Machine learning approach to chance-constrained problems: An algorithm based on the stochastic gradient descent

Lukáš Adam*[1,2] and Martin Branda[2,3]

[1]Southern University of Science and Technology, Shenzhen 518055, China
[2]ÚTIA, The Czech Academy of Sciences, Pod Vodárenskou věží 4, 18208, Prague, Czech Republic
[3]Faculty of Mathematics and Physics, Charles University, Sokolovská 83, 18675, Prague, Czech Republic

December 11, 2018

Abstract

We consider chance-constrained problems with discrete random distribution. We aim for problems with a large number of scenarios. We propose a novel method based on the stochastic gradient descent method which performs updates of the decision variable based only on looking at a few scenarios. We modify it to handle the non-separable objective. A complexity analysis and a comparison with the standard (batch) gradient descent method is provided. We give three examples with non-convex data and show that our method provides a good solution fast even when the number of scenarios is large.

Keywords: Stochastic programming, Chance-constrained programming, Quantile, Stochastic gradient descent, Machine learning, Large-scale
AMS classification: 90C15, 90C26, 49M05.

1 Intruduction

In real-world problems, the data is often stochastic (random). Some examples include uncertain parameters, imprecise measurements or unknown future prices [25]. Since deterministic models do not reflect this fact, they may provide a subpar solution. Stochastic models may provide an alternative. However, changing some variables from deterministic to stochastic brings several issues. The first one is the increased complexity. The second one is the question of how to handle the stochastic variables. As it often provides too conservative results, we do not use the robust optimization approach [3] and focus instead on the stochastic optimization approach [4]. There, the usual approach is to replace the stochastic objective by its expectation and impose probability on fulfilling the random constraints leading to chance constraints. The expectation describes the average behaviour while the chance constraints specify that the stochastic constraints have to be satisfied with a large probability.

First, we review recent contributions to nonlinear chance-constrained problems. [23] derived a formula representing the gradients of nonlinear chance constraints in the Gaussian and Student case as a certain integral over the sphere. [9] proposed an algorithm based on solving inner and outer approximations of the chance constrained problems. The approximations consist of two parametric nonlinear programming problems. Asymptotic convergence to an optimal solution is shown. [10] used an approach called spheric-radial decomposition of multivariate Gaussian distributions to solve a demanding problem of gas network design with uncertain demand.All these methods make use of a continuous distribution of the random vector.

*adam@utia.cas.cz

However, often only an approximation of the true underlying distribution via a finite number of samples is known. In such a case, recent results on nonlinear chance-constrained problems include: [20] generalized the approach based on the difference of two convex functions by proposing new smooth approximating functions and showed a convergence to a stationary point. [1, 2] derived strong and weak necessary optimality conditions. Based on a regularization technique, they proposed an algorithm converging to a stationary point. [22] generalized the Benders' decomposition approach for minimizing convex nondifferentiable functions over a combinatorial set. [27] derived new quantile cuts to strengthen the cutting plane approach for solving mixed-integer nonlinear chance-constrained problems. Since the generation of all cuts is difficult, the authors proposed a practical heuristic approach. [7] introduced a sequential algorithm for solving nonlinear chance constrained problems. The method is based on an exact penalty function which is minimized sequentially by solving quadratic optimization subproblems with linear cardinality constraints.

To the best of our knowledge, all these algorithms either consider a specific continuous distribution or are suitable for a distribution with only a small number of scenarios. Since the number of scenarios should theoretically increase exponentially with the dimension of the random vector, a method which is able to handle a large number of scenarios is needed.

To derive such a method, we seek an inspiration in the machine learning method called the stochastic gradient descent [5]. It is able to train deep neural nets with billions of samples and millions of decision variables [15]. Its main idea is to use the standard (batch) gradient descent method but instead of computing the gradient on the whole dataset, it computes it only for a small number of samples called the minibatch. The stochastic gradient descent has a direct connection to the coordinate descent method [21] where the gradient descent is computed with respect to a few decision variables instead of a few samples. There are several advantages of the stochastic gradient descent over its batch variant:

1. The gradient computation is much faster and has much lower memory requirements. At the same time, it should not need significantly more iterations to approach the solution because the gradient with respect to any one given sample usually points towards the minimum during the early iterations.
2. Datasets often contain duplicate information. Since the batch gradient descent computes the gradients with respect to all samples, unnecessary computation is performed.
3. Due to the noisy gradients, stochastic gradient descent has a higher chance to escape local minima and stationary points for nonconvex problems. This also makes it easier to handle nonsmooth functions.
4. Due to the noisy gradients, stochastic gradient descent does not overfit to the training data and generalizes better to unseen samples [18].

All these advantages are closely related to chance-constrained problems. The first two points are general and refer to a lower computational effort. Chance-constrained problems are nonconvex even for linear data and when a joint chance constraint is converted into an individual one via the max operator, the constraint is also nonsmooth. Thus, the third point applies. The last point is relevant because for a large dimension of the random vector, sampling provides only a rather crude approximation of the true distribution and overfitting to the training data may decrease the solution quality.

The biggest disadvantage of the stochastic gradient descent is that it often only approaches the solution and then oscillates around it. A brilliant overview of the convergence of the stochastic gradient descent method is provided in [5]. However, this work did not consider any constraints. This was partially handled in [24].

Our paper is organized as follows. In Section 2, we provide a brief introduction into the chance-constrained problems and into the stochastic gradient descent method. The stochastic gradient method requires that the objective is separable with respect to samples. However, the chance constraint combines all samples together. Thus, in Section 3 we provide an algorithm which handles this obstruction and we provide a computational complexity and comparison to the batch gradient descent in Section 4. Finally, in Section 5 we provide a brief description of three testing problems and show a good performance of our method in Section 6. We stress that we aimed at nonconvex problems with a rather large number of scenarios.

2 Preliminaries

In this section, we introduce the chance-constrained problems and the stochastic gradient descent.

