Computational Management Science (2024) 21:31
https://doi.org/10.1007/s10287-024-00517-y

ORIGINAL PAPER

Hybrid simplicial-randomized approximate stochastic dynamic programming for multireservoir optimization

Luckny Zephyr[1] · **Bernard F. Lamond**[2] · **Pascal Lang**[2]

Received: 17 December 2023 / Accepted: 22 April 2024 / Published online: 10 May 2024

Abstract

We revisit an approximate stochastic dynamic programming method that we proposed earlier for the optimization of multireservoir problems. The method exploited the convexity properties of the value function to sample the reservoir level space based on the local curvature of the value function, which is estimated by the difference between a lower and an upper bounds (error bound). Unlike the previous approach in which the state space is exhaustively partitioned into full dimensional simplices whose vertices formed a discrete grid over which the value function was approximated, here we propose instead a new randomized approach for selecting the grid points from a small number of randomly sampled simplices from which an error bound is estimated. Results of numerical experiments on three literature test problems and simulated midterm reservoir optimization problems illustrate the advantages of the randomized approach which can solve models of higher dimensions than with the exhaustive approach.

Keywords Reservoir optimization · Stochastic dynamic programming · Simplicial-randomized approximation · Piecewise linear approximation

✉ Luckny Zephyr
lzephyr@laurentian.ca

Bernard F. Lamond
Bernard.Lamond@fsa.ulaval.ca

Pascal Lang
pascal.lang.1@ulaval.ca

[1] School of Business Administration/Barthi School of Engineering and Computer Science (Cross-Appointment), Laurentian University, Sudbury, ON P3E 2C6, Canada

[2] Département Opérations et systèmes de décision, Université Laval, Québec G1V 0A6, Canada

1 Introduction

This work deals with a mid-term reservoir optimization problem over a finite planning horizon. In each period, water must be released from the reservoirs to produce electricity. However, the release decisions are constrained by not only the availability of water, but also the physical limits of the turbines, and bounds on the level of the reservoirs, that may be set by legal requirements. This problem is rightfully acknowledged to be difficult, in particular due to the uncertainty associated with the natural inflows to the reservoirs, e.g., snow-melt, snow water equivalent.

Thus, the mid-term reservoir optimization is inherently a multiperiod stochastic problem. As a result, the problem is often cast as a multiperiod stochastic program or formulated under the framework of stochastic dynamic programming. Numerous meta-heuristic approaches have also been proposed for reservoir optimization problems, e.g., Almubaidin et al. (2022). Two recent systematic reviews of such methods are available in Azad et al. (2020), Beiranvand and Ashofteh (2023).

When stochastic programming (SP) is employed to solve the problem, the random variables, e.g., natural inflows, and demand for energy, are discretized via a so-called scenario tree, which easily becomes intractable if a detailed representation of the stochastic variables is needed. This issue is often dealt with through decomposition strategies, such as Benders' decomposition, e.g., Carpentier et al. (2014), Rebennack (2016), the progressive hedging algorithm, e.g., Gonçalves et al. (2012), Carpentier et al. (2013), Zéphyr et al. (2014), in which the so-called non-anticipativity constraints are dualized in the objective function, stochastic dynamic programming (SDP) (Ruszczyński and Shapiro 2003; Shapiro et al. 2009), scenario tree reduction strategies (Dupačová et al. 2003; Bin et al. 2015), model predictive control, e.g., Nolde et al. (2008), Uysal et al. (2018), Lin et al. (2020), etc.

Being a sequential decision-making problem, the mid-term optimization of reservoir lends itself naturally to stochastic dynamic programming (SDP). Indeed, in the groundbreaking theory of dynamic programming presented in Bellman (1958), Bellman decomposed a *multi-stage decision process* stagewise in a coordinated manner. Thus, it is no surprise that DP quickly found a fertile ground for reservoir optimization applications (Labadie 2004).

The solution of SP or SDP reservoir management problems broadly consists of two main steps, namely (i) the calculation of an expectation; and (ii) an optimization step, or vice-versa. In models for the mid- or long-term planning of hydroelectric production, the optimization step often has to deal with nonlinear objective functions, due to, among others, nonlinear production functions (Cerisola et al. 2012). To take advantage of the widespread availability of linear programming solvers, the combined power response curve of the turbines at a power plant can often be approximated reasonably well by a concave, piecewise linear function of turbined water flow, even though the response curves of the individual turbines may be highly nonlinear. For instance, this strategy is used by companies like

Hydro-Quebec (Carpentier et al. 2013) and Rio Tinto (Côté and Arsenault 2019) (4-reservoir system) to approximate production functions; similarly in studies on the Colombian power network (Morillo et al. 2020) (15-reservoir system), on a "network of hydropower plants and irrigated areas in the Nile Basin" (Goor et al. 2011), the Brazilian electrical system (Diniz and Maceira 2008) (110 hydro plants). An immediate consequence of this approximation scheme is that under mild assumptions on the terminal value/cost-to-go function, one can easily show that the value/cost-to-go functions are concave/convex in the reservoir levels. These ideas are also exploited in Zéphyr et al. (2015), Zéphyr et al. (2017) where an approximate stochastic dynamic programming model of a multiperiod, multireservoir hydroelectric system is presented in which the Bellman value function is approximated by a piecewise linear function that is evaluated by linear programming. The piecewise linear approximation is supported by a finite grid of node points (or vertices) in the continuous state space where the Bellman function is evaluated at the nodes. Similarly, using a finite grid for the reservoir levels, Dias et al. (2010) approximates the expected value of the Bellman value function with a piecewise linear function, generated by a set of hyperplanes using a convex hull algorithm. The latter approach is applied to a power generation planning problem in Dias et al. (2013).

Resorting to SDP to solve reservoir optimization problems poses another technical challenge, since in theory an optimization problem has to be solved for each possible state value, which is impossible due to the fact that the reservoir level space is continuous. Thus, the latter must be discretized or sampled. The simplest discretization strategy to approximate our continuous dynamic program consists in constructing a uniform grid, obtained as the Cartesian product of same-size and fixed-spacing grids along each dimension of the reservoir level (state) space. However, this approach is impractical, as the complexity of the problem increases exponentially with the dimension of the state space, limiting applications to three to four reservoirs. This is known in dynamic programming as the *curse of dimensionality*.

The above uniform discretization scheme has inspired the development of parsimonious approaches that select sub-samples of points along each dimension of the state space, and then use analytical functions based on multi-linear interpolations, polynomials, cubic splines, to approximate the Bellman function (Johnson et al. 1993). As these techniques did not prove to be a panacea against the dimensionality issue, statistical techniques have been employed to sample the state space more efficiently. Perhaps, one of the oldest strategies is *Latin hypercube*, in which each dimension of the state space is discretized into p values, and the overall sample is chosen so that each uni-dimensional value is selected exactly once. This is a special case of *orthogonal array with strength* d, where $d \leq n$, n being the dimension of the state space. Under this scheme, each uni-dimensional grid point is chosen exactly a same number of times in each possible d-dimensional subspace (Chen et al. 1999).

Other sampling techniques resort to some form of Monte Carlo simulation to sample the state space in contrast to the discretization strategies used in the above-mentionned schemes. For instance, in stochastic dual dynamic programming

(SDDP), originally developed for reservoir optimization problems in the seminal works (Pereira and Pinto 1985, 1991), the connections between SP and SDP, e.g., Ruszczyński and Shapiro (2003), Shapiro et al. (2009), are exploited to efficiently sample the reservoir level space, based on Monte Carlo simulation. Assuming the natural inflows to be temporally independent, SDDP alternates between a backward pass, to build the so-called *value/cost-to-go functions*, and a forward step, to draw a sample of state space values to approximate the value/cost-to-go functions in the next backward loop, until a convergence criterion is met. In contrast with classical SDP, where the state space is distretized into an evenly spaced grid, in SDDP, the sate space is iteratively sampled by simulating trajectories of reservoir levels through the forward passes, thus mitigating the inherent curse of dimensionality of SDP. By thus sampling the state space, SDDP selects grid points in regions of operations of the reservoirs, in contrast with classical SDP, where the state space is uniformly discretized.

On the other hand, quasi-randomized or quasi-Monte Carlo sampling techniques, where randomly generated points are replaced with more evenly distributed ones, based on the notion of *low-discrepancy sequences*, are known to enjoy faster convergence rate than randomized techniques (Cervellera et al. 2013; Cervellera and Muselli 2007). For further account of reservoir optimization techniques, please see Labadie (2004), Rani and Moreira (2010), Ahmad et al. (2014) and Dobson et al. (2019).

In Zéphyr et al. (2015), we proposed a simplicial approximate SDP approach for the mid-term optimization of reservoirs. While the previous work laid the foundation of the approach, in Zéphyr et al. (2017), we extended the methodology to the optimization of multireservoir systems with highly correlated natural inflows, in which the support of the random variable reduces from n to 1. The simplicial approach was exploited to derive an analytical form for the expected value of the value function under the assumptions that the natural inflows follow a truncated normal or a log-normal distribution.

The iterative scheme amounts to partitioning the reservoir level space into a finite but potentially large set of simplices in each period of the planning horizon. The value function is evaluated at the extreme points of the resulting simplices, and interpolated elsewhere. In addition, error bounds are computed for all simplices and, at each iteration, a new grid point associated with the largest error bound is added to the grid, and the simplex containing the point is divided into smaller simplices that are appended to the list of existing simplices. Thus, in each period, constructing the grid requires to maintain a complete list of simplices that spans the whole reservoir level space. Because the number of simplices increases fast with the grid size and with the dimension of the state space, this method becomes impractical for models with many reservoirs.

This work is essentially a revisit of the sampling approach presented in Zéphyr et al. (2015), in which, in each period, we avoid making a list of simplices and randomly sample the reservoir level space to select grid points at which the value function is approximated. We resort to linear programming to identify the simplex containing a candidate grid point and to obtain a local error bound on the approximation of the Bellman function. Then, the global error bound is estimated using a statistical model. This is motivated by the computational burden of the simplicial scheme, induced by the exponential growth of the number of created simplices, which limits applications to dimensions lower than ten, based on our empirical observations.

The remainder of the paper is organized as follows. We provide a detailed description of the problem under analysis in Sect. 2. Next, we discuss a simplicial approximate stochastic dynamic programming (ASDP) scheme for the problem in Sect. 3, followed by a hybrid Monte Carlo simplicial ASDP proposal in Sect. 4. Results of extensive numerical experiments are reported in Sect. 5. The paper ends with concluding remarks in Sect. 6.

2 Reservoir optimization problem

A hydropower system often comprises power plants that may or may not be associated with reservoirs. Reservoir optimization problems are typically divided into long-, mid-, and short-term, depending on, among other factors, the length of the planning horizon (Raso and Malaterre 2017). In a mid-term problem, which is of interest to us, the time span is typically between one and five years (van Ackooij et al. 2014), divided into daily, weekly, or monthly time steps (Zéphyr et al. 2017).

In this work, we consider a mid-term reservoir optimization problem over a finite horizon of T periods. At each period t, the operator of the system wants to find the release, $\boldsymbol{u}_t$, and storage, $\boldsymbol{s}_t$, decisions that maximize the expected total energy production. Without loss of generality, we assume each plant to be associated with a reservoir; the random natural inflows to the reservoirs are denoted $\tilde{\boldsymbol{q}}_t$.

At each period t, water released from each reservoir $i = 1, \ldots, n$, is limited by the turbine capacity, $\overline{\boldsymbol{u}}$, to prevent physical damage. Similarly, due to legal and environmental considerations, at each time period, the level of the reservoirs must be kept between lower and upper limits, $\underline{\boldsymbol{s}}$, and $\overline{\boldsymbol{s}}$, respectively.

In addition, we assume the topology of the system to form an arborescence, i.e., a combination of reservoirs in series and in parallel. Water released upstream are absorbed by the immediate successors (reservoirs) at the same period, and in case of overflow, excess of water from upstream reservoirs, $\boldsymbol{y}_t$, are absorbed by immediate successors or spilled out of the system.

At each period t, the state of the system is governed by the standard mass balance equation:

$$\boldsymbol{s}_t = \boldsymbol{s}_{t-1} - \boldsymbol{B}\boldsymbol{u}_t - \boldsymbol{C}\boldsymbol{y}_t + \tilde{\boldsymbol{q}}_t, \tag{1}$$

where entries of the square connectivity matrix, B_{ij}, are 1 for $i = j$, -1 if the water released from reservoir j is routed to reservoir i, and 0, otherwise. The elements of the square matrix $\boldsymbol{C}$ similarly define the routing of the spilled water.

As in Zéphyr et al. (2015), for each plant $i = 1, \dots, n$, we assume the production function p_{it} to nonlinearly depend on the release and the storage at the beginning of the period.

A typical multi-period mid-term reservoir optimization problem reads:

$$\max_{\boldsymbol{u}_t, \boldsymbol{y}_t} \mathbb{E}_{\tilde{\boldsymbol{q}}_t} \left[\sum_{t=1}^{T} \sum_{i=1}^{n} p_{it}(u_{it}) + V_{T+1}(\boldsymbol{s}_{T+1}) \right] \tag{2}$$

$$\text{s.t., for } t = 1, \dots, T: \tag{3}$$

$$\boldsymbol{s}_{t+1} = \boldsymbol{s}_t - \boldsymbol{B}\boldsymbol{u}_t - \boldsymbol{C}\boldsymbol{y}_t + \tilde{\boldsymbol{q}}_t \tag{4}$$

$$\underline{\boldsymbol{s}} \le \boldsymbol{s}_{t+1} \le \bar{\boldsymbol{s}} \tag{5}$$

$$0 \le \boldsymbol{u}_t \le \bar{\boldsymbol{u}} \tag{6}$$

$$\boldsymbol{y}_t \ge \boldsymbol{0}, \tag{7}$$

where, $\mathbb{E}$ is the expectation operator, and $V_{T+1}(\boldsymbol{s}_{T+1})$, assumed to be a concave function, captures the terminal value of the stored water in the system.

At each time period t, assume the operator of the system observes the level of the reservoirs, the realization $\boldsymbol{q}_t$ of the random natural inflows, $\tilde{\boldsymbol{q}}_t$, and decides on the water released, spilled and stored to find the best trade-off between utilizing the available water for current production needs and leaving it for the future. Under this setting, and by Bellman's principle of optimality, Problem (2)-(7) can be reformulated as a sequence of coordinated subproblems, moving backward in time, i.e., for $t = T, T-1, \dots, 1$,

$$V_t\left(\boldsymbol{s}_t, \boldsymbol{q}_t\right) := \max_{\boldsymbol{u}_t, \boldsymbol{y}_t} \left\{ \sum_{i=1}^{n} p_{it}(u_{it}) + \mathcal{V}_{t+1}\left(\boldsymbol{s}_{t+1}, \tilde{\boldsymbol{q}}_{t+1}\right) \right\} \tag{8}$$

$$\text{s.t. } (4)–(7), \tag{9}$$

where $V_t(\cdot)$, called value function, measures the value of the stored water from period t onward, and $\mathcal{V}_{t+1}(\cdot) := \mathbb{E}_{\tilde{\boldsymbol{q}}_{t+1}|\boldsymbol{q}_t} V_{t+1}\left(\boldsymbol{s}_{T+1}, \tilde{\boldsymbol{q}}_{t+1}\right)$. As in Zéphyr et al. (2017), since the terminal value function is concave, we observe that if the production functions are concave, the problem is convex and the concavity of the value function $V_t(\boldsymbol{s}_t, \cdot)$ propagates backwards.

Proposition 1 *If* (*i*) *$p_{it}(u_{it})$ is concave in u_{it}, and* (*ii*) *the support of $\tilde{\boldsymbol{q}}_t$ is discrete and finite, then $V_t(\boldsymbol{s}_t, \cdot)$ is concave in $\boldsymbol{s}_t$.*

Proof The feasible domain of Problem (8)–(9) is a polyhedron; since $V_{T+1}(\boldsymbol{s}_{T+1},\cdot)$ is concave in $\boldsymbol{s}_{T+1}$, by the concavity of the production function, and the linearity property of the expectation operator, it follows that $V_T(\boldsymbol{s}_T,\cdot)$ is concave in $\boldsymbol{s}_T$. The concavity property then follows by backward induction on t, for $t = T-1,\dots,1$. □

Problem (8)–(9) may be nonlinear, in particular due to the nonlinearity of the production functions. Indeed, in practice, production functions are often nonconcave (i) due to head effects, i.e, the difference between upstream and downstream reservoir levels; and (ii) because the power produced by a plant varies nonlinearly with the water release and the number of turbines, whose efficiency may decrease beyond a maximum flow rate (Zéphyr et al. 2017). In industry, this issue is often dealt with by approximating production functions with their concave envelopes (e.g., Goor et al. 2011; Carpentier et al. 2013; Côté and Arsenault 2019; Morillo et al. 2020).

