proc Int Astron Union*. 2010;6(S270):397-400.

24. Eklund A, Dufort P, Forsberg D, LaConte SM. Medical image processing on the GPU—past, present and future. *Med Image Anal*. 2013;17(8):1073-1094.

25. Le Grand S, Götz AW, Walker RC. SPFP: speed without compromise—a mixed precision model for GPU accelerated molecular dynamics simulations. *Comput Phys Commun*. 2013;184(2):374-380.

26. Kim H, Nam H, Jung W, Lee J. Performance analysis of CNN frameworks for GPUs. *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE; 2017:55-64.

27. Vingelmann P, Fitzek FH. CUDA, Release: 10.2.89; 2020. https://developer.nvidia.com/cuda-toolkit

28. Cook S. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. 1st ed. Morgan Kaufmann Publishers Inc.; 2012.

29. Abadi M, Agarwal A. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems; 2015. https://www.tensorflow.org/

30. Jia Y, Shelhamer E, Donahue J, et al. Caffe: convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093, 2014.

31. Hartley EN, Maciejowski JM. Field programmable gate array based predictive control system for spacecraft rendezvous in elliptical orbits. *Optim Control Appl Methods*. 2014;36(5):585-607. doi:10.1002/oca.2117

32. Ling K, Wu B, Maciejowski J. Embedded model predictive control (MPC) using a FPGA. *IFAC Proc Vol*. 2008;41(2):15250-15255.

33. Jerez JL, Goulart PJ, Richter S, Constantinides GA, Kerrigan EC, Morari M. Embedded online optimization for model predictive control at megahertz rates. *IEEE Trans Automat Contr*. 2014;59(12):3238-3251. doi:10.1109/TAC.2014.2351991

34. Dang TV, Ling K, Maciejowski J. Embedded ADMM-based QP solver for MPC with polytopic constraints. *2015 European Control Conference (ECC)*. IEEE; 2015:3446-3451.

35. Rubagotti M, Patrinos P, Guiggiani A, Bemporad A. Real-time model predictive control based on dual gradient projection: theory and fixed-point FPGA implementation. *Int J Robust Nonlinear Control*. 2016;26(15):3292-3310.

36. Wills AG, Knagge G, Ninness B. Fast linear model predictive control via custom integrated circuit architecture. *IEEE Trans Control Syst Technol*. 2012;20(1):59-71. doi:10.1109/TCST.2010.2096224