2.1 Chance-constrained problems

Chance-constrained problems are specific types of optimization problems where some constraints have to be satisfied only "sufficiently" often and may be violated in some cases. More formally, for a random vector ξ, we require that the random constraint $g(x,\xi) \leq 0$ is satisfied with probability at least $1-\varepsilon$ for some small ε. With an objective function f and a deterministic constraint set X, the chance-constrained problem may be written as

$$\begin{aligned} &\text{minimize } \mathbb{E} f(x,\xi) \\ &\text{subject to } \mathbb{P}(g(x,\xi) \leq 0) \geq 1-\varepsilon, \\ &\qquad x \in X. \end{aligned} \tag{1}$$

Here, $\mathbb{E}$ refers to the expectation and $\mathbb{P}$ to the probability. The chance constraint states that the $(1-\varepsilon)$-quantile of $g(x,\cdot)$ is at most 0. Defining the $(1-\varepsilon)$-quantile function formally by

$$q(x) := \min\{t |\ \mathbb{P}(g(x,\xi) \leq t) \geq 1-\varepsilon\}, \tag{2}$$

then the chance constraint is equivalent to $q(x) \leq 0$. Then problem (1) amounts to

$$\begin{aligned} &\text{minimize } \mathbb{E} f(x,\xi) \\ &\text{subject to } q(x) \leq 0, \\ &\qquad x \in X. \end{aligned} \tag{3}$$

In Appendix A we show that under mild conditions q is a Lipschitz continuous function, which allows us to solve this problem by the (stochastic) (sub)gradient descent. To this aim, the constraints have to be in a simple form so that we can compute the projection fast. Thus, we penalize the constraint on the quantile to obtain

$$\begin{aligned} &\text{minimize } \mathbb{E} f(x,\xi) + \lambda \phi(q(x)) \\ &\text{subject to } x \in X. \end{aligned} \tag{4}$$

where $\lambda > 0$ is the penalization parameter and $\phi(z) = \max\{z,0\}^2$ is a penalty function. Then we solve (4) by the projected stochastic gradient descent described in the next section.

2.2 Stochastic gradient descent

In machine learning, the typical optimization problem takes form

$$\text{minimize } \frac{1}{S}\sum_{i=1}^{S} h(x,\xi_i). \tag{5}$$

Here, x is a decision variable, h is a loss function (usually a discrepancy between predictions and labels) and ξ_i denotes individual samples. In stochastic optimization, ξ_i denotes scenarios (realizations of a random vector) and the goal is to minimize the expectation of h.

The simplest approach to solve (5) to apply the (batch) gradient descent, where at iteration k we set

$$x^{k+1} := x^k - \alpha^k \frac{1}{S}\sum_{i=1}^{S} \nabla_x h(x,\xi_i), \tag{6}$$

where $\alpha^k > 0$ is the stepsize. Since the number of samples S is often large, computing the full gradient in (6) is time-consuming. The usual strategy is to replace the batch gradient by a stochastic gradient, where we select a subset I^k of $\{1,\dots,S\}$ and perform the update as

$$x^{k+1} := x^k - \alpha^k \frac{1}{|I^k|}\sum_{i\in I^k} \nabla_x h(x,\xi_i). \tag{7}$$

The name stochastic gradient descent reflects the fact that the gradient update is performed only with respect to a stochastic subset of observations.

3 Solving chance-constrained problems via stochastic gradient descent

In this section, we propose a novel scalable method for solving the chance-constrained problem

$$\begin{aligned} &\text{minimize } \mathbb{E} f(x,\xi) \\ &\text{subject to } \mathbb{P}(g(x,\xi) \leq 0) \geq 1-\varepsilon, \\ &\qquad x \in X. \end{aligned} \tag{8}$$

We recall that we minimize the expectation of $f(x,\xi)$ while we prescribe the probability that the random constraint $g(x,\xi) \leq 0$ is satisfied. Using the quantile functon q defined in (2), we may equivalently rewrite the chance contraint into $q(x) \leq 0$. When this constraint is penalized a penalization parameter λ and a penalization function ϕ, we arrive at (4). If ξ has a finite number of scenarios $\{\xi_1, \dots, \xi_S\}$, then this problem amounts to

$$\begin{aligned} &\text{minimize } \frac{1}{S}\sum_{i=1}^{S} f(x,\xi_i) + \lambda \phi(q(x)) \\ &\text{subject to } x \in X. \end{aligned} \tag{9}$$

Note that when we drive λ to infinity, the solutions of (9) will converge (under a constraint qualification) to a solution of (8). Thus, we concentrate on solving (9).

We intend to apply the stochastic gradient descent described in Section 2.2. To this aim, the projection onto X has to be simple and the objective function has to be separable as in (5). This is true for the first part of our objective (9). However, the quantile function q in the second part of (9) combines all scenarios and thus, it is not separable. In this part, we will extend the stochastic gradient descent to handle this obstruction.

3.1 How to compute derivatives?

First, we will compute the derivative for q. Since there are finite number of scenarios, the quantile is realized at some scenario, see Appendix A. Formally, for every x, there exists some index $i(x)$ such that

$$q(x) = g(x, \xi_{i(x)}). \tag{10}$$

If this index is unique, then q is differentiable at x and the derivative equals to

$$\nabla q(x) = \nabla_x g(x, \xi_{i(x)}).$$

Then the derivative of the objective function (9) can be computed via the chain rule and equals to

$$\frac{1}{S}\sum_{i=1}^{S} \nabla_x f(x,\xi_i) + \lambda \phi'(q(x)) \nabla_x g(x, \xi_{i(x)}). \tag{11}$$

Note that the last part of the gradient depends only on one scenario $i(x)$. Having the gradient at hand, it is simple to write the gradient descent update

$$\begin{aligned} y^{k+1} &:= x^k - \alpha^k \left(\frac{1}{S}\sum_{i=1}^{S} \nabla_x f(x^k,\xi_i) + \lambda \phi'(q(x^k)) \nabla_x g(x^k, \xi_{i(x)}) \right), \\ x^{k+1} &:= P_X(y^{k+1}), \end{aligned} \tag{12}$$

where α^k is the stepsize and P_X is the projection onto the feasible set X and k denotes the iteration index. We summarize the (batch) gradient descent in Algorithm 3.1.