As in Zéphyr et al. (2017), the nonlinearity hurdle is passed using inner generalized linear programming (GLP) on a support grid to obtain a convex approximation of the problem. For each plant i, assume that the production function is evaluated over a finite grid of reservoir releases $\mathcal{U}_t := \{u_i^k | k \in K_i\}$, constructed in a preprocessing step, where K_i is the set of indices associated with the discrete releases u_i^k, $i = 1,\dots,n$. Similarly, the expected value function $\mathcal{V}_{t+1}(\cdot)$ is evaluated over a finite set of states $\mathcal{G}_t := \{\boldsymbol{s}_{t+1}^j | j \in J_t\}$, where J_t is the set of indices associated with the discrete storage vectors $\boldsymbol{s}_{t+1}^j$, possibly obtained by division of simplices as explained in Sect. 3. The following GLP is a linear approximation of Problem (8)–(9):

$$\hat{V}_t(\boldsymbol{s}_t,\boldsymbol{q}_t) := \max_{\boldsymbol{u}_t,\boldsymbol{y}_t,\boldsymbol{\lambda},\boldsymbol{\mu}} \left\{ \sum_{i=1}^{n} \sum_{k\in K_i} p_{it}(u_i^k)\lambda_i^k + \sum_{j\in J_t} \hat{\mathcal{V}}_{t+1}\left(\boldsymbol{s}_{t+1}^j,\cdot\right)\mu^j \right\} \tag{10}$$

$$\text{s.t. } (4)\text{–}(7) \tag{11}$$

$$u_{it} - \sum_{k\in K_i} \lambda_i^k u_i^k = 0, \quad i = 1,\dots,n \tag{12}$$

$$\boldsymbol{s}_{t+1} - \sum_{j\in J_t} \mu_j \boldsymbol{s}_{t+1}^j = 0 \tag{13}$$

$$\sum_{k\in K_i} \lambda_i^k = 1, \quad i = 1,\dots,n \tag{14}$$

$$\sum_{j\in J_t} \mu^j = 1 \tag{15}$$

$$\lambda, \boldsymbol{\mu} \geq 0 \tag{16}$$

Note that λ and $\boldsymbol{\mu}$ are vectors of convex combination coefficients, as expressed in equations (12)-(16). Thus, for each power plant i, in each period, the release is interpolated on the discrete release values; similarly the next period storage level is interpolated on the storage grid.

Since the calculation of the expected value is not the focus of this work, we assume the natural inflow process to be finite, and serially independent. As a result, in the numerical experiments, in each period, we will use Monte Carlo simulation to generate a finite sample of natural inflows, and the expected value of the approximate value function, $\hat{V}_{t+1}(\cdot)$, will be estimated by the sample mean of the $\hat{V}_{t+1}(\boldsymbol{s}_t, \boldsymbol{q}_t)$'s. Similarly, at each time period, for a given state point $\boldsymbol{s}_t^k$, let $\boldsymbol{\pi}_t^j$ be a vector of optimal dual prices associated with the mass-balance constraints (1), for a given observation $\boldsymbol{q}_t^j, j = 1, \dots, J$. In the sequel, a vector of subgradient, $\boldsymbol{g}_t^k$, will be taken as the sample mean of the $\boldsymbol{\pi}_t^j$'s.

In closing this section, observe that since (i) Problem (10)–(16) is linear and its objective maximized; and (ii) s_t is in the right hand side of the water-balance constraint (4), therefore the GLP is a parametric linear program, so that its optimal value function $\hat{V}_t(\boldsymbol{s}_t, \boldsymbol{q}_t)$ is a piecewise linear concave function of $\boldsymbol{s}_t$.

3 Simplicial approximate stochastic dynamic programming

Despite its theoretical elegance, it is well known that dynamic programming is plagued by the so-called *curse of dimensionality*, in the sense that the computational burden of Problem (10)–(16) increases exponentially with the dimension, n, of the reservoir level space S_t, except for rare cases (e.g.,unconstrained linear systems with quadratic production functions), for which analytical solutions can be derived easily. As a result, the problem cannot be solved for all possible reservoir level vectors; thus, we have to resort to some numerical procedure. To tackle the curse of dimensionality, in each time period t, we need to select a sample of discrete state vectors $\mathcal{G}_t := \left\{ \boldsymbol{s}_t^j \in S_t, j = 1, 2, \dots, m \right\}$, $t = T, T-1, \dots 1$. As discussed earlier, popular sampling techniques include Monte Carlo simulation (Chen et al. 2020; Morillo et al. 2020; Zéphyr and Anderson 2018; Morillo et al. 2017; De Matos et al. 2015), quasi-Monte Carlo simulation (Cervellera et al. 2013; Alessandri et al. 2010; Mello et al. 2011), Latin hypercube (Feng et al. 2020; Mello et al. 2011), orthogonal arrays (Feng et al. 2017; Chen 1999).

In our context, the state space is defined by the level of the reservoirs, which is confined within the hyperrectangle $S_t := \{\boldsymbol{s}_t \in \boldsymbol{R}^n \mid \underline{\boldsymbol{s}} \leq \boldsymbol{s}_t \leq \bar{\boldsymbol{s}}\}$, as defined by the box constraint (5). As a result, the state space is continuous, and as aforementioned, the approximate value function (10)–(16) cannot be evaluated for all possible pairs $(\boldsymbol{s}_t, \boldsymbol{q}_t)$. Therefore, we have to resort to some form of discretization or sampling of the state space S_t.

Under a simplicial approximate stochastic dynamic scheme, the set S_t is iteratively partitioned into smaller *convex* subsets, called *simplices*, and the approximate value function (10)–(16) is evaluated at their vertices, or extreme points.

Simplicial partitioning of convex sets is widespread in the global optimization literature (e.g., Gimbutas and Žilinskas 2018; Žilinskas and Žilinskas 2002; Paulavičius and Žilinskas 2014, 2009; Horst 1976; Tuy 1991; Bomze and Eichfelder 2013), and less popular in the field of dynamic programming (e.g., Zéphyr et al. 2017, 2015; Habets et al. 2006; Yershov and LaValle 2012; Sala and Armesto 2022). Perhaps simplicial partitioning has received a lot of attention in global optimization as a simplex is an n-dimensional polyhedron with "the minimal number of vertices", at which the function is evaluated (Paulavičius and Žilinskas 2009).

In our previous work (Zéphyr et al. 2015), that we revisit here, we iteratively sampled the state space based on the curvature of the value function, which we locally estimated by the difference between an upper and a lower bounds constructed on each simplex.

We provide a detailed review of simplicial partitioning of hyperrectangles in "Appendix 1".

3.1 Simplicial piecewise linear approximation of the value function

In any period t, assume at some iteration of the simplicial algorithm, the state space S_t has been partitioned into simplices, and the expected value function has been evaluated at the extreme points $\boldsymbol{s}_t^k \in S_t$, $k = 1, \ldots, K$, $f^k := \hat{\mathcal{V}}_t(\boldsymbol{s}_t^k, \tilde{\boldsymbol{q}}_t)$. (In the sequel, we drop the time index t for ease of notation.) Then, for any point $\boldsymbol{s} \in S$, the expected value function can be approximated by the following linear program, which by the concavity of the approximate value function yields a lower bound, $B_L(\boldsymbol{s})$:

$$B_L(\boldsymbol{s}) := \max \sum_{k=1,\ldots,K} \lambda_k f^k \text{ s.t. } \boldsymbol{s} = \sum_{k=1,\ldots,K} \lambda_k \boldsymbol{s}^k, \ \sum_{k=1,\ldots,K} \lambda_k = 1, \text{ and } \lambda_k \geq 0 \ \forall k. \tag{17}$$

Let $\mathcal{B}(\boldsymbol{s})$ be the set of indices of the nonzero components λ_k in a basic optimal solution of the linear program (17); $\mathcal{B}(\boldsymbol{s})$ contains at most $n+1$ elements so that the point $\boldsymbol{s}$ can be expressed as a convex combination of at most $n+1$ vertices, and the set of all convex combinations of these vertices is a simplex. Also, if vectors of subgradients $\boldsymbol{g}^k, k \in \mathcal{B}(\boldsymbol{s})$, are known at the grid points $\boldsymbol{s}^k$, then the expected value function is bounded above by:

$$B_U(\boldsymbol{s}) := \min_{k \in \mathcal{B}(\boldsymbol{s})} f^k + {\boldsymbol{g}^k}^\top (\boldsymbol{s} - \boldsymbol{s}^k). \tag{18}$$

Then $B_L(\boldsymbol{s}) \leq f(\boldsymbol{s}) \leq B_U(\boldsymbol{s})$ so that $B_U(\boldsymbol{s}) - B_L(\boldsymbol{s})$ is an upper bound on the approximation error at the point $\boldsymbol{s}$ using the support vertices $\boldsymbol{s}^1, \ldots, \boldsymbol{s}^K$. It is also pointed out in Zéphyr et al. (2015) that the largest error bound on the simplex with vertex set $\mathcal{B}$ is given by the linear program:

$$\overline{E}_{\mathcal{B}} := \max_{s,\phi,\lambda_k,k\in\mathcal{B}} \phi - \sum_{k\in\mathcal{B}} \lambda_k f^k$$
$$\text{s.t.} \quad \boldsymbol{s} = \sum_{k\in\mathcal{B}} \lambda_k \boldsymbol{s}^k, \ \sum_{k\in\mathcal{B}} \lambda_k = 1, \ \lambda_k \geq 0 \ \text{ and } \phi \leq f^k + \boldsymbol{g}^{k\top}(\boldsymbol{s} - \boldsymbol{s}^k), \ \forall k \in \mathcal{B}. \tag{19}$$

If the error bound $\overline{E}_{\mathcal{B}}$ exceeds a certain criterion, then an optimal point $s^*_{\mathcal{B}}$ of (19) would be a candidate vertex for being added to the set of vertices as $\boldsymbol{s}^{K+1} := \boldsymbol{s}^*_{\mathcal{B}}$. Similarly, if there exists some analytical expression for the function $f(\boldsymbol{s}) := \hat{\mathcal{V}}_t(\boldsymbol{s}_t, \boldsymbol{q}_t)$, the largest actual approximation error on a simplex with vertices in $\mathcal{B}$ can be found through the nonlinear program:

$$E_{\mathcal{B}} := \max_{s,\lambda_k,k\in\mathcal{B}} f(\boldsymbol{s}) - \sum_{k\in\mathcal{B}} \lambda_k f^k \ \text{s.t.}\ \boldsymbol{s} = \sum_{k\in\mathcal{B}} \lambda_k \boldsymbol{s}^k, \ \sum_{k\in\mathcal{B}} \lambda_k = 1 \ \text{and} \ \lambda \geq \boldsymbol{0}. \tag{20}$$

In the approach of Zéphyr et al. (2015), an initial set of vertices is first chosen, for example the 2^n vertices of the hyperrectangle S plus one interior point $\boldsymbol{s}^{(2^n+1)}$. Next an initial set of simplices is explicitly enumerated that spans these vertices. Then the linear program (19) is solved for every simplex in the set and the next vertex to be added is selected as the optimal solution $\boldsymbol{s}^*_{\mathcal{B}}$ for the simplex $\mathcal{B}$ with the largest error bound $\overline{E}_{\mathcal{B}}$. Such a point $\boldsymbol{s}^*_{\mathcal{B}}$ is called a division point and the list of simplices is correspondingly updated by deleting the simplex with vertex set $\mathcal{B}$ from the list and adding to the list the new simplices created by dividing $\mathcal{B}$. Iterating this way until a termination criterion is satisfied, the method of Zéphyr et al. (2015) stops with a list of, say, K vertices $\boldsymbol{s}^1, \dots, \boldsymbol{s}^K$ at which the approximate value function and its expectation are evaluated, together with a potentially very large list of associated simplices.

The advantage of this scheme is that it provides a monotonic error bound sequence on the approximation error. However, its Achille's heel is the exhaustive examination of the list of created simplices that is kept in memory in each time period, and the slow convergence. Depending on the size of such a list, this might be very expensive in terms of memory usage; this is the focus of the next subsection.

3.2 Complexity and convergence analysis

A detailed complexity analysis of general operations on simplices (not the simplicial approximation itself) is provided in Zéphyr et al. (2017). In particular, at each iteration k of the procedure, assume we have a list of r^k *active* simplices, finding the simplex with the worst approximation error requires $\mathcal{O}(r^k)$ operations.

Now, assume we want to partition the hypercube S into simplices until a desired error bound, $\overline{E}_0$, is attained. Therefore, our goal is to find a full-dimensional simplex $\mathcal{B} \subset S$ generated by the columns of a full row rank matrix $\boldsymbol{S}_{\mathcal{B}} \in \boldsymbol{R}^{n\times(n+1)}$, such that the optimal value of (19) is $\overline{E}_{\mathcal{B}} \leq \overline{E}_0$. Toward this end, we first decompose the hypercube S into initial simplices, and for each created simplex solve (19) to find the largest error bound as well as the divisison point $\boldsymbol{s}$. Then, the initial simplex with the largest error is divided at the corresponding division point using the radial ω-subdivision strategy (see “Appendix 1”). We repeat the same process until the threshold $\overline{E}_0$ is met.

Proposition 2 *Let Vol(S) be the volume of the hyperrectangle S, the number of simplices required to achieve the error bound $\overline{E}_0$ is of the order* $\mathcal{O}\left(\frac{\mathrm{Vol}(S)n!}{(n+1)\overline{E}_0^{n/2}}\right)$.

A proof of this proposition is provided in “Appendix 2”.

Furthermore,

Proposition 3 *Assume at each iteration of the simplicial scheme, the ω-subdivision of simplex is used, the simplicial algorithm will converge to the desired error bound $\overline{E}_0$ in a finite number of steps, which is proportional to an exponential factor.*

Proof Under the ω-subdivision strategy, at each iteration k of the simplicial partitioning scheme, the number of created simplices (subdivision of the simplex with

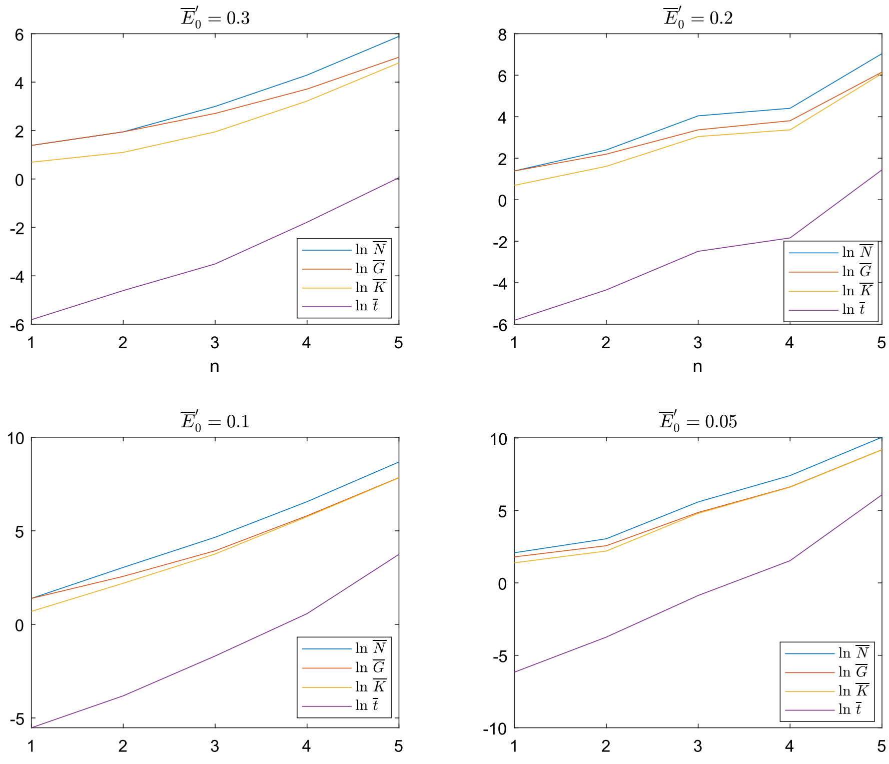


Fig. 1 Graphical illustration of the simplicial approximation complexity for quadratic functions

the highest error bound), N^k, is $2 \leq N^k \leq n+1$. In addition, assume K iterations (simplex subdivisions) are performed, and N simplices created, then we have $2K \leq KN^k \leq K(n+1)$, i.e., $K \geq \frac{N}{n+1} \geq \frac{2K}{n+1}$. It follows from (32) that K is of the order $\mathcal{O}\left(\frac{\mathrm{Vol}(S)n!}{\overline{E}_0^{n/2}}\right)$, which concludes the proof. □

Let us numerically illustrate Proposition (3). First, let us consider hypothetical quadratic expected value functions, of the form $\mathcal{V}(\boldsymbol{s}) = -\frac{1}{2}\boldsymbol{s}^\top \boldsymbol{A}\boldsymbol{s} + \boldsymbol{b}^\top \boldsymbol{s}$, where the matrices $\boldsymbol{A}$ and vectors $\boldsymbol{b}$ are randomly generated.

Let us consider relative error bounds $\overline{E}_0'$, as the ratio of a simplex error bound to the maximal error over the initial simplices. For each considered state dimension and relative error threshold indicated in the results reported in Fig. 1, five replications of the simplicial decomposition algorithm are performed.