37. Gade-Nielsen N. *Interior Point Methods on GPU with Application to Model Predictive Control*. Ph.D. thesis. DTU Compute; 2014.
38. Yu L, Goldsmith A, Cairano SD. Efficient convex optimization on GPUs for embedded model predictive control. *Proceedings of the General Purpose GPUs on - GPGPU-10*. Association for Computing Machinery; 2017.
39. Fang CH, Kylasa S, Roosta F, Mahoney M, Grama A. Newton-ADMM: a distributed GPU-accelerated optimizer for multiclass classification problems. arXiv preprint arXiv:1807.07132, 2020.
40. Enfedaque P, Chang H, Krishnan H, Marchesini S. GPU-based implementation of Ptycho-ADMM for high performance x-ray imaging. In: Shi Y, Fu H, Tian Y, et al., eds. *Science—ICCS 2018*. Springer; 2018:540-553.
41. Qureshi Z, East S, Cannon M. Parallel ADMM for robust quadratic optimal resource allocation problems. *2019 American Control Conference (ACC)*. IEEE; 2019:3402-3407.
42. Gaetano R, Chierchia G, Pesquet-Popescu B. Parallel implementations of a disparity estimation algorithm based on a proximal splitting method. *2012 Visual Communications and Image Processing*. IEEE; 2012:1-6.
43. Schubiger M, Banjac G, Lygeros J. GPU acceleration of ADMM for large-scale quadratic programming. *J Parallel Distrib Comput*. 2020;144:55-67.
44. Deng W, Lai MJ, Peng Z, Yin W. Parallel multi-block ADMM with $\mathcal{O}(1/k)$ convergence. *J Sci Comput*. 2016;71(2):712-736.
45. Kouzoupis D, Klintberg E, Frison G, Gros S, Diehl M. A dual Newton strategy for tree-sparse quadratic programs and its implementation in the open-source software treeQP. *Int J Robust Nonlinear Control*. 2019;29(8):2438-2457.
46. Sopasakis P, Sampathirao AK, Bemporad A, Patrinos P. Uncertainty-aware demand management of water distribution networks in deregulated energy markets. *Environ Model Software*. 2018;101:10-22.
47. Sampathirao A, Sopasakis P, Bemporad A, Patrinos P. Distributed solution of stochastic optimal control problems on GPUs. *2015 54th IEEE Conference on Decision and Control*. IEEE; 2015:7183-7188.
48. Stella L, Themelis A, Patrinos P. Forward-backward quasi-Newton methods for nonsmooth optimization problems. *Comput Optim Appl*. 2017;67(3):443-487.
49. Stella L, Themelis A, Patrinos P. Newton-type alternating minimization algorithm for convex optimization. *IEEE Trans Automat Contr*. 2019;64(2):697-711.
50. Sampathirao A, Sopasakis P, Bemporad A, Patrinos P. Proximal limited-memory quasi-Newton methods for scenario-based stochastic optimal control. *IFAC-Pap*. 2017;50(1):11865-11870.
51. Combettes PL, Pesquet JC. Proximal splitting methods in signal processing. In: Bauschke H, Burachik R, Combettes P, Elser V, Luke D, Wolkowicz H, eds. *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*. Springer; 2011:185-212.
52. Bemporad A, Breschi V, Piga D, Boyd SP. Fitting jump models. *Automatica*. 2018;96:11-21.
53. Frison G. *Numerical Methods For Model Predictive Control*. Master's thesis. Università degli studi di Padova, Facoltà di Ingegneria; 2012.
54. Rockafellar R, Wets R. *Variational Analysis*. Grundlehren der Mathematischen Wissenschaften. Vol 317. Springer-Verlag; 1998.
55. Bauschke H, Combettes P. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer; 2011.
56. Parikh N, Boyd S. Proximal algorithms. *Found Trends Optim*. 2013;1(3):123-231.
57. Patrinos P, Stella L, Bemporad A. Forward-backward truncated newton methods for convex composite optimization; 2014. http://arxiv.org/abs/1402.6655
58. Patrinos P, Bemporad A. Proximal Newton methods for convex composite optimization. *52nd IEEE Conference on Decision and Control*. IEEE; 2013:2358-2363.
59. Gorni G. Conjugation and second-order properties of convex functions. *J Math Anal Appl*. 1991;158(2):293-315.
60. Sun W, Yuan YX. *Optimization Theory and Methods: Nonlinear Programming*. Springer; 2006.
61. Li DH, Fukushima M. On the global convergence of the BFGS method for nonconvex unconstrained optimization problems. *SIAM J Optim*. 2001;11(4):1054-1064.
62. Rockafellar R. *Convex Analysis*. Princeton University Press; 1976.
63. Cruz JYB, Nghia TT. On the convergence of the forward–backward splitting method with linesearches. *Optim Methods Softw*. 2016;31(6):1209-1238.
64. Giselsson P, Boyd S. Metric selection in fast dual forward–backward splitting. *Automatica*. 2015;62:1-10.
65. Patrinos P, Bemporad A. An accelerated dual gradient-projection algorithm for embedded linear model predictive control. *IEEE Trans Automat Contr*. 2014;59(1):18-33.
66. Sampathirao A, Sopasakis P. GPUEngineering/RapidNet: RapidNet 1.0; 2022.
67. Sopasakis P, Herceg D, Bemporad A, Patrinos P. Risk-averse model predictive control. *Automatica*. 2019;100:281-288.