Unfortunately, update (12) requires to compute the quantile $q(x^k)$, which in turn requires the evaluation of $g(x,\xi_i)$ for all scenarios $i \in \{1,\dots,S\}$. Since this is a costly update, we will suggest a new update which

Algorithm (Auxiliary) 3.1 Batch gradient descent for solving (9)

1: Set index $k \leftarrow 0$
2: Initialize variable x^0
3: **while not** termination criterion **do**
4: Compute $g_i^k \leftarrow g(x^k, \xi_i)$ for all $i \in \{1, \dots, S\}$ ▷ Update g
5: Find quantile q^k and the index $i(x^k)$ from (10) realizing the quantile ▷ Find quantile
6: Update x^k according to (12) ▷ Update x
7: Increase k by one
8: **end while**

evaluates g only on a (small) number of samples I^k called the minibatch. The main idea is to use auxiliary variables z_i^k which approximate $g(x^k, \xi_i)$ and to consecutively update this approximation by setting

$$z_i^k := \begin{cases} g(x^k, \xi_i) & \text{if } i \in I^k, \\ z_i^{k-1} & \text{otherwise.} \end{cases} \tag{13}$$

Since k is the iteration index, z_i^k contains the evaluation of $g(\cdot, \xi_i)$ for some (possibly delayed) value of x.

Then we compute the approximation q^k of the quantile $q(x^k)$ defined in (2) as a quantile of $\{z_i^k\}$, which amounts to solving

$$q^k := \min\left\{ t \mid \frac{1}{S} \sum_{i=1}^{S} \chi(z_i^k \leq t) \geq 1 - \varepsilon \right\}. \tag{14}$$

Here, χ is the characteristic (0-1) function checking if $z_i^k \leq t$ is satisfied. Similarly to (10), there is some i^k such that

$$q^k = z_{i^k}^k. \tag{15}$$

Then we can approximate the gradient in (11) by

$$\frac{1}{|I^k|} \sum_{i \in I^k} \nabla_x f(x^k, \xi_i) + \lambda \phi'(q^k) \nabla_x g(x^k, \xi_{i^k}). \tag{16}$$

and the next iterate in (12) by

$$\begin{aligned} y^{k+1} &:= x^k - \alpha^k \left(\frac{1}{|I^k|} \sum_{i \in I^k} \nabla_x f(x^k, \xi_i) + \lambda \phi'(q^k) \nabla_x g(x^k, \xi_{i^k}) \right), \\ x^{k+1} &:= P_X(y^{k+1}). \end{aligned} \tag{17}$$

There are three differences between (12) and (17). First, the expectation of the gradient of f with respect to all samples is replaced by its expectation with respect to I^k. Second, the derivative of the penalization function at the exact quantile $\phi'(q(x^k))$ is replaced by its derivative at the approximative quantile $\phi'(q^k)$. Third, index $i(x^k)$ satisfying (10) is replaced by i^k satisfying (15). Naturally, this makes the update (17) inexact. However, its main strength lies in the fact that g is evaluated only at the active minibatch I^k in (13) and at one index i^k in (16). Thus, g is computed only at a small number of indices and thus update (17) is much faster than (12), especially if the computation of g is difficult.

3.2 How to efficiently update the quantile?

The simplest way to compute the quantile q^k in (14) is to sort the samples z_i^k and select the index $\lceil S(1-\varepsilon) \rceil$. This can be done in $O(S \log S)$. Even though there are specialized algorithms [14] which are able to find the quantile in $O(S)$, they do not utilize the fact that the minibatch has a much smaller size than the number of samples S. For this reason, we designed a specialized sorting algorithm.

Its basic idea is to keep z^k sorted at every iteration. Due to (13), z^{k+1} differs from z^k only on the minibatch I^{k+1}. Since z^k is already sorted, it suffices to sort the new values $g(x^{k+1}, \xi_i)$ on I^{k+1} and then

merge these two sorted arrays. This can be done in one pass through both arrays. Then we obtain the sorted version of z^{k+1} and the quantile equals to index $\lceil S(1-\varepsilon)\rceil$ of this sorted array. Since the details are technical, we formalize this idea in Appendix B.

3.3 Numerical implementation

In this section, we combine the derivative computation from Section 3.1 with the quantile computation from Section 3.2 into Algorithm 3.2 which solves (9). Further, in Algorithm 3.3 we provide a simple wrapper for solving the original problem (8).

We will comment now on Algorithm 3.2. As we have mentioned earlier, this algorithm is based on keeping the values z^k sorted. This is done by a permutation π of $\{1,\dots,S\}$ such that $s_i^k = z_{\pi(i)}^k$ and s^k is sorted. For more details see (29) in Appendix B. In initialization steps 1-4 we choose an initial point x^1, compute z^0 with components $z_i^0 = g(x^1,\xi_i)$ and sort this array into s^0 with the corresponding permutation π^0 satisfying (29). Steps 6-7 form a reshuffling procedure which ensures that minibatches are selected randomly. Note that after reshuffling, we still have (29) and $I^k = \{i_{\text{low}},\dots,i_{\text{high}}\}$ is now a random minibatch. If the minibatch contains $\text{size}_{\text{minibatch}} = i_{\text{high}} - i_{\text{low}} + 1$ scenarios, then $\{1,\dots,S\}$ can be split into

$$n_{\text{minibatch}} := \left\lfloor \frac{S}{\text{size}_{\text{minibatch}}} \right\rfloor \tag{18}$$

minibatches. In step 8 we perform a loop over these minibatches. In step 9 we define the minibatch I^k and we update $g(x^k,\xi_i)$ on this minibatch in step 10. Since Algorithm B.1 requires these new values to be sorted, we perform it in step 11 and then run Algorithm B.1 in step 12 to compute the quantile q^k. Before updating x^k in step 14, we need to find the index i^k of the quantile in step 13. Finally, we increase k and reiterate.