Figure 1 depicts the natural logarithm of the average total number of created simplices ($\overline{N}$), grid points ($\overline{G}$), iterations ($\overline{K}$), which also is the additional simplices

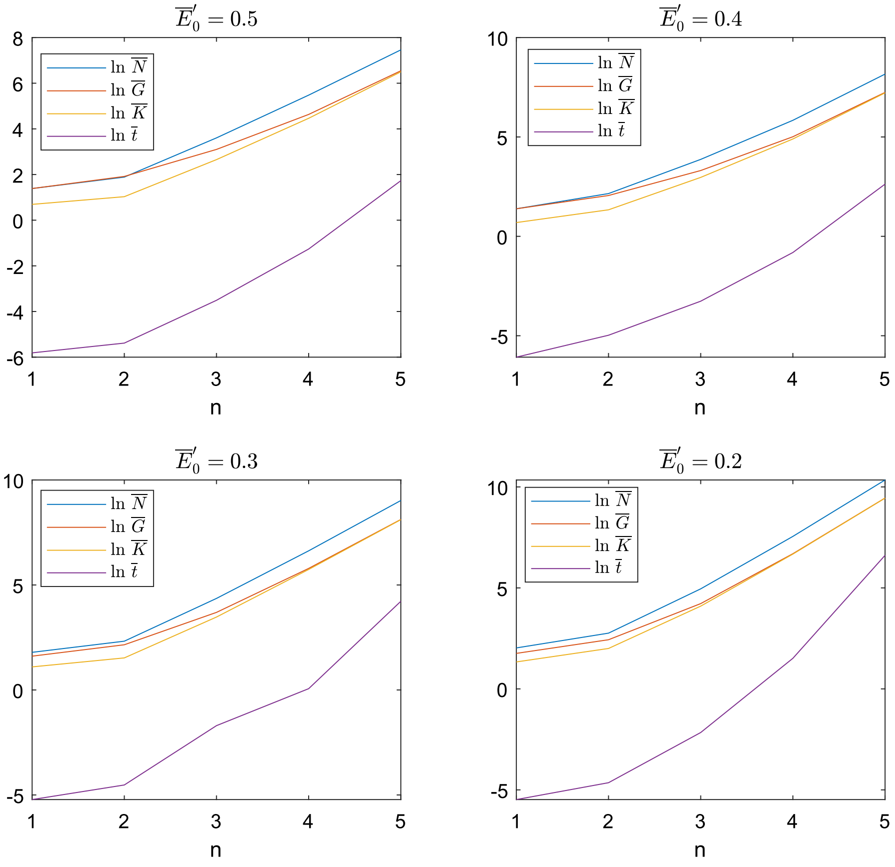


Fig. 2 Graphical illustration of the simplicial approximation complexity for concave Cobb–Douglas functions

created (in addition to the initial ones), and the CPU time ($\bar{t}$), for different error thresholds and state space dimensions. These results confirm that the computational burden to achieve a fixed error bound increases more than exponentially with the dimension, n, of the hyperrectangles.

Let us repeat the same tests on hypothetical Cobb–Douglas expected value functions of the form

$$\mathcal{V}(\boldsymbol{s}) = \prod_{i=1}^{n} s_i^{\alpha_i} \quad (\alpha_i \geq 0 \text{ and } \sum_{i=1}^{n} \alpha_i \leq 1). \tag{21}$$

As for the quadratic functions, for each error threshold and each state space dimension, the simplicial procedure is carried out to construct grid points to approximate the functions, and five replications are performed. The same statistics are calculated as above. Samples of results reported in Fig. 2 also confirm that the complexity of the simplicial scheme is exponential in the state space dimension.

In closing,

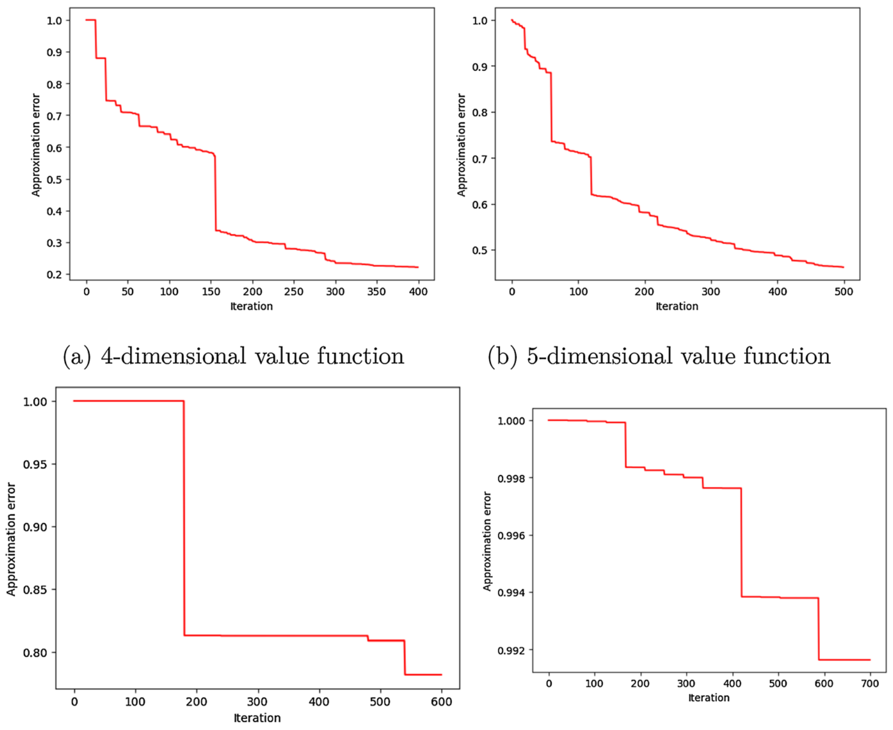
Proposition 4 *The convergence rate of the simplicial algorithm is at best linear.*

Proof Since at each iteration the simplex with maximal error bound $\overline{E}_{\mathcal{B}}$ is divided, the simplicial algorithm generates a non-increasing sequence $\{\overline{E}_{\mathcal{B}k}\}$, such that, by Proposition (3), $\lim\limits_{k\to\infty} \overline{E}_{\mathcal{B}k} = 0$. Indeed, at any iteration of the algorithm, assume simplex $\mathcal{B} \subset S$, generated by the matrix $\boldsymbol{S}_{\mathcal{B}}$ (a matrix whose columns are the extreme points of the simplex), is divided; consider any resulting subsimplex $\mathcal{B}^c$ with generating matrix $\boldsymbol{S}_{\mathcal{B}}{}^c$. Matrices $\boldsymbol{S}_{\mathcal{B}}$ and $\boldsymbol{S}_{\mathcal{B}}{}^c$ differ only by one column. The only column of $\boldsymbol{S}_{\mathcal{B}}{}^c$ that is not in $\boldsymbol{S}_{\mathcal{B}}$ is the division point, $\boldsymbol{s}^*_{\mathcal{B}}$, of the parent simplex $\mathcal{B}$, and is a convex combination of the columns of $\boldsymbol{S}_{\mathcal{B}}$.

Now, given that the approximate value function (10)–(16) and its expectation are concave, we have $\sum_{k\in\mathcal{B}} \lambda^*_k \hat{\mathcal{V}}(\boldsymbol{s}^k, \cdot) \leq \hat{\mathcal{V}}(\boldsymbol{s}^*_{\mathcal{B}}, \cdot)$, where λ^* is the optimal λ from Problem (19), and the $\boldsymbol{s}^k$'s are the vertices of the parent simplex $\mathcal{B}$, or the columns of matrix $\boldsymbol{S}_{\mathcal{B}}$. Thus, we always have $\sum_{k\in\mathcal{B}} \lambda^*_k \hat{\mathcal{V}}(\boldsymbol{s}^k, \cdot) \leq \sum_{j\in\mathcal{B}^c} \lambda_j \hat{\mathcal{V}}(\boldsymbol{s}^j, \cdot)\ 0 \leq \lambda_j \leq 1$, where the $\boldsymbol{s}^j$'s (one of them being the optimal division point $\boldsymbol{s}^*_{\mathcal{B}}$) are the extreme points of the subsimplex $\mathcal{B}^c$. Similarly, due to the concavity of the function, $\hat{\mathcal{V}}(\boldsymbol{s}^*_{\mathcal{B}}, \cdot) \leq \min_{k\in\mathcal{B}}\{f^k + \boldsymbol{g^k}^\top(\boldsymbol{s}^*_{\mathcal{B}} - \boldsymbol{s}^j)\}$ (the extrapolation of the function at $\boldsymbol{s}_{\mathcal{B}}$). It is also clear that $\min_{j\in\mathcal{B}^c}\{f^j + \boldsymbol{g^k}^\top(\boldsymbol{s}^c - \boldsymbol{s}^j),\ s^c \in \mathcal{B}^c\} \leq \min_{k\in\mathcal{B}}\{f^k + \boldsymbol{g^k}^\top(\boldsymbol{s} - \boldsymbol{s}^k)\},\ \boldsymbol{s} \in \mathcal{B}$.

Therefore, due to the concavity of the approximate value function, we always have $\overline{E}_{\mathcal{B}^c} \leq \overline{E}_{\mathcal{B}}$, where $\overline{E}_{\mathcal{B}^c}$ and $\overline{E}_{\mathcal{B}}$ are the maximal error bound on the function over subsimplex $\mathcal{B}^c$ and parent simplex $\mathcal{B}$, respectively. As a result, the error sequence $\{\overline{E}_{\mathcal{B}k}\}$ is non-increasing, and $\lim\limits_{k\to\infty} \frac{\overline{E}_{\mathcal{B}k+1}}{\overline{E}_{\mathcal{B}k}} \leq 1$; and the proof is complete. □

Figure 3 illustrates the convergence of the simplicial algorithm on the approximation of value functions for four midterm reservoir problems. We consider a ten-period planning horizon, and the parameters of the problems are generated as described in the numerical experiment section. For each case, we generate five



(a) 4-dimensional value function (b) 5-dimensional value function

(c) 6-dimensional value function (d) 7-dimensional value function

Fig. 3 Illustration of the convergence of the simplex algorithm

replications. The grid sizes are fixed at $100n + 2^n$. The evolution of the average relative error (ratio of the error at each iteration to that of the first iteration) for the first period is depicted in Fig. 3.

As stated in the proof of Proposition (4), we see that the sequence of the approximation error is non-increasing. For the four-dimensional problems, at the last iteration, the initial error is reduced to approximately 20%, and around 75% for the six-dimensional problems, suggesting that denser grid sizes are needed to obtain a similar precision as for the four-dimensional problems.

In general, the approximation error decreases relatively fast over the first few iterations, then slows down dramatically. This is due to the fact that, as the active simplices (not yet divided) become smaller, the local curvature of the function does not vary significantly, as a result, the approximation error is relatively steady on the existing simplices.

An apparent disadvantage of the simplicial scheme, especially for state space dimensions greater than or equal to ten, is the extra computational burden associated with a potentially very large list of simplices as well as the complete, uniform exploration of the whole state space which may not be required in practical applications where more localized approximations would be adequate.

Therefore in this paper we seek to explore other ways of constructing grid points to evaluate the approximate value function and its expectation in each period without enumerating an exhaustive list of associated simplices in the hope to alleviate the inherent exponential complexity of the simplicial approach.

4 Hybrid simplicial approximate dynamic programming

We now examine some randomized approaches for selecting new grid points at which to evaluate the approximate value function (10)–(16) in each period t that avoid making a large list of active simplices. With these approaches, it is not possible to identify a division point of largest error bound, so there is a need for statistical estimation of the approximation error, and other heuristics must be called upon for selecting a new grid point at each iteration. We first describe three such heuristics and next we discuss statistical estimation of the approximation error.

4.1 Randomized simplex-based sampling of the reservoir level space

Monte Carlo (MC). Instead of using a regular grid of equally spaced vertices, one simple and very crude approach is to use a sequence of pseudo-random vertices. In each period t, let $\boldsymbol{v}^k$ be a sequence of n-vectors of independent variates, uniformly distributed in $[\mathbf{0}, \mathbf{1}]$. Again, we drop the time period index t for ease of notation. Starting with the initial set of 2^n extreme points of the hyperrectangle S, the i-th component of the kth random vertex is given by $s_i^{(2^n+k)} = \underline{s}_i + (\overline{s}_i - \underline{s}_i)v_i^k$, for $i = 1, \ldots, n$.

This naïve random sequence of approximation nodes can be considered neutral with respect to the approximation error in the sense that the choice of the next vertex to enter the support set is not based on an error criterion such as the division point of a simplex with largest error bound in Eq. (19). Therefore one would expect that a numerical comparison of this naïve scheme with the previous method would show a significant difference in accuracy.

MC simplicial. This method combines the idea of using a simplicial approximation with that of using the Monte Carlo scheme, but the way it searches for a simplex to be divided is different than what was done in our previous simplicial method, because here the state space is not exhaustively partitioned into an simplices, as discussed in “Appendix 3”. In period t, suppose the approximate value function has been evaluated at K points. We then generate a random point $\hat{\boldsymbol{s}}$ uniformly in S as before ($\hat{s}_i = \underline{s}_i + (\overline{s}_i - \underline{s}_i)v_i$), solve Eq. (17) to find the vertex set $\mathcal{B}(\hat{\boldsymbol{s}})$ of the simplex containing $\hat{\boldsymbol{s}}$ and solve Eq. (19) to obtain the division point $\boldsymbol{s}^*_{\mathcal{B}}$ that has the largest error bound in that simplex. Lastly, we choose that division point as the new vertex $\boldsymbol{s}^{K+1} = \boldsymbol{s}^*_{\mathcal{B}}$. This procedure is repeated until the size of the grid reaches a desired target.

Batch MC simplicial. As in the MC simplicial method, in period t, suppose at a given iteration there are K vertices in the grid, with $K \geq n+1$. Next, we generate a sample of m random points $\hat{\boldsymbol{s}}^1, \ldots, \hat{\boldsymbol{s}}^m$ uniformly in S. For each random point $\hat{\boldsymbol{s}}^j$ in the sample, Eq. (17) is solved to find the vertex set $\mathcal{B}^j$ of the simplex that contains $\hat{\boldsymbol{s}}^j$, and Eq. (19) to obtain the division point $\boldsymbol{s}^*_{\mathcal{B}^j}$ that has the largest error bound $\overline{E}_{\mathcal{B}^j}$ in that simplex. Then the new vertex is chosen as the division point of the simplex with the largest error bound in the sample, so $\boldsymbol{s}^{K+1} = \boldsymbol{s}^*_{\mathcal{B}^{j^*}}$ where $j^* = \arg\max_{j=1,\ldots,m} \overline{E}_{\mathcal{B}^j}$. This way, by evaluating a small number m of simplices, we have good chances of choosing a candidate with a relatively large error bound, but without having to maintain a large list of simplices as in the previous papers.

By keeping one candidate out of m at each iteration, the best we can hope for is that the selected vertices would belong to the top (1/m)th among the sampled candidates. But there is a probability $(1-1/m)^m$ that the selected vertex is not in the top (1/m)th, and also some probability that the sample has more than one candidate in the top (1/m)th, so that good candidates are discarded in some iterations. With $m=3$, these probabilities are 8/27 (select bad vertex because all 3 candidates are not in the top one third, each with probability 2/3) and 7/27 (discard a good vertex, since at least 2 candidates are in the top one third). While this seems better than the MC and the MC simplicial methods, where all vertices are selected (good and bad),

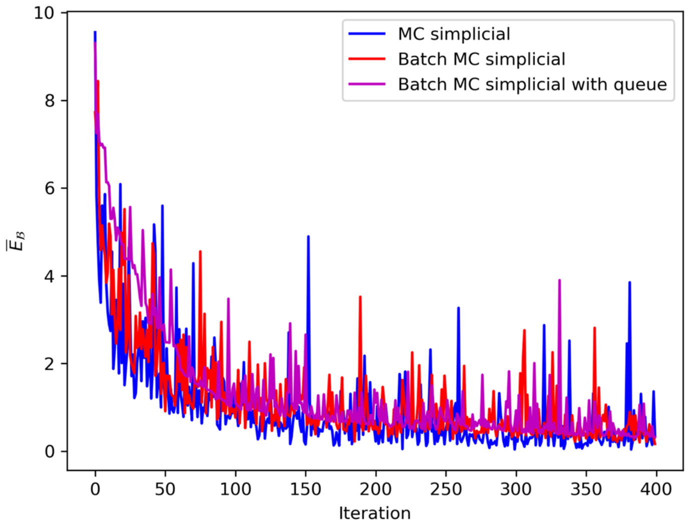


Fig. 4 Illustration of the convergence of the hybrid simplicial methods on the approximation of the first period value function for one of the 4-reservoir literature test problems described in Sect. 5.3

we can try to improve the selection process by putting some candidate vertices in a waiting line instead of discarding them right away.

Batch MC simplicial with queue. As in the batch MC simplicial method, but now, we keep a list, of at most r recently explored simplices, which have been queued from previous iterations instead of being discarded. Initially, the queue is empty. In a typical iteration, m new candidates are sampled as in the batch MC simplicial method, which are combined into a pool with the (at most) r candidates from the queue. The new vertex is chosen as the division point of the simplex with the largest error bound in the pooled candidate list. The next r candidates with largest error bounds are held in the queue, and the remaining candidates with the smallest error bounds are discarded.

Parameters for this would need to be experimented if this turns out to be a tempting avenue. The computational effort is similar to the batch MC simplicial method but it is hoped that the batch MC simplicial with queue would have smaller approximation error than the batch MC simplicial.

The above methods attempt to replace an exhaustive list of simplices with a shorter list from which a division point is chosen with the largest error bound at each step. It is hoped that the use of a truncated candidate list will be compensated by the large number of sampled points and simplices over a large number of steps. However, in absence of an exhaustive list of simplices, there is no uniform upper bound on the approximation error. Also, as it is illustrated in Fig. 4, in contrast to the simplicial scheme, there is no guarantee as to the monotonicity of the sequence of generated approximation errors. Thus, the next section will discuss the statistical estimation of error.

An illustrative comparison between the original and the hybrid simplicial methods is provided in “Appendix 3”.

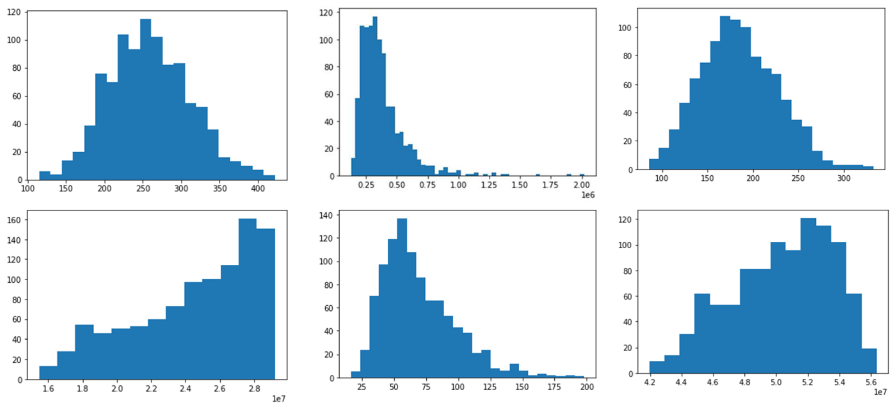


Fig. 5 Examples of empirical distributions of the approximation errors

4.2 Statistical estimation of the approximation error

Under the concavity of the expected value function $\hat{\mathcal{V}}_t(\boldsymbol{s}_t, \cdot)$, the approximation error is the difference between the function and its piecewise linear approximation. At any point $\boldsymbol{s}_t \in S_t$, the approximation error is $\hat{\mathcal{V}}_t(\boldsymbol{s}_t, \cdot) - B_L(\boldsymbol{s}_t)$, where $B_L(\boldsymbol{s}_t)$ solves Eq. (17). Then Eq. (18) implies the approximation error is bounded by $B_U(\boldsymbol{s}_t) - B_L(\boldsymbol{s}_t)$. At all points in simplex $\mathcal{B}$ that contains $\boldsymbol{s}_t$, the approximation error is bounded by $\overline{E}_{\mathcal{B}}$ of Eq. (19), while the largest error on the simplex is $E_{\mathcal{B}}$ of Eq. (20). Here we are interested in the estimation of the largest actual error $\max_{\boldsymbol{s}_t \in S_t}\{\hat{\mathcal{V}}_t(\boldsymbol{s}_t, \cdot) - B_L(\boldsymbol{s}_t)\}$ or the largest error bound $\max_{\mathcal{B}} \overline{E}_{\mathcal{B}}$. In both cases, we will use a random sample of m points $\hat{\boldsymbol{s}}^1, \dots, \hat{\boldsymbol{s}}^m \in S$.

Since function $\hat{\mathcal{V}}_t(\boldsymbol{s}_t, \cdot)$ is finite and concave everywhere on S_t, by construction, the approximation error is also a well-behaved function; it is equal to zero at the support nodes and varies smoothly on the simplices. Therefore, when sampling the state space uniformly, it might be reasonable to assume that the corresponding distribution of the approximation error is also well-behaved. However, since we do not know the theoretical distribution, first, we conduct an empirical investigation. To this end, we generate samples of grid points with the different randomized methods - except for the pure Monte Carlo and the simplicial methods, and calculate the true approximation errors for random sample points. Examples of empirical distributions are illustrated in Fig. 5. The true empirical distribution seems to be less scattered than a uniform distribution, and perhaps to lie somewhere between a left triangular distribution with mode at the minimum value, or a right triangular distribution with mode at the maximum value.

Table 1 Formulas for sampling distributions and quantiles

	TR(0,b)	U(0,b)	TL(0,b)
$p = F_m(y)$	y^{2m}	y^m	$\left[1 - (1 - y)^2\right]^m$
$y = F_m^{-1}(p)$	$p^{1/2m}$	$p^{1/m}$	$1 - \sqrt{1 - p^{1/m}}$

Table 2 Formulas for unbiased point estimate $\hat{b}$ of b and limits of confidence interval

	TR $(0, b)$	U $(0, b)$	TL $(0, b)$
$\hat{b}_{1-\alpha/2}$	$\frac{1}{(1-\alpha/2)^{1/2m}}$	$\frac{1}{(1-\alpha/2)^{1/m}}$	$\frac{1}{1-\sqrt{1-(1-\alpha/2)^{1/m}}}$
$\hat{b}$	$\frac{2m+1}{2m}$	$\frac{m+1}{m}$	$\frac{1}{1-A_m}$
$\hat{b}_{\alpha/2}$	$\frac{1}{(\alpha/2)^{1/2m}}$	$\frac{1}{(\alpha/2)^{1/m}}$	$\frac{1}{1-\sqrt{1-(\alpha/2)^{1/m}}}$

Table 3 Numerical example for point estimate $\hat{b}$ of b and confidence interval with $m = 3$ and 30, $\alpha = 0.05$ and $x_{(m)} = 10$

Symbol	TR $(0, b)$		U $(0, b)$		TL $(0, b)$	
m	3	30	3	30	3	30
$\hat{b}_{1-\alpha/2}$	10.04	10.00	10.08	10.01	11.01	10.30
$\hat{b}$	11.67	10.17	13.33	10.33	18.42	11.90
$\hat{b}_{\alpha/2}$	18.49	10.63	34.20	11.31	62.97	15.16

Therefore, we propose to use, as statistical models, four simple distributions on (0, b): the right-angled triangular with mode at right $\mathrm{TR}\,(0, b)$, the uniform $\mathrm{U}\,(0, b)$, the right-angled triangular with mode at left $\mathrm{TL}\,(0, b)$, and the symmetrical triangular with mode at the center $\mathrm{TC}\,(0, b)$. For these distributions, the parameter b can be estimated with order statistics.

If a random variable X has a uniform distribution on the interval [0, b], then it is well known, see e.g., Gibbons (1974), that the maximum likelihood estimator (MLE) of the parameter b is the largest observation in the sample. So with sample size m and observed values $x_1, \ldots, x_m$, the MLE of b is $x_{(m)} = \max_{i=1,\ldots,m} x_i$. Estimators of the limit parameters of a right-angled triangular distribution on the interval [a, b], with the mode at the upper limit b, are given in Kachiashvili and Topchishvili (2016), where it is shown that the MLE of b is also $x_{(m)}$. However, by arguing as in Lamond and Zéphyr (2021), it is easily seen that $x_{(m)}$ is not an MLE of b for a right-angled triangular distribution with the mode at the lower limit. The true MLE is provided in "Appendix 4". By the symmetry of the triangular distribution with mode at the center, a simple unbiased estimator of the upper limit is $\hat{b} = x_{(1)} + x_{(m)}$ where $x_{(1)} = \min_{i=1,\ldots,m} x_i$.

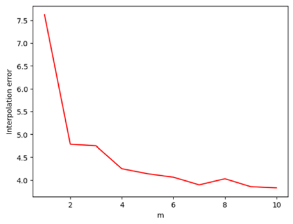
In addition to the point estimates of parameter b, it is useful to obtain confidence intervals. For this, it is convenient to define the standardized random variable Y with distribution on the unit interval [0, 1]. For a random sample of m observations, we define the largest of them by $y_{(m)}$, with the random variable $Y_{(m)}$ representing its sampling distribution. Let $F_m(y)$ be the cumulative distribution function of $Y_{(m)}$. Then $p = F_m(y)$ is the cumulative probability and $y = F_m^{-1}(p)$, the quantile. Formulas for these are given in Table 1 for our first three distributions.

Formulas for unbiased point estimates, $\hat{b}$, of parameter b with lower and upper limits of confidence intervals are given in Table 2 as multipliers of $x_{(m)}$, where

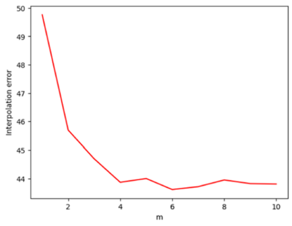
$$A_m = \prod_{j=1}^{m} \frac{j}{j + 0.5}, \tag{22}$$

from adapting equation (6) of Kachiashvili and Topchishvili (2016).

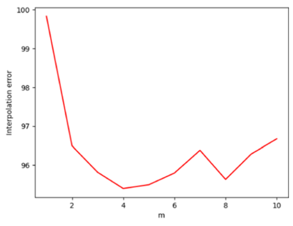
A numerical example is given in Table 3. For the triangular distribution with mode at left $\mathrm{TL}\,(0, b)$, we see that the unbiased estimate and confidence interval limits based on the order statistic $x_{(m)}$ are quite large compared to the other two distributions. There might be an interest here in using an MLE estimate instead, which has small bias and smaller variance as pointed out in Lamond and Zéphyr (2021) thus allowing a smaller sample size for estimating the approximation error, and therefore fewer computations. In the absence of a simple formula for the cumulative distribution of $x_{(1)} + x_{(m)}$ for the triangular distribution with mode at center $\mathrm{TC}\,(0, b)$, we can use Monte Carlo simulations to obtain approximate confidence intervals for b. For instance, for $x_{(1)} + x_{(m)} = 10$ the point estimate is $\hat{b} = 10$ and the 95% confidence intervals are respectively $6.1 \leq b \leq 27$ for $m = 3$, and $8.1 \leq b \leq 13$ for $m = 30$.



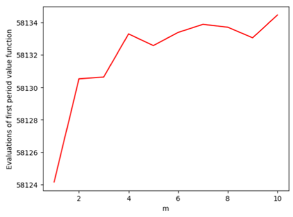
(a) Interpolation errors of 3-dimensional Cobb-Douglas functions.



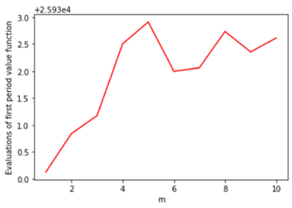
(b) Interpolation errors of 6-dimensional Cobb-Douglas functions.



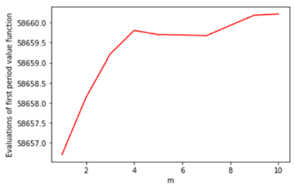
(c) Interpolation errors of 9-dimensional Cobb-Douglas functions.



(d) First period 3-dimensional value function values.



(e) First period 4-dimensional value function values.



(f) First period 6-dimensional value function values.

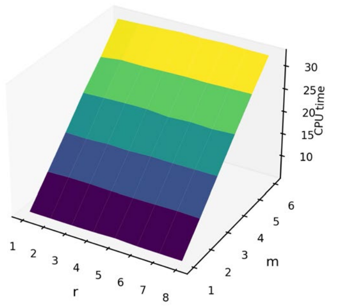
Fig. 6 Performance of the batch MC simplicial method on different types of problems and by sample size

5 Numerical experiments

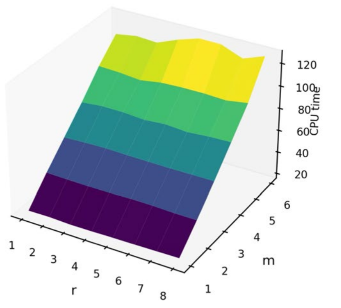
Three types of analysis are carried out in the numerical experiments. First, in Sect. 5.1, we appraise the sensitivity of the performance of the two Monte Carlo simplicial methods with respect to their underlying parameters. Second, in Sect. 5.2 the methods are compared on the trade-off between accuracy and computational burden on (i) the approximation of concave functions; and (ii) several simulated reservoir optimizations problems. Lastly, in Sect. 5.3, we compare the methods on three reservoir optimization problems available in the literature.

5.1 Sensitivity of solution performance to parameter values: batch MC simplicial and bath MC simplicial with queue methods

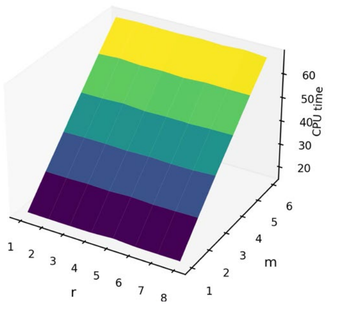
Recall that in the batch MC simplicial method, in each period t, at each iteration, a sample of m random points is chosen in the state space, S_t. Intuitively, this approach is approximately m times slower than the MC simplicial scheme, in which one random point is selected at each iteration. One natural question is how to determine the appropriate sample size m. Though we do not have any theoretical answer to this question, we perform numerical experiments to analyze the sensitivity of solution performance on the approximations of Cobb–Douglas type functions (in dimension n=3, 6, and 9,



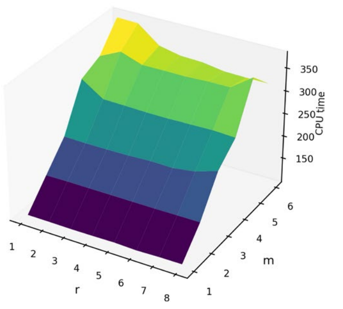
(a) Interpolation of 6-dimensional Cobb-Douglas functions.



(b) Interpolation of 10-dimensional Cobb-Douglas functions.



(c) Approximation of 3-dimensional first period value functions.

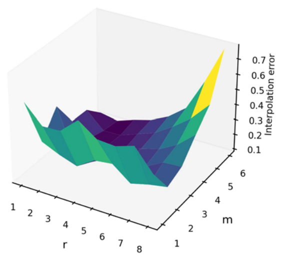


(d) Approximation of 4-dimensional first period value functions.

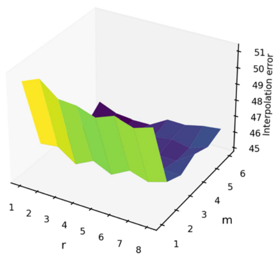
Fig. 7 Average CPU time in seconds of the batch MC simplicial with queue method for different types of problems and by sample size

respectively), with randomly generated parameters, and the approximation of value functions for reservoir management problems (in dimension n=3, 4, and 6, respectively).

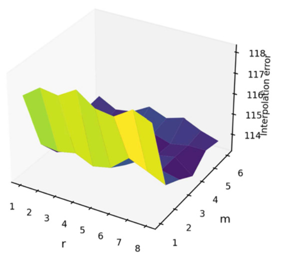
We approximate the Cobb–Douglas functions on grids of size 100n, then interpolate the values of the functions on other grids (out-of-sample) of size 200n (solving Problem (17)) and calculate the true approximation errors. For the reservoir management problems, we approximate the value functions (in each time period) on grids of size 100n as well, then solve the first period problem for a sample of 200n $(\boldsymbol{s}_1, \boldsymbol{q}_1)$ state pairs. For each case (Cobb–Douglas function approximations and value function approximations), five replications are performed for values of m ranging from one to ten. The average results are reported in Fig. 6. Note that smaller values are better in the upper portion of the figure, and the opposite in the lower portion of the figure. The figure displays an "imperfect elbow shape", and seems that values of m between three to five would suffice to obtain good approximation performance. The computational burden grows linearly with the parameter m; since we strive for



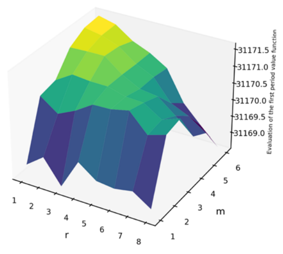
(a) Interpolation errors of 2-dimensional Cobb-Douglas functions.



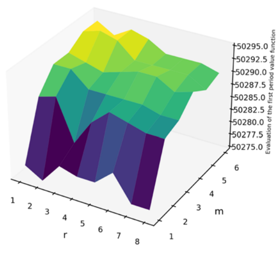
(b) Interpolation errors of 6-dimensional Cobb-Douglas functions.



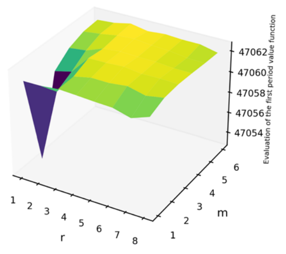
(c) Interpolation errors of 10-dimensional Cobb-Douglas functions.



(d) Approximation of 2-dimensional first period value functions.



(e) Approximation of 3-dimensional first period value functions.



(f) Approximation of 4-dimensional first period value functions.

Fig. 8 Performance of the batch MC simplicial with queue method on different types of problems and by sample size

a good trade-off between computational burden and accuracy, in the sequel, we will fix m at 3.

Similarly, the batch MC simplicial with queue method features two parameters m (same as the previous method), and r, the size of the queue of previously generated random points. We perform the same experiments as above to assess the sensitivity of solution performance with respect to these parameters. We vary the values of m between one and six (based on the above observations), and the values of r between one and eight. Overall, the computational burden is linear in m, and does not seem to be influenced by the lenght of the queue, r (Fig. 7); similarly for the performance of the solution (Fig. 8). In addition, in Fig. 8, in most of the cases, for fixed value r, we observe an elbow shape at $m = 3$ (except for the last picture), suggesting that $m = 3$ seems to be a good enough sample size. Extensive numerical experiments have demonstrated that this method exhibits similar performance (both in terms of computational burden and accuracy) than the batch MC simplicial scheme; thus, results for this method will not be reported in the sequel for the sake of brevity.