Algorithm (Auxiliary) 3.2 Stochastic gradient descent for solving (9)

1: Initialize variable x^1
2: Set $z_i^0 = g(x^1,\xi_i)$ for all $i = 1,\dots,S$
3: Sort z^0 to obtain s^0 and find the sorting permutation π^0 satisfying (29)
4: Set index $k \leftarrow 1$
5: **while not** termination criterion **do** ▷ Epoch index
6: Find a random permutation θ of $\{1,\dots,S\}$ ▷ Randomly shuffle scenarios
7: Replace ξ_i by $\xi_{\theta(i)}$, replace z_i^{k-1} by $z_{\theta(i)}^{k-1}$ and replace $\pi^{k-1}(i)$ by $\theta^{-1}(\pi^{k-1}(i))$
8: **for** $j = 1,\dots,n_{\text{minibatch}}$ **do** ▷ Loop within an epoch
9: $I^k \leftarrow [i_{\text{low}}, i_{\text{high}}] = [(j-1)\text{size}_{\text{minibatch}} + 1,\ i_{\text{high}} \leftarrow j\text{size}_{\text{minibatch}}]$ ▷ Determine minibatch
10: Compute $g_i^k \leftarrow g(x^k,\xi_i)$ for all $i \in I^k$ ▷ Update g on minibatch
11: Sort g^k and find ϕ satisfying (29) ▷ Sort g on minibatch
12: Apply Algorithm B.1 to get s^k, π^k and q^k ▷ Insert g and find quantile
13: Get index $i^k \leftarrow \pi^k(\lceil S(1-\varepsilon)\rceil)$ ▷ Get index realizing quantile
14: Update x^k according to (17) ▷ Update x
15: Increase k by one
16: **end for**
17: **end while**

The procedure described in Algorithm 3.2 solves problem (9) for one fixed λ. When we are satisfied with the current solution, we increase λ and use the terminal value from the previous λ as the starting value for the next λ. This provides a solution to the chance-constrained problem (8) and the procedure is summarized in Algorithm 3.3.

4 Complexity analysis and comparison with the batch algorithm

In this section, we provide a complexity analysis of Algorithm 3.2 and we show the benefits of our algorithm over the standard batch gradient from Algorithm 3.1. In machine learning, the crucial term is the epoch. We provide the definition now.

Algorithm 3.3 For solving the chance-constrained problem (8)

1: **for** $\lambda^1 < \lambda^2 < \cdots < \lambda^L$ **do**
2: Employ Algorithm 3.2 with starting value x^{l-1} and $\lambda = \lambda^l$ to get x^l
3: **end for**

Definition 4.1. One epoch is the time when g is evaluated S times.

In other words, the epoch is the time during which g looks at the whole dataset and evaluates each scenario for some x. Since the evaluation of g is often the most demanding computation, epoch acts as an indicator for the computational time.

In the batch algorithm from Algorithm 3.1, one epoch equals to one `while` loop in step 3. Denoting g the complexity of computing $g(\cdot, \xi_i)$ for one sample i, then the complexity of step 4 is $O(gS)$. For step 5 there are algorithms [14] for finding the quantile with complexity $O(S)$. In total, the complexity is $O(gS)$ and during one epoch we will perform one update of x.

In the stochastic gradient descent from Algorithm 3.2, the epoch consists of one `while` loop in step 5. The reordering in steps 6-7 can be performed for example with the Fisher-Yates shuffle [14] with complexity $O(S)$. Computing $g(x^k, \xi_i)$ on the minibatch I^k in step 10 has complexity $O(g\text{size}_{\text{minibatch}})$ and sorting it in step 11 can be done with the quicksort [14] with complexity $O(\text{size}_{\text{minibatch}} \log(\text{size}_{\text{minibatch}}))$. Step 12 requires to apply the sorting procedure from Algorithm B.1 which amounts to one pass through the sorted array and thus has complexity $O(S)$. All other steps are negligible. Thus, the complexity of the inner loop in step 8 amounts to

$$O(g\text{size}_{\text{minibatch}} + \text{size}_{\text{minibatch}} \log(\text{size}_{\text{minibatch}}) + S)$$

Since for one epoch we perform this update $n_{\text{minibatch}}$ times, due to (18) the complexity of one epoch equals to

$$O(gS + S\log(\text{size}_{\text{minibatch}}) + n_{\text{minibatch}}S).$$

During one epoch, $n_{\text{minibatch}}$ updates of x are performed.

Table 1: Comparison of the complexity of the batch and stochastic gradient descent methods for solving the (penalization of) chance-constrained problem (9).

	Complexity	Updates of x
Batch gradient descent	$O(gS)$	1
Stochastic gradient descent	$O(gS + S\log(\text{size}_{\text{minibatch}}) + n_{\text{minibatch}}S)$	$n_{\text{minibatch}}$
Batch gradient descent	$O\big(g\sqrt{S}S\big)$	$\sqrt{S}$
Stochastic gradient descent	$O\big((g+\sqrt{S})S\big)$	$\sqrt{S}$

We summarize the complexity in Table 1. The first two rows follow directly from the discussion above. For a simple comparison, we choose $\text{size}_{\text{minibatch}} = n_{\text{minibatch}} = \sqrt{S}$, for which the complexity for $\sqrt{S}$ epochs reduces to $O(g\sqrt{S}S)$ and $O((g+\sqrt{S})S)$, respectively. Thus, we see that the stochastic gradient descent provides a clear benefit whenever the computation of g is time-consuming. Moreover, the O-constants from the stochastic gradient descent Algorithm 3.2 are tight. Quicksort is known to have a very tight constant and the sorting procedure in Algorithm B.1 requires only one pass through the array.

5 Applications: Description

In this section, we describe three applications to test the performance of our Algorithm 3.3. The numerical results are postponed to Section 6.