Table 4 Statistics pertaining to interpolation errors of Cobb–Douglas concave functions

			Relative averages				
n	Method	$\overline{\text{Abs. CPU time}}$	$\overline{ti}$	$\overline{E}_{\text{min}}$	$\overline{E}_{\text{max}}$	$\overline{E}_{\text{av}}$	$\overline{E}_{\text{std}}$
3	Monte Carlo (MC)	0.0053	0.0052	0.5900	16.9683	3.0710	11.0501
3	MC simplicial	2.6148	2.5680	0.5494	6.7624	0.9346	3.3748
3	Batch MC simplicial	7.1774	7.0491	0.7292	1.5631	0.5604	0.9166
3	Simplicial	1.0182	1	1	1	1	1
5	Monte Carlo (MC)	0.0195	0.0032	0.2967	1.9246	0.7857	1.9677
5	MC simplicial	6.8779	1.1196	0.3354	0.9538	0.3653	0.7031
5	Batch MC simplicial	21.8420	3.5556	0.4331	1.0314	0.3080	0.5134
5	Simplicial	6.1430	1	1	1	1	1
8	Monte Carlo (MC)	0.0353	0.0001	0.2867	0.9061	0.5261	1.1074
8	MC simplicial	10.5880	0.0347	0.3358	0.5883	0.3364	0.5819
8	Batch MC simplicial	32.6510	0.1070	0.3689	0.5450	0.3217	0.4596
8	Simplicial	305.2224	1	1	1	1	1
10	Monte Carlo (MC)	0.0891	1.3808E−06	0.3499	0.9094	0.6171	1.2257
10	MC simplicial	22.1880	3.4376E−04	0.4115	0.7349	0.4438	0.7686
10	Batch MC simplicial	66.3756	1.0284E−03	0.5561	0.6184	0.4227	0.5881
10	Simplicial	64,544.1457	1	1	1	1	1

5.2 Accuracy versus computational burden

Here, we focus on the trade-off between accuracy and computation time. First, in Sect. 5.2.1, we compare the performance of the methods on the approximation of Cobb–Douglas concave functions of the form (21) for different state dimensions n. Though the primary interest of this work is mid-term reservoir management problems, this first setting is motivated by the fact that (i) the simplex-based approximations exploit the concavity property of the functions to be approximated, in contrast to the pure Monte Carlo (MC) scheme; (ii) in the reservoir management context, to handle the nonlinearity of the production functions, we approximate the latter by piecewise concave linear functions (Problem (10)–(16)); (iii) similarly, the value functions are approximated by piecewise concave linear functions (Problem (10)–(16)). Thus, it is no easy task isolating the sole effects of the methods, due to the multiple layers of approximation embedded in the dynamic programs.

Next, in Sect. 5.2.2, the schemes are gauged on several simulated reservoir management problems.

5.2.1 Approximation of concave functions

Grid points of size $2^n + 100n$ are generated with each method; then the out-of-sample interpolation errors - the difference between the true and interpolated values- are calculated on randomly generated samples of sizes $200n$. In addition, under each

Table 5 Statistics pertaining to interpolation errors of 11- to 15-dimensional Cobb–Douglas concave functions using the hybrid methods

n	Method	$\overline{ti}$	$\overline{\mathrm{E}}_{\mathrm{min}}$	$\overline{\mathrm{E}}_{\mathrm{max}}$	$\overline{\mathrm{E}}_{\mathrm{av}}$	$\overline{\mathrm{E}}_{\mathrm{std}}$
11	MC	0.344	59.028	412.691	218.829	59.242
11	MC simplicial	58.226	81.156	326.580	181.372	40.439
11	Batch MC simplicial	172.198	89.063	324.405	178.096	34.672
12	MC	0.654	68.220	355.558	199.278	47.175
12	MC simplicial	104.374	81.156	300.623	171.980	32.605
12	Batch MC simplicial	308.748	97.082	285.989	169.595	28.850
13	MC	1.420	82.863	380.341	226.818	48.482
13	MC simplicial	191.568	108.638	329.196	199.129	34.918
13	Batch MC simplicial	583.787	107.968	317.029	197.336	31.458
14	MC	2.649	92.604	392.457	234.484	45.704
14	MC simplicial	350.843	117.821	339.119	209.566	33.505
14	Batch MC simplicial	1,035.349	119.204	323.903	208.207	30.196
15	MC	5.803	101.726	348.771	220.790	37.342
15	MC simplicial	717.993	110.628	312.374	204.199	27.734
15	Batch MC simplicial	2,265.467	127.493	301.875	203.217	26.217

method and at each iteration, we record the time in seconds to build the grid (ti), the minimum ($\mathrm{E}_{\mathrm{min}}$), the maximum ($\mathrm{E}_{\mathrm{max}}$), the mean ($\mathrm{E}_{\mathrm{av}}$), and the standard deviation of the interpolation error ($\mathrm{E}_{\mathrm{std}}$). We take the simplicial method as our benchmark, and for each method, we calculate relative performance measures as the ratio of the corresponding measure to that of the simplicial. Furthermore, in additional to the relative computation times (in seconds), we also report the absolute times. The results are depicted in Table 4.

As expected, the pure MC method is the fastest as no additional optimization problem is solved except for the approximate dynamic Problems (10)–(16). Also, notice that as we conjectured, the batch MC simplicial scheme is about three times slower than its MC simplicial counterpart, as in the former, in each iteration, we generate three sample points, compared to one in the latter. For three-dimensional functions, the average CPU time of the simplicial method is lower than that of the MC simplicial scheme; for five-dimensional problems, the computation times are comparable. For dimensions equal to eight, the relative average CPU time of the MC simplicial method is only 3% that of the simplicial benchmark, which becomes practically intractable for ten-dimensional problems.

Accuracy-wise (average interpolation errors), except for the three-dimensional problems on which it performs better than the pure MC scheme, the simplicial approach features the worst performance. The batch MC simplicial is the top performer on all cases, followed by its MC simplicial counterpart; however, the difference grows smaller as the dimensions of the functions increase, and the MC simplicial scheme still remains about three times faster.

Table 6 Statistics pertaining to interpolation errors of Cobb–Douglas concave functions for fixed CPU time

		Total set CPU time in seconds									
		1		2		3		4		5	
n	Method	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$
3	MC	246.0443	0.0170	211.6483	0.0384	233.5693	0.0212	250.9239	0.0195	269.7663	0.0143
3	MC simplicial	0.4594	1.6358	0.4573	1.4919	0.4770	1.4466	0.4960	1.4137	0.4956	1.2604
3	Batch MC simplicial	0.1538	4.0873	0.1428	3.7745	0.1501	3.3776	0.1564	2.9679	0.1618	2.9184
3	Simplicial	1	1	1	1	1	1	1	1	1	1
		Total set CPU time in seconds									
		2		5		10		15		20	
n	Method	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$
5	MC	253.5391	0.0596	261.5483	0.0537	253.0394	0.0504	274.1172	0.0495	286.4562	0.0444
5	MC simplicial	1.0234	0.4520	0.8193	0.4695	0.7182	0.4565	0.7174	0.4703	0.7265	0.4514
5	Batch MC simplicial	0.5078	0.6683	0.3583	0.7244	0.2876	0.7524	0.2837	0.7323	0.2857	0.6707
5	Simplicial	1	1	1	1	1	1	1	1	1	1
		Total set CPU time in seconds									
		5		10		15		20		25	
n	Method	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$	$\overline{K}$	$\overline{\mathrm{E}}_{\mathrm{av}}$
8	MC	199.6836	0.1717	400.3906	0.1307	619.5977	0.1124	768.6211	0.1036	969.9180	0.0969
8	MC simplicial	1.6094	0.5708	2.1797	0.4693	2.7422	0.4187	3.2773	0.3880	3.8164	0.3674
8	Batch MC simplicial	1.2031	0.6914	1.4023	0.6165	1.6016	0.5436	1.8047	0.5157	1.9883	0.4826
8	Simplicial	1	1	1	1	1	1	1	1	1	1

Table 7 Model parameters borrowed from Zéphyr et al. (2017)

Parameter	Lower limit	Upper limit
$\underline{s}_{it}$	150	600
$\bar{s}_{it}$	800	7,000
$\underline{u}_{it}$	0	0
$\bar{u}_{it}$	$0.05\bar{s}_{it}$	$1.5\bar{s}_{it}$
β_i	0.9	1.5
α_i	0.7	0.9
γ_i	$125u_i$	$170u_i$
q_{it}	500	3,000

Furthermore, we test the scalability of the randomized methods on the approximation of 11- to 15-dimensional Cobb–Dougblas concave functions. As above, we use all the methods, but the simplicial one (as it is intractable for such high-dimensional problems) to generate sample points of size $2^n + 100n$; then interpolation errors are calculated on samples of size $100n$. We also perform five replications with each method and calculate the same performance statistics, which are reported in Table 5. In addition to being tractable for all the cases, the hybrid methods still outperform the naïve approach (MC) in terms of the maximum and average interpolation errors; they also feature lower standard deviations of the approximation errors. The batch MC simplicial method still outperforms the MC simplicial one, but at the expense of higher computation time.

Lastly, instead of a fixed grid size per period, we fix the total CPU time to build the grids for three-, five- and eight-dimensional Cobb–Douglas concave functions. As above, the constructed grids are then used to calculate interpolation errors on samples of size $100n$. We report in Table 6 the relative average of the size of the grids $\overline{K}$, and interpolation errors $\overline{\mathrm{E}}_{\mathrm{av}}$ calculated over five replications. Being the fastest method, the MC approach generates grids, on average, between 200 and 970 times denser than the simplicial approach. Consistent with the observation from Table 4 that the simplest method is faster than the hybrid ones on three- and five-dimensional problems, the latter schemes generate grids less dense than the simplicial method (on average). As the hybrid methods are faster on the eight-dimensional problems, they generate grids of bigger size (on average) than the simplicial approach. The overwhelmingly larger size of the grids generated by the MC approach allows for smaller interpolations errors compared to all three other methods. While smaller average interpolations errors are obtained with simplicial scheme compared to the hybrid ones, the latter outperform the simplicial approach on the five- and eight-dimensional problems.

5.2.2 Simulated mid-term reservoir optimization problems

As in Zéphyr et al. (2015), for each plant $i = 1, \ldots, n$, we assume the production function to be of the form

Table 8 Statistics pertaining to the first period evaluations of the value functions for three reservoir configurations ($n = 4, 6, 8$) taking the simplicial method as our baseline

n	Method	Abs. CPU time	Relative averages				
			$\overline{ti}$	$\overline{V}_{1\,\min}$	$\overline{V}_{1\,\max}$	$\overline{V}_{1\,\mathrm{av}}$	$\overline{V}_{1\,\mathrm{std}}$
4	Monte Carlo (MC)	10.0974	0.2921	1.0001	1.0006	1.0006	0.9996
4	MC simplicial	30.7411	0.8892	1.0012	1.0012	1.0012	0.9979
4	Batch MC simplicial	76.9120	2.2248	1.0012	1.0012	1.0012	0.9977
4	Simplicial	34.5699	1	1	1	1	1
6	MC	105.5044	0.3082	0.9954	1.0021	1.0021	1.0131
6	MC simplicial	228.4980	0.6676	1.0026	1.0023	1.0026	1.0030
6	Batch MC simplicial	475.4854	1.3892	1.0029	1.0023	1.0026	1.0020
6	Simplicial	342.2702	1	1	1	1	1
8	MC	158.9624	0.0161	0.9949	0.9952	0.9951	1.0000
8	MC simplicial	433.8685	0.0439	1.0000	1.0000	1.0000	1.0000
8	Batch MC simplicial	882.3298	0.0893	1.0000	1.0000	1.0000	1.0000
8	Simplicial	9,879.5983	1	1	1	1	1

Table 9 Statistics pertaining to the first period evaluations of the value functions for three reservoir configurations ($n = 4, 6, 8$) using the MC method as our baseline

n	Method	Relative averages				
		$\overline{ti}$	$\overline{V}_{1\,\min}$	$\overline{V}_{1\,\max}$	$\overline{V}_{1\,\mathrm{av}}$	$\overline{V}_{1\,\mathrm{std}}$
4	MC	1	1	1	1	1
4	Simplicial MC	3.044443	1.001166	1.000615	1.000577	9.982757E-01
4	Batch simplicial MC	7.616980	1.001174	1.000578	1.000548	9.981081E-01
6	MC	1	1	1	1	1
6	Simplicial MC	2.165768	1.007247	1.000158	1.000503	9.899871E-01
6	Batch simplicial MC	4.506782	1.007507	1.000169	1.000530	9.890111E-01
8	MC	1	1	1	1	1
8	Simplicial MC	2.729379	1.005094	1.004841	1.004930	1.000000
8	Batch simplicial MC	5.550558	1.005094	1.004841	1.004930	1.000000

$$p_{it}(u_{it}) := \beta_i\big((u_{it} + \gamma_i)^{\alpha_i} - \gamma_i^{\alpha_i}\big),\ \beta_i > 0,\ \gamma_i \geq 0,\ 0 \leq \alpha_i \leq 1 \tag{23}$$

These production functions are linearized as in (10)–(16). Furthermore, we consider a planning horizon of length T=10, and three reservoir configurations in dimension $n = 4, 6, 8$, respectively. The problems' parameters, including bounds on the reservoir and water release levels, borrowed from Zéphyr et al. (2017), are shown in Table 7.

For each reservoir configuration, problem instances are randomly generated based on the experimental framework depicted in Table 7. To mitigate boundary

Table 10 Variation rate of the average first period value functions with the size of the grid for two reservoir configurations ($n = 4, 6$)

n	Method	Grid size				
		$2^n + 20n$	$2^n + 40n$	$2^n + 60n$	$2^n + 80n$	$2^n + 100n$
4	MC	–	1.00017	1.00007	1.00020	1.00022
4	MC simplicial	–	1.00012	1.00003	0.99999	1.00002
4	Batch MC simplicial	–	1.00008	1.00002	1.00001	1.00004
4	Simplicial	–	1.00009	1.00005	1.00006	1.00004
6	MC	–	1.00001	1.00001	1.00000	1.00000
6	MC simplicial	–	1.00002	1.00000	1.00000	1.00000
6	Batch MC simplicial	–	1.00002	1.00000	1.00000	1.00000
6	Simplicial	–	1.00007	1.00023	0.99999	1.00002

effects, the terminal value function, $\mathcal{V}_{T+1}(s_{T+1})$, is chosen as a concave function of the form (21).

In addition, in each period of the planning horizon, we use each method to generate samples of $2^n + 200n$ grid points to evaluate the approximate value function (10)–(16). Then, we randomly generate a sample of $1,000n$ initial reservoir levels and natural inflows. Next, as in Cervellera et al. (2017), the first period approximate problem is solved with each method for each state observation of the sample, and we record the minimum ($V_{1\min}$), the maximum ($V_{1\max}$), the average ($V_{1\text{av}}$), and the standard deviation ($V_{1\text{std}}$) of the first period value function evaluation. Five replications are performed for each case, then we calculate the average of each such statistic as well as the average time ($\overline{ti}$) to build the ten value functions. As in the above comparisons, we take the simplicial scheme as the benchmark method. The results (relative measures) are reported in Table 8 as well as the average absolute CPU times (in seconds).

Again, without any surprise, the pure MC method is the fastest. The average CPU time is relatively the same under the simplicial and its MC simplicial variant on the four-dimensional problems; the latter scheme features lower computational burden on the 6- and eight-dimensional instances. Both hybrid methods outperform the simplicial scheme on all the other metrics on the 6-dimensional problems. The performance of the methods is similar on the 8-dimensional problems, however at lower computational burden for the MC variant methods. Indeed, the CPU time of the MC method is approximately 2% of that of the simplicial scheme, and 4% and 9%, for the MC simplicial and its batch variant, respectively.

Let us now take the MC approach as our benchmark against the two hybrid schemes. Results depicted in Table 9 show that, on average, the MC approach is between two and three times as fast as the MC simplicial counterpart, and between 4.5 and 7.6 times faster than the batch simplicial MC. On the other hand, the two hybrid methods provide slightly better accuracy than the MC scheme.