5.1 Application 1: Optimal control of fish population

This application is adapted from [16, Chapter 8], where the authors provided the optimal fishing strategy for a deterministic fishery model without a terminal condition. The model assumes the logistic population

evolution

$$\dot{x} = rx\Big(1 - \frac{x}{K}\Big),$$

where $x(t)$ is the number of fish, r is the growth rate, K is the carrying capacity and the initial condition $x(0) = x_0$ is satisfied. We introduce control variable $u(t)$ which measures the fishing rate. Then model changes into

$$\dot{x} = rx\Big(1 - \frac{x}{K}\Big) - ux.$$

Discretization this ODE via the forward Euler scheme with time step Δt leads to

$$x_{t+1} = x_t + \Delta t\left(rx_t - \frac{1}{K}rx_t^2 - u_t x_t\right). \tag{19}$$

The profit from fishing may be written as

$$\Delta t \sum_{t=0}^{T-1} \left(p_t u_t x_t - d_t u_t^2 x_t^2 - c_t u_t\right)$$

where Δt is the time step, p is the fish price, d the cost of diminishing returns and c the fishing costs. Putting this all together, we obtain an optimization problem

$$\begin{aligned} &\text{maximize } \Delta t \sum_{t=0}^{T-1} \left(p_t u_t x_t - d_t u_t^2 x_t^2 - c_t u_t\right) \\ &\text{subject to system (19) holds true,} \\ &\qquad u(t) \in [0, u_{\max}], \end{aligned} \tag{20}$$

where $u_{\max}$ is the maximal fishing rate.

The solution of this problem will likely result in an empty fishery, which is not optimal for a long-term development. Thus, we add the constraint $x_T \geq x_{\text{des}}$. We do not need to consider the non-negativity constraint $x_t \geq 0$. Indeed, if x is negative, then the solution may be improved by considering $u = 0$. Since the carrying capacity K, the growth rate r, the initial state x_0 and the time-dependent future prices p, d, c are not known precisely, it makes sense to consider them as stochastic variables. Then x becomes a stochastic variable as well and the deterministic constraint $x_T \geq x_{\text{des}}$ needs to be changed into a stochastic one. This leads to the problem

$$\begin{aligned} &\text{maximize } \Delta t \sum_{t=0}^{T-1} \mathbb{E}\left(p_t u_t x_t - d_t u_t^2 x_t^2 - c_t u_t\right) \\ &\text{subject to system (19) holds true,} \\ &\qquad \mathbb{P}(x_T \geq x_{\text{des}}) \geq 1 - \varepsilon, \\ &\qquad u_t \in [0, u_{\max}]. \end{aligned} \tag{21}$$

For every u, we are able to compute x in a unique manner. Thus, the decision variable is only u and $x(u)$ can be considered as the state variable. Then, problem (21) fits into setting (8) and we may apply Algorithm 3.3 to solve it. The derivates are computed via the backpropagation technique described in Appendix D. We would like to note that since x_t does not depend on p_t, these random variables are independent. Thus, we have $\mathbb{E}p_t u_t x_t = u_t \mathbb{E}p_t \mathbb{E}x_t$ and the prices p_t, d_t and c_t may be replaced by their expectations. Here, we keep the original more complex problem (21) to simulate the situation where our algorithm is applied in a brute force manner. We comment more on this in Appendix E.

5.2 Application 2: Optimal control of electrostatic separator

Recently, a great emphasis has been put on the circular economy where resources are reused instead of being discarded. One of the major problems is the abundance of plastics, for example, the packaging is often discarded immediately after being used [19]. Since there are multiple types of plastics, the plastic

waste usually does not contain only one type but their mixture. Since each plastics has a different recycling procedure, a necessary preliminary step before recycling is their separation [8].

One of the separation possibilities is the electrostatic free-fall separator [26] depicted in Figure 3. Two types of particles are charged with opposite polarities, placed into the feeder and then dropped into the separator with an electrostatic field generated by two parallel electrodes. Due to the gravity, the particles fall downwards and due to the electrostatic field and opposite polarities, one of the plastic types falls leftwards while the other one rightwards. Thus, the particles separate.

There were several attempts to optimize the shape of the free-fall separator [17]. Here, we consider the simplified version with parallel electrodes. We consider three forces acting on the particles: the gravitational force $F_g = mg$, the Coulomb force $F_c = \frac{QU}{d_r - d_l}$ and the air drag $F_a = \frac{1}{2} CS\rho c v^2$. Here, m is the mass of the particle, Q its charge, U the voltage, $d_r - d_l$ the distance of the electrodes, C the drag coefficient, S the particle cross-section, ρ the density of air and v the particle speed. Since the electrodes are assumed to be parallel, the Coulomb force F_c reduces to the simple form above. The equations for particle position s and velocity v in components (x, y) equal to

$$\begin{aligned} \dot{s}_x &= v_x, \\ \dot{s}_y &= v_y, \\ \dot{v}_x &= \frac{QU}{m(d_r - d_l)} - \frac{1}{2m} CS\rho v_x \sqrt{v_x^2 + v_y^2}, \\ \dot{v}_y &= g - \frac{1}{2m} CS\rho v_y \sqrt{v_x^2 + v_y^2}. \end{aligned} \tag{22}$$

We add the impact of particles on the electrodes rather informally by writing

$$v_x \text{ changes sign if } s_x = d_l \text{ or } s_x = d_r. \tag{23}$$

Due to this constraint, the system is non-differentiable. As the control variables we consider the voltage U, the position of the left electrode d_l and the position of the right electrode d_r. The random variables include the charge Q, the mass m and the initial position (s_x, s_y) and the initial velocity (v_x, v_y).

Since the cost of the electrostatic separator is proportional to the voltage on the electrodes, we want to minimize it. At the same time, we want to achieve a high-quality separation. Since the positively charged particles are supposed to fly right, we expect $s_{x,\text{pos}} \geq x_{\text{des}}$, where x_{des} is some positive desired state. Similarly, the negatively charged particles are supposed to fly left and thus, we want them to satisfy $s_{x,\text{neg}} \leq -x_{\text{des}}$. This gives rise to the following problem

$$\begin{aligned} &\text{minimize } U \\ &\text{subject to system (22) and (23) holds true,} \\ &\qquad \mathbb{P}(s_{x,\text{pos}}(T) \geq x_{\text{des}},\ s_{x,\text{neg}}(T) \leq -x_{\text{des}}) \geq 1 - \varepsilon, \\ &\qquad U \in [U_{\min}, U_{\max}],\ d_l \in [-d_{\max}, -d_{\min}],\ d_r \in [d_{\min}, d_{\max}]. \end{aligned} \tag{24}$$

Similarly to the previous application, we can consider the decision variables as only U, d_l and d_r while considering s_x, s_y, v_x and v_y as the dependent state variables. The ODE is again discretized via the forward Euler scheme and the derivatives are computed via the backpropagation technique described in Appendix C.