In addition, we conduct an analysis of the sensitivity of the solution accuracy of the different methods to the size of the grids. We repeat the above experiments on four- and six-dimensional reservoir problems. The parameters are generated as in

Table 11 Statistics pertaining to the first period evaluations of the value functions for two reservoir configurations ($n = 4, 6$) using the simplicial method as our baseline and with fixed CPU time to build the value functions

		Total set CPU time in seconds							
		10		20		30		40	
n	Method	$\overline{K}_1$	$\overline{V}_{1_{\text{av}}}$	$\overline{K}_1$	$\overline{V}_{1_{\text{av}}}$	$\overline{K}_1$	$\overline{V}_{1_{\text{av}}}$	$\overline{K}_1$	$\overline{V}_{1_{\text{av}}}$
4	MC	2.473451	1.003137	2.315663	1.002313	2.161074	1.001947	2.047120	1.001562
4	MC simplicial	1.022124	1.002838	0.983133	1.002253	0.956376	1.001941	0.942408	1.001601
4	Batch MC simplicial	0.637168	1.001262	0.537349	1.001413	0.505034	1.001453	0.485602	1.001224
4	Simplicial	1	1	1	1	1	1	1	1
		Total set CPU time in seconds							
		50		100		150		200	
n	Method	$\overline{K}_1$	$\overline{V}_{1_{\text{av}}}$	$\overline{K}_1$	$\overline{V}_{1_{\text{av}}}$	$\overline{K}_1$	$\overline{V}_{1_{\text{av}}}$	$\overline{K}_1$	$\overline{V}_{1_{\text{av}}}$
6	MC	5.037500	1.008982	5.339506	1.009966	2.731495	1.007931	1.833631	1.007668
6	MC simplicial	2.675000	1.009712	4.305556	1.010693	2.718433	1.008498	2.064401	1.008599
6	Batch MC simplicial	1.771875	1.009169	2.586420	1.010249	1.593614	1.008249	1.201252	1.008350
6	Simplicial	1	1	1	1	1	1	1	1

Table 7. In each period, for each problem, we construct grids of sizes varying between $K_1 = 2^n + 20n$, and $K_5 = 2^n + 100n$, in increment of $20n$. As before, the first period value functions are solved for 1, 000n randomly generated initial reservoir levels and inflows, then the average is taken. For each grid size $K_j, j = 2, \dots, 5$, Table 10 depicts the relative average value function $\frac{\overline{V}_j}{\overline{V}_{j-1}}$. The results show that the average evaluations of the first period value functions are relatively steady.

We repeat same experiments as above on four- and six-dimensional simulated reservoir problems, now instead of fixing the grid size per time period, we set the total CPU time to build the ten value functions, which is split evenly between the ten periods. We also solve the first period approximate problem (10)–(16) for 1, 000n pairs of reservoir levels and natural inflows. The average relative first period value function $\overline{V}_{1_{\text{av}}}$, as well as the average relative first period grid size $\overline{K}_1$ under each method are depicted in Table 11. As above, we perform five replications.

Again, as expected, being the fastest, the MC method generates denser grids compared to the other approaches, ranging approximately between two times and five times the sizes of the grids generated by the simplicial scheme. For the four reservoir problems, on average, the size of the grids generated by the MC simplicial scheme varies between approximately 94.24% and 1.02% of those generated under the simplicial method. However, similarly to our previous observations, the simplicial

Table 12 Optimality gap of the first four-reservoir problem (Problem 1) described in Chow and Cortes-Rivera (1974), Murray and Yakowitz (1979) across the tested methods for different grid size

	Grid size					
Method	2^n+50n	2^n+100n	2^n+200n	2^n+300n	2^n+500n	$2^n+1000n$
MC	1.34%	0.88%	0.67%	0.56%	0.46%	0.40%
Simplicial MC	0.73%	0.49%	0.38%	0.27%	0.26%	0.18%
Batch simplicial MC	0.52%	0.31%	0.19%	0.19%	0.15%	0.11%
Simplicial	1.24%	0.55%	0.53%	0.36%	0.20%	0.15%

Table 13 CPU time in seconds to approximate the value functions for the first four-reservoir problem (Problem 1) reported in Chow and Cortes-Rivera (1974), Murray and Yakowitz (1979) for different grid size across the tested methods

	Grid size					
Method	2^n+50n	2^n+100n	2^n+200n	2^n+300n	2^n+500n	$2^n+1000n$
MC	0.6255	1.4599	4.2737	8.7034	21.3315	97.0282
Simplicial MC	32.6700	51.0181	196.8790	293.7830	487.9010	1,411.0400
Batch simplicial MC	52.9183	102.4470	225.4920	369.6240	1324.6700	3,024.9700
Simplicial	7.8186	17.4825	64.3685	156.5440	338.2210	1,082.5300

Table 14 Optimality gap of the second four-reservoir problem (Problem 2) described in Chow and Cortes-Rivera (1974), Murray and Yakowitz (1979), Moravej and Hosseini-Moghari (2016) across the tested methods for different grid size

	Grid size					
Method	2^n+50n	2^n+100n	2^n+200n	2^n+300n	2^n+500n	$2^n+1000n$
MC	2.86%	2.16%	1.81%	1.75%	1.35%	1.09%
MC simplicial	1.55%	1.21%	0.80%	0.69%	0.52%	0.33%
Batch MC simplicial	1.04%	0.61%	0.40%	0.28%	0.25%	0.19%
Simplicial	1.97%	1.73%	0.89%	0.84%	0.48%	0.28%

Table 15 CPU time in seconds to approximate the value functions for the second four-reservoir problem (Problem 2) reported in Chow and Cortes-Rivera (1974), Murray and Yakowitz (1979), Moravej and Hosseini-Moghari (2016) for different grid size across the tested methods

	Grid size					
Method	2^n+50n	2^n+100n	2^n+200n	2^n+300n	2^n+500n	$2^n+1000n$
MC	0.6298	1.3913	3.8732	7.7293	18.9845	137.1070
MC simplicial	16.7826	35.4257	87.7464	136.9160	274.4680	1,118.660
Batch MC simplicial	48.6417	102.1330	225.0250	370.3550	725.1740	2,259.890
Simplicial	7.9736	17.7167	81.9990	110.7210	165.5750	954.704

Table 16 Optimality gap of the ten-reservoir problem described in Chow and Cortes-Rivera (1974), Murray and Yakowitz (1979), Moravej and Hosseini-Moghari (2016) across the tested methods for different grid size

	Grid size					
Method	$2^n + 50n$	$2^n + 100n$	$2^n + 200n$	$2^n + 300n$	$2^n + 500n$	$2^n + 1000n$
MC	3.15%	3.07%	3.00%	2.73%	2.39%	2.39%
MC simplicial	3.99%	3.21%	2.45%	2.14%	2.20%	1.61%
Batch MC simplicial	4.24%	3.31%	2.73%	2.59%	1.79%	1.31%
Simplicial	n/a	n/a	n/a	n/a	n/a	n/a

Table 17 CPU time in seconds to approximate the value functions for the ten-reservoir problem reported in Chow and Cortes-Rivera (1974), Murray and Yakowitz (1979), Moravej and Hosseini-Moghari (2016) for different grid size across the tested methods

	Grid size					
Method	$2^n + 50n$	$2^n + 100n$	$2^n + 200n$	$2^n + 300n$	$2^n + 500n$	$2^n + 1000n$
MC	25.401	59.020	124.719	141.087	332.738	1,993.690
MC simplicial	132.705	531.597	745.877	1,135.720	2,603.180	8,319.800
Batch MC simplicial	311.661	667.861	1899.350	4107.330	6143.250	27,818.900
Simplicial	n/a	n/a	n/a	n/a	n/a	n/a

algorithm is slower on the six-dimensional problems; the MC simplicial approach generates grids, on average, more than twice as dense as those generated under the simplicial counterpart. In terms of accuracy, all three other methods sightly outperform the simplicial one. The MC and MC simplicial schemes exhibit similar accuracy level, with a slight edge for the MC simplicial on the six-dimensional problems.

5.3 Performance comparisons on three literature reservoir optimization problems

Our last comparison setting is three literature reservoir optimization problems: two four-dimensional and one ten-dimensional problems. The planning horizons are one year divided into monthly time steps. These problems were designed to assess the effectiveness of reservoir optimization solution methods. For details about their characteristics, please see Chow and Cortes-Rivera (1974), Murray and Yakowitz (1979), Moravej and Hosseini-Moghari (2016). The main difference between the two four-dimensional problems is that in one of them (hereafter Problem 1), the release decisions are less constrained, and the upper bounds on the reservoirs are stationary (do not vary with time), in contrast with the second one (Problem 2).

Table 18 Optimality gap of the first four-reservoir problem (Problem 1) described in Chow and Cortes-Rivera (1974), Murray and Yakowitz (1979) across the tested methods for different total CPU time budget

	Total set CPU time in seconds					
	360		480		600	
Method	$\overline{K}_t$	ϵ	$\overline{K}_t$	ϵ	$\overline{K}_t$	ϵ
MC	3228	1.696%	3,317	1.5792%	3,700	1.5837%
MC simplicial	2096	0.789%	2,879	0.2628%	1807	0.2411%
Batch MC simplicial	491	0.229%	675	0.5363%	774	0.1169%
Simplicial	1761	0.520%	2215	0.3152%	2524	0.3579%

Table 19 Optimality gap of the second four-reservoir problem (Problem 2) described in Chow and Cortes-Rivera (1974), Murray and Yakowitz (1979), Moravej and Hosseini-Moghari (2016) across the tested methods for different fixed CPU time budget

	Total set CPU time in seconds					
	360		480		600	
Method	$\overline{K}_t$	ϵ	$\overline{K}_t$	ϵ	$\overline{K}_t$	ϵ
MC	1696	0.4782%	1925	0.401%	3130	0.4114%
MC simplicial	483	0.4755%	905	0.355%	1294	0.3569%
Batch MC simplicial	186	0.5691%	343	0.323%	548	0.2506%
Simplicial	813	0.3907%	1389	0.197%	1760	0.0612%

Table 20 Optimality gap of the ten-reservoir problem described in Chow and Cortes-Rivera (1974), Murray and Yakowitz (1979), Moravej and Hosseini-Moghari (2016) across the tested methods for different CPU time budget

	Total set CPU time in seconds					
	360		480		600	
Method	$\overline{K}_t$	ϵ	$\overline{K}_t$	ϵ	$\overline{K}_t$	ϵ
MC	4520	1.1303%	6571	2.5484%	8601	0.1422%
MC simplicial	2705	2.2250%	3359	3.0712%	3899	2.5599%
Batch MC simplicial	1734	7.2165%	2054	3.6955%	2332	3.7240%
Simplicial	n/a	n/a	n/a	n/a	n/a	n/a

In all three problems, the first period reservoir level (s_1) is fixed, similarly for the terminal one (s_{13}). Though these constraints can easily be handled in a multi-period model, this is not the case in dynamic programming-like methods, as in period $t = 12$, the algorithms can pick a reservoir level that violates the terminal value constraints on the reservoir levels. Similarly, in any period t, the bounds may also be violated. We mitigate this issue by introducing linearized penalty functions in the objective functions. We calibrate the penalty coefficients through trial-and-errors, until we obtain solutions that meet all the constraints (solving the value functions forward in time as explained below).

We build the value functions moving backward in time. Then, starting from the initial reservoir level, we solve the value functions forward in time, using the previous period suboptimal reservoir level as initial value. In each time period, we calculate the suboptimal current period objective value (say the current period suboptimal production in our context). Thus, the suboptimal value of the problem is the sum of such suboptimal objective values.

Under each method, first, we use different grid sizes to build the value functions, as illustrated in Tables 12, 13, 14, 15, 16 and 17. Under the simplicial method, each problem is solved once (one backward and one forward steps), as the problems are deterministic and the simplicial method is also a deterministic algorithm. Under the hybrid methods, we perform five replications, and calculate the averages (solution times and suboptimal values).

Tables 12, 14, and 16 report the optimality gaps (difference between the known optimal values and the suboptimal ones obtained with the methods) for each grid size and each method. No results are reported for the simplicial method for the largest problem (ten-dimensional), which proved intractable for this method (we stopped the algorithm after several hours spent in the last period recursion).

The optimality gaps decreases as the grid size increases, regardless of the method. Overall, the batch MC simplicial scheme consistently exhibits the lowest optimality gaps, followed by the MC simplicial method, though the latter is outperformed by the simplicial approach on the two four-dimensional problems for the two largest grid sizes. The pure MC method consistently features the highest optimality gaps. The associated CPU times (in seconds) are reported in Tables 13, 15, 17, respectively.

Second, as for the simulated problems, we also compare the performance of the methods by fixing a total computation time to build the 12 value functions. The total time is evenly split between the 12 periods. In addition to the optimality gap, ϵ, we also calculate the average grid size per period, $\overline{K}_t$. For each problem, we consider three total CPU times. The results are reported in Tables 18, 19 and 20. Since the MC approach is the fastest, it generates the highest average number of grid points per period. Detailed analysis of the results (not reported here) show that in the backward pass, the MC method alternates between very high and relatively low number of grid points. This is due to the fact that when a very dense grid is generated in a period t, then the computational burden of the approximate problem (10)–(16) increases in period $t-1$, as the grid generated in period t is used to approximate the value function in the previous period, as well as to interpolate the next period reservoir levels. Thus, a low-size grid is generated in period $t-1$. If this behaviour, combined with the uniform discretization of the reservoir level space (not taking into account information about the curvature of the value function), seems to put the MC scheme at a disadvantage in terms of optimality gap compared to the other three methods on the two four-reservoir problems, this is not the case on the ten-dimensional one, on which it outperforms the other methods. The simplicial method systematically outperforms the two MC variants on the second four-reservoir problem, but it is intractable for the largest dimension problem, due to the exponential complexity of the initial partitioning of the hypercube into simplices.

We close the numerical experiments section with the following remarks about the scalability of the hybrid methods to problems with more than ten reservoirs, the largest size problems solved in this work.

On the one hand, in addition to function evaluations, the complexity of the hybrid methods depends on the sizes of the linear programs (17) and (19). The former program features K decision variables and $n+1$ constraints, and the latter at most $2n+2$ decision variables and at most $2n+2$ constraints; K, being the size of the grid at each iteration. At each iteration, the MC simplicial scheme solves each of the two programs once, as one additional grid point is added to the grid. Under the batch MC variants, at each iteration, each program is called m times; m being the size of the randomly generated candidate sample points. Also notice that the size of Program (17) varies with K, which is not the case of Program (19). Results of extra numerical experiments not reported in this paper suggested that the computational burden of the hybrid methods is proportional to the grid sizes. Thus, we believe the proposed methods are scalable to more than ten reservoirs.

On the other hand, the value functions are typically built off-line. Then, in each time period, once the pair $(\boldsymbol{s}_t, \boldsymbol{q}_t)$ is observed, the approximate problem (10)–(16) is solved online using the previously built value function to make operational decisions. Solving this problem is relatively fast. The value functions are updated off-line as more data (natural inflows) become available.

6 Conclusions

This work has revisited a simplicial approximate stochastic dynamic programming scheme presented in Zéphyr et al. (2015) for the mid-term sub-optimal operations of multi-period multi-reservoir systems. This iterative method relies on the exhaustive examination of a list of created simplices, whose vertices define grid points at which the value functions are evaluated at each period. The scheme is limited by the computational burden of partitioning a hypercube into simplices.

We have proposed two hybrid methods that combine random sampling strategies with the approach proposed in Zéphyr et al. (2015) to locally estimate the approximation error. Simulation results of randomly generated and three literature mid-term reservoir management test problems showed that, compared to the simplicial methods, the hybrid methods seem to offer a good trade-off between solution time and accuracy, in particular when the state space dimension is greater than nine. Approximation of functions of dimension up to 15 within reasonable computation time illustrated the potential scalability of the proposed randomized methods, which might further be leveraged through parallelization.

Appendices

Appendix 1: Simplicial partioning of hypercubes

Definition 1 Let S be some set in the Euclidean space $\boldsymbol{R}^n$, its *affine envelope* is the set of all affine combinations of points in S, or equivalently the smallest affine set that contains S, i.e., the set $\mathbf{aff}\, S := \left\{ \sum_{i=1}^{k} \lambda_i x^i \mid x^i \in S,\ i = 1, \ldots, k,\ \sum_{i=1}^{k} \lambda_i = 1 \right\}$; its convex envelope is the set of all convex combinations of points in S, or equivalently, the smallest convex set that contains S, i.e., the set $\mathbf{conv}\ S := \left\{ \sum_{i=1}^{k} \lambda_i x^i \mid x^i \in S, \lambda_i \geq 0,\ i = 1, \ldots, k,\ \sum_{i=1}^{k} \lambda_i = 1 \right\}$.

Furthermore,

Definition 2 A closed convex set $\mathcal{B} \in \boldsymbol{R}^n$ is called a simplex if it is the convex envelope of $n + 1$ affinely independent points $\boldsymbol{s}^1, \boldsymbol{s}^2, \ldots, \boldsymbol{s}^{n+1}$ in $\boldsymbol{R}^n$, i.e., $\mathcal{B} := \mathbf{conv}\ \{\boldsymbol{s}^1, \ldots, \boldsymbol{s}^{n+1}\} := \left\{ \sum_{i=1}^{n+1} \lambda_i \boldsymbol{s}^i \mid \lambda_i \geq 0,\ i = 1, \ldots, n+1,\ \sum_{i=1}^{n+1} \lambda_i = 1 \right\}$.

As examples, a one-dimensional simplex is a line segment, a two-dimensional simplex a triangle, and a three-dimensional simplex a tetrahedron.

Partitioning the hyperrectangular state set S_t into simplices entails two steps, namely, (i) its initial partitioning into simplices; and (ii) the iterative subdivision of existing simplices until a prescribed criterion is met. The popular *Kuhn triangulation*, implemented in this work for our benchmark method, partitions S_t into $n!$ initial simplices (Moore 1992; Munos and Moore 2002). By a simple change of scale, each point $\boldsymbol{s}_t \in S_t$ can be mapped to a point $\mathbf{0} \leq \boldsymbol{x}_t \leq \boldsymbol{e}$; $\boldsymbol{e}$ being an n-vector filled with 1's. Then each simplex in the Kuhn triangulation corresponds to one possible permutation, p, of the indices $(1, \ldots, n)$ of the

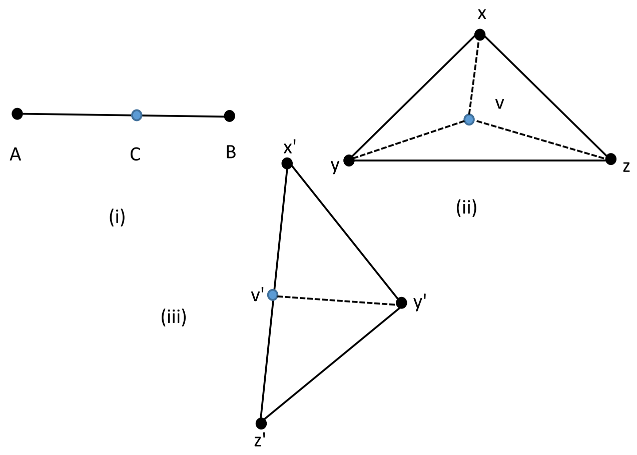


Fig. 9 Illustrative examples of simplicial subdivision

dimension of $\boldsymbol{x}_t$, and is given by the set of points $\boldsymbol{x}_t$ whose coordinates satisfy the inequalities $0 \leq x_t^{p(1)} \leq x_t^{p(2)} \leq \ldots \leq x_t^{p(n)} \leq 1$ (Davies 1997).

A less expensive strategy, called *Delaunay triangulation*, partitions a hyperrectangle into at most $\mathcal{O}(N^{\lceil \frac{n}{2} \rceil})$ simplices, where $N = 2^n$ Žilinskas and Žilinskas (2002). In Zéphyr et al. (2015), starting with its 1-dimensional faces (line segments), k-dimensional faces of the hyperrectangle are iteratively lifted into $k+1$-dimensional simplices until the hyperrectangle is partitioned into n-dimensional simplices. The complexity of this proposal is more than exponential in the dimension n of the hyperrectangle.

If either the Kuhn or the Delaunay triangulation is used, the initial step generates a grid of 2^n points, i.e., the vertices of the hyperrectangle, at which the approximate value function (10)–(16) is evaluated. If one wants to densify the initial grid in the hope of improving the approximation, the initial simplices can iteratively be subdivided into smaller ones. A popular technique used in global optimization consists in bisecting edges of simplices based on their diameter or local Lipschitz lower bounds (e.g., Žilinskas and Žilinskas 2002; Paulavičius and Žilinskas 2014). Another population strategy, called radial or ω-subdivision (Žilinskas and Žilinskas 2002), consists in choosing a point in some d-dimensional subset of a simplex $\mathcal{B}$, $d = 1, \ldots, n-1$, called a *face* of $\mathcal{B}$, and creating subsimplices around this point (e.g., Horst 1976; Tuy 1991; Bomze and Eichfelder 2013; Zéphyr et al. 2015).

More specifically, let $\mathcal{B} \subset \boldsymbol{R}^n$ be an n-dimensional simplex generated by the $n+1$ affinely independent points $\{\boldsymbol{s}^1, \boldsymbol{s}^2, \ldots, \boldsymbol{s}^{n+1}\}$, and denote $\boldsymbol{S}_{\mathcal{B}} := [\boldsymbol{s}^1, \boldsymbol{s}^2, \ldots, \boldsymbol{s}^{n+1}] \in \boldsymbol{R}^{n\times(n+1)}$ the full row rank associated matrix. It follows from Definition 2 that a point $\boldsymbol{s}$ lives in $\mathcal{B}$ if and only if the system

$$\begin{bmatrix} \boldsymbol{S}_{\mathcal{B}} \\ \boldsymbol{e}^\top \end{bmatrix} \lambda = \begin{pmatrix} \boldsymbol{s} \\ 1 \end{pmatrix}, \ \lambda \geq \boldsymbol{0}, \tag{24}$$

has a unique solution $\lambda \in \boldsymbol{R}^{n+1}$. In addition, let $\mathcal{B}(\boldsymbol{s})$ be a subset of $\{1, \ldots, n+1\}$ such that in Eq. (24), $\lambda_j > 0$, $j \in \mathcal{B}(\boldsymbol{s})$. Let $\boldsymbol{S}_{\mathcal{B}(\boldsymbol{s})^j}$ be the $n \times (n+1)$ matrix obtained by replacing the j^{th} column of $\boldsymbol{S}_{\mathcal{B}}$, $j \in \mathcal{B}(\boldsymbol{s})$, with the point $\boldsymbol{s}$, which we assume is not a vertex of the simplex. Clearly, the columns of $\boldsymbol{S}_{\mathcal{B}(\boldsymbol{s})^j}$ are affinely independent; as a result, their convex envelope defines a simplex. This way, $\mathcal{B}$ is subdivided into d simplices, d being the cardinality of $\mathcal{B}(\boldsymbol{s})$.

Illustrative examples of simplicial subdivision are provided in Fig. 9. In case (i), the *division point* C is located in the *relative interior* of the simplex $[A, B]$, which is subdivided into two simplices, namely $[A, C]$ and $[C, B]$. In case (ii), the division point, v, lies in the relative interior of the simplex $[x, y, z]$; the latter is partitioned into three simplices. Lastly, the simplex $[x', y', z']$ is partioned into two simplices, since the division point v' is located on the line segment $[x', y']$.

Appendix 2: Proof of proposition

Proof of Proposition 2 We will derive our complexity results in two steps. First, we will show that the error bound on a simplex $\mathcal{B}$ can be approximated by a quadratic function of the function values at its vertices. This result will be used next to show that the number of simplices required to obtain the desired threshold on the approximation error is proportional to an exponential factor.

In (19), let us collect the evaluations of the function at the vertices of simplex $\mathcal{B}$ in the vector $\boldsymbol{f}_{\mathcal{B}} := \left(f^1, \dots, f^{n+1}\right)^\top$; similarly, let us define the vector $\boldsymbol{\lambda}_{\mathcal{B}} := \left(\lambda_1, \dots, \lambda_{n+1}\right)^\top$. Substituting $\boldsymbol{s}$ with its expression in the inequalities, and rearranging terms, we see that (19) is the same as:

$$\begin{aligned} \overline{E}_{\mathcal{B}} := \;& \max_{\phi,\boldsymbol{\lambda}_{\mathcal{B}}} \; \phi - \boldsymbol{f}_{\mathcal{B}}^\top \boldsymbol{\lambda}_{\mathcal{B}} \\ \text{s.t.} \;& \phi - {\boldsymbol{g}^1}^\top \boldsymbol{S}_{\mathcal{B}} \boldsymbol{\lambda}_{\mathcal{B}} \le f^1 - {\boldsymbol{g}^1}^\top \boldsymbol{s}^1, \\ & \quad\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ & \phi - {\boldsymbol{g}^{n+1}}^\top \boldsymbol{S}_{\mathcal{B}} \boldsymbol{\lambda}_{\mathcal{B}} \le f^{n+1} - {\boldsymbol{g}^{n+1}}^\top \boldsymbol{s}^{n+1} \\ & \boldsymbol{e}^\top \boldsymbol{\lambda}_{\mathcal{B}} = 1, \; \boldsymbol{\lambda}_{\mathcal{B}} \ge \boldsymbol{0}. \end{aligned} \tag{25}$$

For simplicity, let us relax the non-negativity constraints on $\boldsymbol{\lambda}_{\mathcal{B}}$, allowing the division point to be located outside the simplex, and thus overestimating the error bound $\overline{E}_{\mathcal{B}}$. The relaxed problem can be re-written in compact form as:

$$\begin{aligned} \overline{E}'_{\mathcal{B}} := \;& \max_{\phi,\boldsymbol{\lambda}_{\mathcal{B}}} \; \phi - \boldsymbol{f}_{\mathcal{B}}^\top \boldsymbol{\lambda}_{\mathcal{B}} \\ \text{s.t.} \;& -\boldsymbol{G}_{\mathcal{B}} \boldsymbol{S}_{\mathcal{B}} \boldsymbol{\lambda}_{\mathcal{B}} + \boldsymbol{e}\phi \le \boldsymbol{f}_{\mathcal{B}} - \boldsymbol{d}\boldsymbol{S}_{\mathcal{B}}^\top, \; \boldsymbol{e}^\top \boldsymbol{\lambda}_{\mathcal{B}} = 1. \end{aligned} \tag{26}$$

where $\boldsymbol{G}_{\mathcal{B}} := \left(\boldsymbol{g}^1, \dots, \boldsymbol{g}^{n+1}\right)^\top$, and $\boldsymbol{d}$ is an $(n+1) \times (n+1)$ block diagonal matrix filled with the $\boldsymbol{g}^i$'s, $i = 1, \dots, n+1$, on the main diagonal, and with an n-dimensional zero-vector in each off-diagonal position. Furthermore, assuming that at optimality all the inequalities of (26) are binding, with the only risk of underestimating the error bound, we have the solution:

$$\begin{pmatrix} \boldsymbol{\lambda}_{\mathcal{B}} \\ \phi \end{pmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{e} \\ \boldsymbol{e}^\top & 0 \end{bmatrix}^{-1} \begin{pmatrix} \boldsymbol{f}_{\mathcal{B}} - \boldsymbol{h} \\ 1 \end{pmatrix}, \tag{27}$$

where $\boldsymbol{A} := -\boldsymbol{G}_{\mathcal{B}} \boldsymbol{S}_{\mathcal{B}}$, and $\boldsymbol{h} := \boldsymbol{d}\boldsymbol{S}_{\mathcal{B}}^\top$.

It is easy to see that $\begin{bmatrix} \boldsymbol{A} & \boldsymbol{e} \\ \boldsymbol{e}^\top & 0 \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{A}^{-1} - c\boldsymbol{A}^{-1}\boldsymbol{e}\boldsymbol{e}^T\boldsymbol{A}^{-1} & c\boldsymbol{A}^{-1}\boldsymbol{e} \\ c\boldsymbol{e}^\top \boldsymbol{A}^{-1} & -c \end{bmatrix}$, where the constant $c := \boldsymbol{e}^\top A^{-1} \boldsymbol{e}$. We then have:

$$\overline{E}'_{\mathcal{B}} := \phi - \boldsymbol{f}_{\mathcal{B}}^\top \boldsymbol{\lambda}_{\mathcal{B}} = \begin{pmatrix} -\boldsymbol{f}_{\mathcal{B}} \\ 1 \end{pmatrix}^\top \begin{bmatrix} \boldsymbol{A}^{-1} - c\boldsymbol{A}^{-1}\boldsymbol{e}\boldsymbol{e}^T\boldsymbol{A}^{-1} & c\boldsymbol{A}^{-1}\boldsymbol{e} \\ c\boldsymbol{e}^\top \boldsymbol{A}^{-1} & -c \end{bmatrix} \begin{pmatrix} \boldsymbol{f}_{\mathcal{B}} - \boldsymbol{h} \\ 1 \end{pmatrix}. \tag{28}$$

In (28), let $\boldsymbol{B} := \boldsymbol{A}^{-1} - c\boldsymbol{A}^{-1}\boldsymbol{e}\boldsymbol{e}^T\boldsymbol{A}^{-1}$, an $(n+1) \times (n+1)$ matrix, $\boldsymbol{b} := c\boldsymbol{A}^{-1}\boldsymbol{e}$, an $n+1$-dimensional column vector, and $\boldsymbol{\beta}^\top := c\boldsymbol{e}^\top \boldsymbol{A}^{-1}$, an $n+1$-dimensional row vector. With some algebra, if follows from (28) that:

$$\overline{E}'_{\mathcal{B}} := -\boldsymbol{f}_{\mathcal{B}}^{\top}\boldsymbol{B}\boldsymbol{f}_{\mathcal{B}} + \left((\boldsymbol{B}\boldsymbol{h})^{\top} + \boldsymbol{\beta} + \boldsymbol{b}^{\top}\right)\boldsymbol{f}_{\mathcal{B}} - \boldsymbol{\beta}\boldsymbol{h} - c. \tag{29}$$

Thus, we see in (29) that the error on simplex $\mathcal{B}$ is a quadratic function of $\boldsymbol{f}_{\mathcal{B}} \in \boldsymbol{R}^{n+1}$.

Now, we need to find the number of required simplices to guarantee that $\overline{E}'_{\mathcal{B}} \le \overline{E}_0$. Though this answer is not straightforward, we argue that this number may depend upon the dimension n of the state space and the size of the generated simplices. Let $\mathcal{B}(1)$ be a unit-volume simplex in $\boldsymbol{R}^n$, and denote $\boldsymbol{S}_{\mathcal{B}}(1)$ the matrix formed by its vertices. In addition, assume this simplex may be scaled by a factor κ to a higher volume simplex $\mathcal{B}(\kappa)$, i.e., $\mathcal{B}(\kappa) \sim \kappa\mathcal{B}(1)$.

Similarly, assume the matrix of the vertices of $\mathcal{B}(1)$ may be scaled by the same factor κ to the matrix of $\mathcal{B}(\kappa)$, i.e., $\boldsymbol{S}_{\mathcal{B}}(\kappa) \sim \kappa\boldsymbol{S}_{\mathcal{B}}(1)$. Therefore, we can take as an estimate of the required number of simplices, $N_n(\overline{E}_0)$, the ratio of the volume of the hyperrectange S to the volume of a simplex $\mathcal{B}(\kappa)$, such that the error on that simplex does not exceed the desired threshold, i.e.,

$$N_n(\overline{E}_0) =: \max_{\kappa} \left\{ \frac{\operatorname{Vol}(S)}{\operatorname{Vol}(\mathcal{B}(\kappa))} \mid \overline{E}'_{\mathcal{B}(\kappa)} \le \overline{E}_0 \right\}. \tag{30}$$

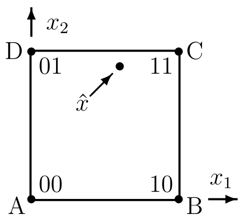
Lastly, ignoring the lower order terms in (29), we see that the error bound is a quadratic function of κ, such that $\overline{E}'_{\mathcal{B}}(\kappa) \sim k_1\kappa^2$, where k_1 is a proportionality constant. As a result, to guarantee the desired error threshold $\overline{E}_0$, we must have $k_1\kappa^2 \le \overline{E}_0$, or

$$\kappa \le \sqrt{\frac{\overline{E}_0}{k_1}}. \tag{31}$$

The volume of a simplex $\mathcal{B}(\kappa)$ being $\operatorname{Vol}(\mathcal{B}(\kappa)) = \frac{1}{n!}\begin{vmatrix} \kappa\boldsymbol{S}_{\mathcal{B}}(1) \\ \boldsymbol{e}^{\top} \end{vmatrix} = \frac{\kappa^n}{n!}\begin{vmatrix} \boldsymbol{S}_{\mathcal{B}}(1) \\ \boldsymbol{e}^{\top} \end{vmatrix} := k_2\frac{\kappa^n}{n!}$, it follows from the inequality (31) that to guarantee the prescribed error bound, $\overline{E}_0$, the volume $\operatorname{Vol}(\mathcal{B}(\kappa))$ should be of the order $\frac{k_2}{n!}\left(\frac{\overline{E}_0}{k_1}\right)^{n/2}$. Thus, the total number of such simplices should be:

Table 21 Data for quadratic example in two dimensions

Vertices	A	B	C	D	Sample $\hat{x}$
s_1	0	1	1	0	0.6
s_2	0	0	1	1	0.9
$f(s_1, s_2)$	0	7	12.5	10.5	11.835



$$N_{\mathcal{B}} := \frac{\mathrm{Vol}(S)}{\mathrm{Vol}(\mathcal{B}(\kappa))} = \mathrm{Vol}(S)\frac{n!}{k_2}\left(\frac{k_1}{\overline{E}_0}\right)^{n/2}, \tag{32}$$

which is of the order $\mathcal{O}\left(\frac{\mathrm{Vol}(S)n!}{(n+1)\overline{E}_0^{n/2}}\right)$. □

Appendix 3: Comparison of the original and hybrid simplicial methods

To summarize, we make a brief comparison between the original and hybrid simplicial methods. Conceptually, the original simplicial method makes an initial list of simplices using the extreme points of the state set as vertices, for instance via Kuhn's triangulation. The function to be approximated is evaluated at the vertices, and corresponding subgradients are calculated. For each simplex in the list, an error bound is obtained by solving Eq. (19) which also returns a division point. Then new vertices are iteratively added by selecting the simplex with largest error bound in the current list, adding its division point as a new vertex where the function and subgradient are evaluated, deleting the simplex from the list, replacing it with the new simplices obtained following its division, and evaluating the error bounds and division points of the new simplices, and so on. Once a sufficiently large list of simplices has been obtained, it provides a partition of the state set S. The value function, call it $f(\boldsymbol{s})$ for simplicity, at any given point $\boldsymbol{s} \in S$ is approximated by finding a simplex in the list containing the point $\boldsymbol{s}$ and interpolating the (known) function values at its vertices.