5.3 Application 3: Optimal design of gas network

We follow the gas network described in [11] by an injection node 0, withdrawal nodes $\{1, \ldots, n\}$ and a set of pipes (edges) with their pressure drop coefficients Φ_e. For each node i, there is a stochastic demand ξ_i which is assumed to follow a known distribution. The goal in [2] was to design the network such that the demand is satisfied with a high probability. This was specified by controlling the upper pressure bounds $p_i^{\max}$ while the lower pressure bounds $p_i^{\min}$ were normalized to one.

For tree-structured networks without cycles, the authors in [11] showed that a random demand ξ can be

satisfied if and only if

$$\begin{aligned}(p_0^{\min})^2 &\le (p_i^{\max})^2 + h_i(\xi),\ i = 1,\dots,n,\\ (p_0^{\max})^2 &\ge (p_i^{\min})^2 + h_i(\xi),\ i = 1,\dots,n,\\ (p_i^{\max})^2 + h_i(\xi) &\ge (p_j^{\min})^2 + h_j(\xi),\ i,j = 1,\dots,n.\end{aligned} \tag{25}$$

Here, functions $h_i(\xi)$ can be computed by

$$h_i(\xi) = \sum_{e\in\Pi(i)} \Phi_e\left(\sum_{j\succeq\pi(e)} \xi_j\right)^2,$$

where $\Pi(i)$ denotes the unique directed path (edges) from the root node 0 to node i, $\pi(e)$ is the end node of edge e and $j \succeq i$ means that the unique path from root to j passes through i.

There are many ways of defining the objective. The simplest way is to minimize the total upper pressure bounds, which results in

$$\begin{aligned}&\text{minimize } \sum_{i=1}^{n} p_i^{\max}\\ &\text{subject to } \mathbb{P}(\text{system (25) is fulfilled}) \ge 1-\varepsilon,\\ &\qquad p_i^{\max} \ge 1.\end{aligned} \tag{26}$$

To apply our algorithm, we need to have only one chance contraint. Since (25) contains multiple constraints, we employ the standard trick of passing to the maximum. This gives rise to the combined single constraint

$$\max_{i=1,\dots,n} \left((p_i^{\max})^2 - \tilde{h}_i(\xi)\right) \ge 0,$$

where $\tilde{h}_i(\xi)$ combines the values of $h_j(\xi)$ and $p_j^{\min}$. Note that this is a nonconvex and a nonsmooth constraint. However, as mentioned in the introduction, this should not be a (big) hurdle for our algorithm due to its stochastic nature.

6 Applications: Numerical results

In this section, we apply our Algorithm 3.3 to the chance-constrained problems described in Section 5. We summarize these applications in Table 2. Note that in all cases we considered a rather large number of $S = 100{,}000$ scenarios. Moreover, two of these applications contains an ODE in the constraints, one is nonsmooth and one contains joint chance constraints. In all three applications, the function $g(\cdot,\xi)$ is nonconvex.

Table 2: Summary of the used problems: number of variables n, number of scenarios S, dimension of the random vector ξ, type of chance constraints, type of algebraic constraints and special features.

	Label	n	S	$\dim\xi$	CCP type	Constraints	Speciality
Fishing	(21)	1000	100,000	$3+3n$	Individual	Box	ODE
Separator	(24)	3	100,000	4	Individual	Box	ODEs, Nonsmooth
Gas network	(26)	12	100,000	n	Joint	Box	-

In Table 3 we summarize the used parameters. If multiple values were used, they are separated by a slash. Parameters specific for individual applications are described in sections dedicated to individual applications. We have already mentioned that the criterion for the computational complexity is the number of epochs from Definition 4.1. The standard criterion for the algorithm progress is the number of updates of the decision variable. Since during one epoch we perform $\frac{S}{\text{size}_{\text{minibatch}}}$ updates, this is equal to

$$n_{\text{update}} = S\frac{n_{\text{epoch}}}{\text{size}_{\text{minibatch}}}. \tag{27}$$

Table 3: Summary of the used parameters: allowed failure level ε, size of the minibatch $\text{size}_{\text{minibatch}}$, number of epochs n_{epoch}, the stepsize α and the initial penalization parameter λ^1.

	ε	$\text{size}_{\text{minibatch}}$	n_{epoch}	α	λ^1
Fishing	0.2	1000/100	10/1	10^{-2}	10^{1}
Separator	0.1	1000/100/10	20	10^{-4}	10^{3}
Gas network	0.15	2000/1000/500/250/100	40/20/10/5/2	10^{-4}	10^{-3}

In other words, if S is fixed, the progress of Algorithm 3.3 should be constant (for sufficiently large minibatches) in $\frac{n_{\text{epoch}}}{\text{size}_{\text{minibatch}}}$.

All codes were implemented in Matlab. The only exception was the merging Algorithm B.1 where we spent hours of cursing and thousands of tears before we finally managed to implement it in C.

6.1 Application 1: Optimal control of fish population

For the fishing application, the prices p, d, c follow a time-dependent multivariate random walk. More precise generation process and other random variables are described in Appendix F.1. The time interval was chosen as $[0, 10]$, which we discretized into $n = 100$ and $n = 1000$ time steps. Note that this number also equals to the number of decision variables. We repeated the experiment with $(\text{size}_{\text{minibatch}}, n_{\text{epoch}}) = (100, 1)$ and $(\text{size}_{\text{minibatch}}, n_{\text{epoch}}) = (1000, 10)$. Other parameters were chosen as described in Table 3.

We compare the total time over all values of the penalization parameter λ in Table 4. First, the total time is relatively small, for example for a problem with $n = 1000$ decision variables and $S = 100000$ scenarios, we need only 106 seconds to solve the problem. Note that evaluations of the objective f and the constraints g, which corresponds to solving the ODE, is the most time-consuming computation while the time to find the quantile is small.