By contrast, the hybrid methods iteratively build a list of vertices but do not make an explicit list of simplices. This way, the value function $f(\boldsymbol{s})$ is approximated at any point $\boldsymbol{s} \in S$ by solving the linear program (17) whose optimal basis identifies a set of vertices that define a simplex containing the point $\boldsymbol{s}$. Since the linear program selects the largest interpolated value among all feasible simplices (not necessary full-dimensional) containing the point $\boldsymbol{s}$, it may provide a better approximation of $f(\boldsymbol{s})$ than the original simplicial method in which there is only one full-dimensional simplex containing the point $\boldsymbol{s}$. The list of vertices is obtained iteratively by sampling a point $\hat{\boldsymbol{s}}$ at random in the state set S, using Eq. (17) to identify an optimal simplex containing the point $\hat{\boldsymbol{s}}$, then using Eq. (19) to find an error bound and a division point for this simplex, and adding this division point as a new vertex in the list, and so on.

In the original simplicial method, by construction the largest error bound in the list of simplices provides an upper bound on the approximation error for all points $\boldsymbol{s} \in S$ although it might be somewhat overestimated. In the hybrid methods, the error bounds are tighter, since, as aforementioned, the largest interpolated value is taken among all feasible simplices.

To illustrate these ideas, Let us consider the 2-dimensional concave quadratic function:

$$f(s_1, s_2) = 9s_1 + 15s_2 - 2s_1^2 - 5s_1s_2 - (9/2)s_2^2.$$

The state set S is the unit square whose vertices are given counterclockwise in Table 21 with their coordinates and function values:

Suppose a Kuhn triangulation was used to partition S into the two simplices $ABC :=$ ◿ and $ACD :=$ ◸. Then Eq. (19) would yield an error bound of 3.6964 in both cases with a division point at $s_1 = s_2 = 0.6786$ for ABC and at $s_1 = s_2 = 0.3214$ for ACD. So the original simplicial method would divide one of the two simplices ABC or ACD at its division point.

By comparison, the MC simplicial method would first sample a point $\hat{s} \in S$ at random and then would use Eq. (17) to find a simplex over which the interpolation of the function is the largest at that point $\hat{s}$. Unlike the original method in which only the simplices already in the list would be considered, in the MC simplicial scheme all possible simplices would be taken into account. For example, suppose the coordinates of the sampled point $\hat{s}$ happened to be $\hat{s}_1 = 0.6$ and $\hat{s}_2 = 0.9$, the supporting simplex found by Eq. (17) would be $BCD :=$ ◹ with an interpolated value of 11.15. Next, Eq. (19) applied to simplex BCD would find an error bound of 1.8 with a division point at coordinates $s_1 = 1$ and $s_2 = 0.6$.

We notice that if the sampled point $\hat{s}$ had been interpolated with simplex ACD from the list, instead of BCD, its interpolated value would have been smaller, i.e., 10.65 instead of 11.15.

Appendix 4: MLE estimation of the upper limit of TL(0,b)

AAdapting the approach of Lamond and Zéphyr (2021), it is possible to find a MLE for parameter b by solving a nonlinear equation. If a random variable X has a right-angle triangular distribution on the interval $[0, b]$ with mode at the origin, then its density function is

$$g(x) = \begin{cases} \frac{2(b-x)}{b^2} & \text{if } 0 \le x \le b, \\ 0 & \text{else,} \end{cases}$$

so the likelihood function for an observed sample x is

$$L(x|b) = \frac{2^m \prod_{i=1}^m (b - x_i)}{b^{2m}}$$

Then with $\ln L(x|b)$ the first-order optimality condition for the MLE of parameter b is the nonlinear equation

$$\sum_{i=1}^{m} \frac{1}{b - x_i} - \frac{2m}{b} = 0, \tag{33}$$

which needs to be solved numerically, except in special cases.

Proposition 5 *Let b^* be the unique solution of Eq.* (33) *and let* $x_{(m)} = \max_{i=1,\dots,m} x_i$. *Then*

$$\frac{m+1}{m} \times x_{(m)} \le b^* \le 2x_{(m)}. \tag{34}$$

Proof When $x_{(m)} > 0$, the bounds in Eq. (34) are attained in the extreme cases with $x_1 = \cdots = x_{m-1} = 0$ for the lower bound, and $x_1 = \cdots = x_m = x_{(m)}$ for the upper bound. In the limiting case when all observations are 0, i.e. $x_{(m)} = 0$, then Eq. (34) implies that $\hat{b} = 0$ (the unbiased point estimate of b) which is expected since the density function goes to ∞ when $b \to 0$. In order to show that b^* is between the bounds for any sample x, we argue that b^* increases when any observation x_i increases without changing $x_{(m)}$. To do this, we rewrite Eq. (33) as

$$G(x, b) = \sum_{i=1}^{m} \frac{1}{1 - x_i/b} - 2m = 0. \tag{35}$$

We see in Eq. (35) that the function $G(x,\ b)$ is increasing with x_i and that it is decreasing with b. If $G(x, b) = 0$ for given x and b, then having $x_i' = x_i + \epsilon$, say, implies that $G(x', b) > 0$ so we must have $b' < b$ in order for $G(x', b') = 0$. This monotonicity property of b^* thus implies that for any sample x there must be an increasing trajectory from the lower bound to the upper bound that goes through x.

□

The bounds provided by Proposition 5 can be used for initializing a search algorithm for solving Eq. (33). They also imply that the MLE is strictly larger than $x_{(m)}$. However it is not obvious what is the expected value of b^* in general, although in the special case with $m = 1$ it is equal to $2b/3$. Monte Carlo simulations indicate that b^* has a smaller variance than $\hat{b}$ so that, even for small samples, the mean square error of b^* is slightly smaller than that of $\hat{b}$. But in practice the unbiased estimator $\hat{b}$ seems attractive due to its ease of computation. However, the MLE computation might be justified when it saves the effort of obtaining a larger sample.

Acknowledgements The authors acknowledge constructive comments from two anonymous reviewers that helped improve the quality of the paper.

Author contributions LZ contributed to the methodology, performed the numerical experiments and contributed to the write-up. BFL contributed to the methodology and the write-up. PL contributed to the methodology.

Funding The authors did not receive support from any organization for the submitted work, and have no relevant financial or non-financial interests to disclose.

Declarations

Conflict of interest The authors declare no competing interests.

References

Ahmad A, El-Shafie A, Fatin Mohd Razali S, Samba Mohamad Z (2014) Reservoir optimization in water resources: a review. Water Resour Manag 28:3391–3405

Alessandri A, Cervellera C, Maccio D, Sanguineti M (2010) Optimization based on quasi-Monte Carlo sampling to design state estimators for non-linear systems. Optimization 59(7):963–984

Almubaidin MAA, Ahmed AN, Sidek LBM, Elshafie A (2022) Using metaheuristics algorithms (MHAs) to optimize water supply operation in reservoirs: a review. Arch Comput Methods Eng 29(6):3677–3711

Azad AS, Rahaman MSA, Watada J, Vasant P, Vintaned JAG (2020) Optimization of the hydropower energy generation using meta-heuristic approaches: a review. Energy Rep 6:2230–2248

Beiranvand B, Ashofteh P-S (2023) A systematic review of optimization of dams reservoir operation using the meta-heuristic algorithms. Water Resources Manag 1–70

Bellman R (1958) Dynamic programming. Princeton University Press, Princeton

Bomze IM, Eichfelder G (2013) Copositivity detection by difference-of-convex decomposition and ω -subdivision. Math Program 138(1):365–400

Carpentier P-L, Gendreau M, Bastin F (2013) Long-term management of a hydroelectric multireservoir system under uncertainty using the progressive hedging algorithm. Water Resour Res 49(5):2812–2827

Carpentier P-L, Gendreau M, Bastin F (2014) Managing hydroelectric reservoirs over an extended horizon using benders decomposition with a memory loss assumption. IEEE Trans Power Syst 30(2):563–572

Cerisola S, Latorre JM, Ramos A (2012) Stochastic dual dynamic programming applied to nonconvex hydrothermal models. Eur J Oper Res 218(3):687–697

Cervellera C, Muselli M (2007) Efficient sampling in approximate dynamic programming algorithms. Comput Optim Appl 38(3):417–443

Cervellera C, Gaggero M, Macciò D, Marcialis R (2013) Quasi-random sampling for approximate dynamic programming. In: The 2013 international joint conference on neural networks (IJCNN). IEEE, pp 1–8

Cervellera C, Gaggero M, Macciò D (2017) Lattice point sets for state sampling in approximate dynamic programming. Optimal Control Appl Methods 38(6):1193–1207

Chen VCP (1999) Application of orthogonal arrays and mars to inventory forecasting stochastic dynamic programs. Comput Stat Data Anal 30(3):317–341

Chen VCP, Ruppert D, Shoemaker CA (1999) Applying experimental design and regression splines to high-dimensional continuous-state stochastic dynamic programming. Oper Res 47(1):38–53

Chen Y, Liu F, Rosenberger JM, Chen VCP, Kulvanitchaiyanunt A, Zhou Y (2020) Efficient approximate dynamic programming based on design and analysis of computer experiments for infinite-horizon optimization. Comput Oper Res 124:105032

Chow VT, Cortes-Rivera G (1974) Application of DDDP in water resources planning. Technical report, University of Illinois at Urbana-Champaign. Water Resources Center

Côté P, Arsenault R (2019) Efficient implementation of sampling stochastic dynamic programming algorithm for multireservoir management in the hydropower sector. J Water Resour Plan Manag 145(4):05019005

Davies S (1997) Multidimensional triangulation and interpolation for reinforcement learning. In: Advances in neural information processing systems, pp 1005–1011

De Matos VL, Philpott AB, Finardi EC (2015) Improving the performance of stochastic dual dynamic programming. J Comput Appl Math 290:196–208

Dias Bruno H, Marcato Andre LM, Souza Reinaldo C, Soares Murilo P, Silva Ivo C Junior, de Oliveira Edimar J, Brandi Rafael, Ramos Tales P, et al. (2010) Stochastic dynamic programming applied to hydrothermal power systems operation planning based on the convex hull algorithm. Math Probl Eng 2010

Dias BH, Tomim MA, Marcato ALM, Ramos TP, Brandi RBS, da Silva ICJ, Filho JAP (2013) Parallel computing applied to the stochastic dynamic programming for long term operation planning of hydrothermal power systems. Eur J Oper Res 229(1):212–222

Diniz AL, Maceira MEP (2008) A four-dimensional model of hydro generation for the short-term hydrothermal dispatch problem considering head and spillage effects. IEEE Trans Power Syst 23(3):1298–1308

Dobson B, Wagener T, Pianosi F (2019) An argument-driven classification and comparison of reservoir operation optimization methods. Adv Water Resour 128:74–86

Dupačová J, Gröwe-Kuska N, Römisch W (2003) Scenario reduction in stochastic programming. Math Program 95:493–511

Feng Z, Niu W, Cheng C, Liao S (2017) Hydropower system operation optimization by discrete differential dynamic programming based on orthogonal experiment design. Energy 126:720–732

Feng Z, Niu W, Jiang Z, Qin H, Song Z (2020) Monthly operation optimization of cascade hydropower reservoirs with dynamic programming and Latin hypercube sampling for dimensionality reduction. Water Resources Manag 34(6)

Gibbons JD (1974) Estimation of the unknown upper limit of a uniform distribution. Sankhya Indian J Stat Ser B (1960–2002) 36(1):29–40

Gimbutas A, Žilinskas A (2018) An algorithm of simplicial Lipschitz optimization with the bi-criteria selection of simplices for the bi-section. J Global Optim 71(1):115–127

Gonçalves REC, Finardi EC, da Silva EL (2012) Applying different decomposition schemes using the progressive hedging algorithm to the operation planning problem of a hydrothermal system. Electric Power Syst Res 83(1):19–27

Goor Q, Kelman R, Tilmant A (2011) Optimal multipurpose-multireservoir operation model with variable productivity of hydropower plants. J Water Resour Plan Manag 137(3):258–267

Habets LCGJM, Collins PJ, van Schuppen JH (2006) Reachability and control synthesis for piecewise-affine hybrid systems on simplices. IEEE Trans Autom Control 51(6):938–948

Mello TH, De Matos VL, Finardi EC (2011) Sampling strategies and stopping criteria for stochastic dual dynamic programming: a case study in long-term hydrothermal scheduling. Energy Syst 2(1):1–31

Horst R (1976) An algorithm for nonconvex programming problems. Math Program 10(1):312–321

Johnson SA, Stedinger JR, Shoemaker CA, Li Y, Tejada-Guibert JA (1993) Numerical solution of continuous-state dynamic programs using linear and spline interpolation. Oper Res 41(3):484–500

Kachiashvili KJ, Topchishvili AL (2016) Parameters estimators of irregular right-angled triangular distribution. Model Assist Stat Appl 11(2):179–184

Labadie JW (2004) Optimal operation of multireservoir systems: State-of-the-art review. J Water Resour Plan Manag 130(2):93–111

Lamond BF, Zéphyr L (2021) Note on "Parameters estimators of irregular right-angled triangular distribution". Model Assisted Stat Appl 16(4) (to appear)

Moore DW (1992) Simplical mesh generation with applications. Technical report, Cornell University

Moravej M, Hosseini-Moghari S-M (2016) Large scale reservoirs system operation optimization: the interior search algorithm (ISA) approach. Water Resour Manag 30:3389–3407

Morillo JL, Pérez JF, Zéphyr L, Anderson CL, Cadena A (2017) Assessing the impact of wind variability on the long-term operation of a hydro-dominated system. In: 2017 IEEE PES innovative smart grid technologies conference Europe (ISGT-Europe). IEEE, pp 1–6

Morillo JL, Zéphyr L, Pérez JF, Lindsay Anderson C, Cadena Á (2020) Risk-averse stochastic dual dynamic programming approach for the operation of a hydro-dominated power system in the presence of wind uncertainty. Int J Electr Power Energy Syst 115:105469

Munos R, Moore A (2002) Variable resolution discretization in optimal control. Mach Learn 49(2–3):291–323

Murray DM, Yakowitz SJ (1979) Constrained differential dynamic programming and its application to multireservoir control. Water Resour Res 15(5):1017–1027

Lin NM, Tian X, Rutten M, Abraham E, Maestre JM, van de Giesen N (2020) Multi-objective model predictive control for real-time operation of a multi-reservoir system. Water 12(7):1898

Nolde K, Uhr M, Morari M (2008) Medium term scheduling of a hydro-thermal system using stochastic model predictive control. Automatica 44(6):1585–1594

Paulavičius R, Žilinskas J (2009) Global optimization using the branch-and-bound algorithm with a combination of Lipschitz bounds over simplices. Technol Econ Dev Econ 15(2):310–325

Paulavičius R, Žilinskas J (2014) Simplicial global optimization. Springer, Berlin

Pereira MVF, Pinto LMVG (1991) Multi-stage stochastic optimization applied to energy planning. Math Program 52(1):359–375

Pereira MVF, Pinto LMVG (1985) Stochastic optimization of a multireservoir hydroelectric system: a decomposition approach. Water Resour Res 21(6):779–792

Rani D, Moreira MM (2010) Simulation-optimization modeling: a survey and potential application in reservoir systems operation. Water Resour Manag 24:1107–1138

Raso L, Malaterre PO (2017) Combining short-term and long-term reservoir operation using infinite horizon model predictive control. J Irrig Drain Eng 143(3):B4016002

Rebennack S (2016) Combining sampling-based and scenario-based nested benders decomposition methods: application to stochastic dual dynamic programming. Math Program 156:343–389

Ruszczyński A, Shapiro A (2003) Stochastic programming models. Handbooks Oper Res Management Sci 10:1–64

Sala A, Armesto L (2022) Adaptive polyhedral meshing for approximate dynamic programming in control. Eng Appl Artif Intell 107:104515

Shapiro A, Dentcheva D, Ruszczynski A (2009) Lectures on stochastic programming: modeling and theory. SIAM, Philadelphia

Tuy H (1991) Effect of the subdivision strategy on convergence and efficiency of some global optimization algorithms. J Global Optim 1(1):23–36

Uysal G, Schwanenberg D, Alvarado-Montero R, Şensoy A (2018) Short term optimal operation of water supply reservoir under flood control stress using model predictive control. Water Resour Manag 32:583–597

van Ackooij W, Henrion R, Möller A, Zorgati R (2014) Joint chance constrained programming for hydro reservoir management. Optim Eng 15(2):509–531

Xu B, Zhong P-A, Zambon RC, Zhao Y, Yeh WW-G (2015) Scenario tree reduction in stochastic programming with recourse for hydropower operations. Water Resour Res 51(8):6359–6380

Yershov DS, LaValle SM (2012) Simplicial Dijkstra and A* algorithms: From graphs to continuous spaces. Adv Robot 26(17):2065–2085

Zéphyr L, Anderson CL (2018) Stochastic dynamic programming approach to managing power system uncertainty with distributed storage. CMS 15(1):87–110

Zéphyr L, Lang P, Lamond BF (2014) Adaptive monitoring of the progressive hedging penalty for reservoir systems management. Energy Syst 5(2):307–322

Zéphyr L, Lang P, Lamond BF (2015) Controlled approximation of the value function in stochastic dynamic programming for multi-reservoir systems. CMS 12(4):539–557

Zéphyr L, Lang P, Lamond BF, Côté P (2017) Approximate stochastic dynamic programming for hydroelectric production planning. Eur J Oper Res 262(2):586–601

Žilinskas A, Žilinskas J (2002) Global optimization based on a statistical model and simplicial partitioning. Comput Math Appl 44(7):957–967