Table 4: Summary of the needed time for the fishing application from Section 5.1: Total time, time to evaluate the objective f and the constraints g and the time to find the quantile.

					Time [s]			
n	S	$\text{size}_{\text{minibatch}}$	n_{epoch}	n_{updates}	Total	Eval f	Eval g	Quantile
100	100000	100	1	1000	17.1	4.4	2.0	7.9
100	100000	1000	10	1000	100.2	49.5	24.0	8.0
1000	100000	100	1	1000	106.1	53.5	7.2	7.0
1000	100000	1000	10	1000	2321.6	1439.4	657.9	14.1

According to (27), the parameter choice resulted in constant number of decision variable updates $n_{\text{update}} = 1000$. Thus, we expect the solutions to look similar. This is confirmed in Figure 1. Its left part depicts the fishing rate u for all four parameter settings while the right part depicts the number of fish x for randomly selected scenarios. These solutions look similar and can be simply interpreted: In interval $[0, 2]$ the process is driven to a steady state which is kept with relatively constant fishing and the number of fish in the interval $[2, 7]$. Since the steady state has a smaller number of fish than the desired end-time level $x_{\text{des}} = 1.5$, the fishing rate drops in the interval $[7, 10]$ and the number of fish increases.

Looking at Figure 1 again, we would like to emphasize the biggest advantage of the stochastic descent: Even though for each λ we performed only $n_{\text{epoch}} = 1$ evaluation of $g(\cdot, \xi)$ on the whole dataset, we performed $n_{\text{update}} = 1000$ updates of the decision variable. This resulted in a high-quality solution in only 106 seconds.

6.2 Application 2: Optimal control of electrostatic separator

We recall that the electrostatic separator takes two types of plastic or metal particles, triboelectrostatically charges them and exposes them to the electrostatic field. Due to the gravity, the particles fall downwards

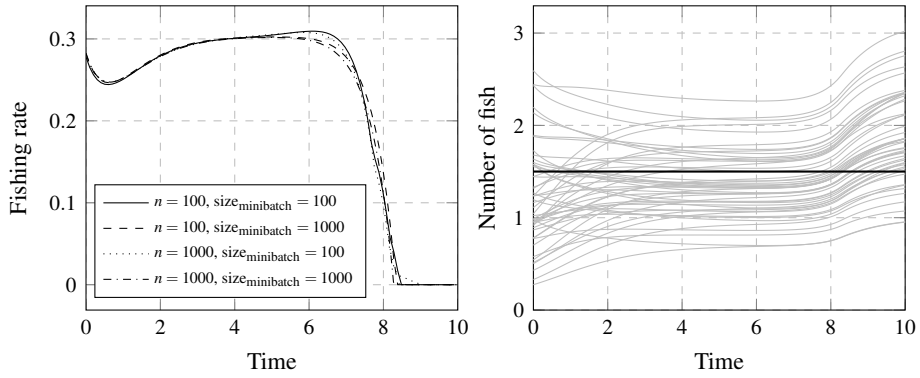

Figure 1: The fishing rate u (left) and the number of fish x (right) for the fishing application from Section 5.1. Note that the desired fish level (bold line) is prescribed only at the final time.

while due to the opposite polarities, they separate. At the bottom of the separator, there are three bins, one for each type and one for middling which is reseparated. The goal is to minimize the voltage such that the separation accuracy is at least $1-\varepsilon$. The decision variables are the voltage U, the position of both electrodes d_l, d_r. The random variables are the particle mass m, charge Q and its initial position $s_x(0), s_y(0)$. More detailed information is presented in Appendix F.2.

We will show the dependence of results on the number of scenarios S. In Figure 2, we fix a design and repeatedly compute the separation accuracy on a randomly selected minibatch with size$_{\text{minibatch}} \in \{10, 100, 1000\}$ and construct histograms. The variance is rather large, especially for the two smaller values. This plays a crucial role in our algorithm since we update the constraint only on a minibatch and smaller minibatches bring a rather large error to the computation.

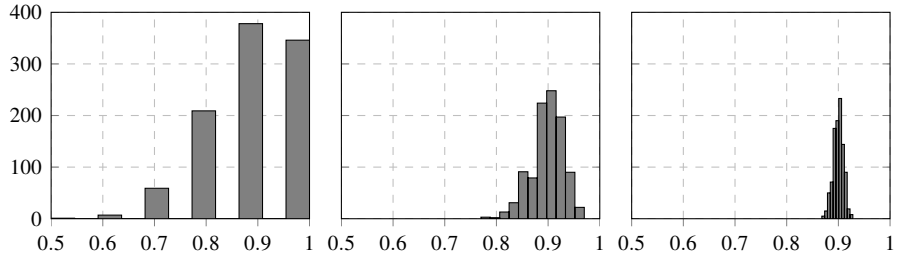

Figure 2: The histograms of the separation accuracy for minibatch sizes size$_{\text{minibatch}} = 10$ (left), size$_{\text{minibatch}} = 100$ (middle) and size$_{\text{minibatch}} = 1000$ (right). These minibatch sizes were used in the computations.

In Table 5 we depict the results for different values of the number of scenarios S and size of the minibatch size$_{\text{minibatch}}$. Since we kept the number of epochs $n_{\text{epoch}} = 20$ constant, this due to (27) resulted in constant number of updates $n_{\text{update}} = 2000$ of the decision variable. We show the optimal voltage U, the separator width $d_r - d_l$, the obtained failure levels $\varepsilon_{\text{data}}$ and $\varepsilon_{\text{true}}$ on the training data and testing data, respectively. Here, by training data, we understand the data on which the algorithm was trained while the testing data refer to computing the failure level on a large number of randomly generated scenarios which the algorithm has not used before. The last two columns refer to the total time and the time needed to evaluate the separation accuracy.

We observe several things. First, the Coulomb force F_c is proportional to $\frac{U}{d_r - d_l}$. Since this force is the main force in the horizontal direction and the particles are supposed to fall into the correct bins, this ratio is constant. Second, the computed failure level $\varepsilon_{\text{data}}$ equals to the requested $\varepsilon = 0.1$ while the true failure level $\varepsilon_{\text{true}}$ is larger when the number of scenarios S used for training is smaller. This makes sense as the algorithm overfitted to the training data.

Table 5: The results for the separator application from Section 5.2. Based on the scenario size, it depicts the optimal voltage U, the optimal separator width $d_r - d_l$, the ratio $\frac{U}{d_r - d_l}$ proportional to the Coulomb force F_c, the failure level on the training data $\varepsilon_{\text{data}}$, the failure level outside of the training data $\varepsilon_{\text{true}}$, the total time and the time needed to evaluate the particle movement inside the separator.

S	$\text{size}_{\text{minibatch}}$	U	$d_r - d_l$	$U/(d_r - d_l)$	$\varepsilon_{\text{data}}$	$\varepsilon_{\text{true}}$	Time [s]	Time g [s]
1000	10	19308	0.178	1.088	0.101	0.116	36	23
10000	100	22337	0.211	1.058	0.100	0.101	69	51
100000	1000	23732	0.227	1.043	0.100	0.100	531	480

In Figure 3 we show the obtained separator. Since the black particles are supposed to fly left and the grey particles are supposed to fly right, the algorithm has produced a good solution. The good results as presented in Table 5 and Figure 3 together with the large variance from Figure 2 indicates that it is indeed necessary to consider the delayed values as in (13) to properly update the quantile.

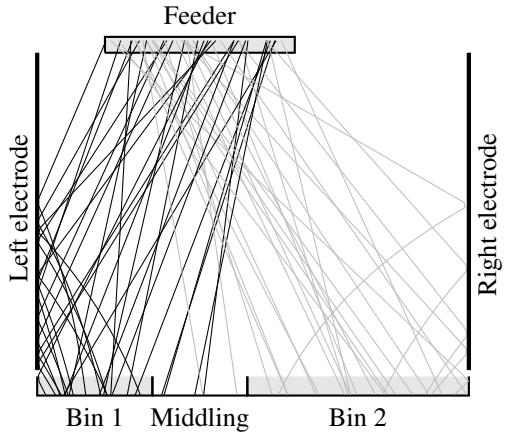

Figure 3: The schema of the free-fall electrostatic separator. Two types of plastics are charged with opposite polarities and placed into the feeder. Due to the gravity, they fall downwards and due to the electrostatic field between the electrodes, they separate.

6.3 Application 3: Optimal design of gas network

For the gas network application we took the same setting as in [2]. The schema of the used network with 1 entry and 11 exit nodes is depicted in Figure 4. The results are depicted in Table 6 (objective f) and Table 7 (failure level). As expected due to (27), the numbers on all diagonals offer comparable results.

We would like to compare our results with the one in [2]. While we managed to obtain the optimal objective 3144.68 with failure level 0.1502, the authors in [2] reported the objective 3145.75 with failure level 0.1500. We can conclude that our method provides a comparable if not a better solution than the one reported earlier. Moreover, the computational time was less than one minute which is significantly faster than the time reported in [2].

Acknowledgements We would like to thank Holger Heitsch for providing us data for the gas network application.

This work was supported by National Natural Science Foundation of China (Grant No. 61850410534), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386), Shenzhen Peacock Plan (Grant No. KQTD2016112514355531) and the Grant Agency of the Czech Republic (19-28231X).

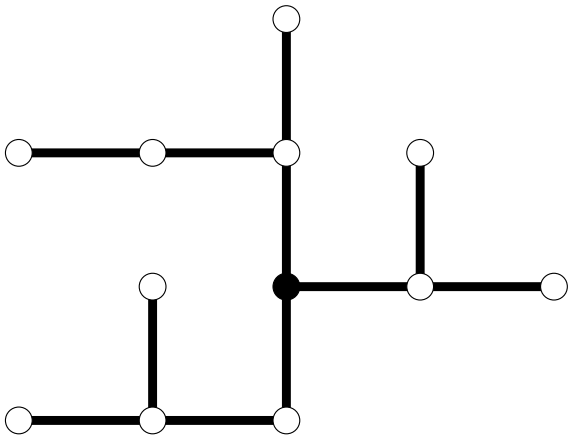

Figure 4: Schema for the gas network from Section 5.3.

Table 6: The obtained objective f for the gas network application from Section 5.3.

	100	250	500	1000	2000
2	3248.80	3169.86	246478.36	810451.46	1202081.95
5	3145.20	3145.04	3221.21	107410.95	632311.21
10	3144.68	3145.17	3145.20	3432.14	110977.10
20	3144.95	3145.56	3145.80	3145.10	3584.16
40	3144.65	3144.97	3144.62	3144.85	3144.31

Table 7: The obtained failure level $\varepsilon_{\text{data}}$ for the gas network application from Section 5.3.

	100	250	500	1000	2000
2	0.113	0.140	0.412	0.829	0.965
5	0.150	0.150	0.153	0.317	0.756
10	0.150	0.150	0.150	0.157	0.333
20	0.150	0.150	0.150	0.150	0.159
40	0.150	0.150	0.150	0.150	0.150

References

[1] L. Adam and M. Branda. Nonlinear chance constrained problems: Optimality conditions, regularization and solvers. *J. Optim. Theory Appl.*, 170(2):419–436, 2016.

[2] L. Adam, M. Branda, H. Heitsch, and R. Henrion. Solving joint chance constrained problems using regularization and Benders' decomposition. *Annals of Operations Resarch*, 10.1007/s10479-018-3091-9.

[3] A. Ben-Tal and A. Nemirovski. Robust convex optimization. *Mathematics of Operations Research*, 23(4):769–805, 1998.

[4] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer, 2011.

[5] L. Bottou, F. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[6] F. Clarke. *Functional analysis, calculus of variations and optimal control*, volume 264. Springer Science & Business Media, 2013.

[7] F. Curtis, A. Wächter, and V. Zavala. A sequential algorithm for solving nonlinear optimization problems with chance constraints. *SIAM Journal on Optimization*, 28(1):930–958, 2018.

[8] M. Dötterl, U. Wachsmuth, L. Waldmann, H. Flachberger, M. Mirkowska, L. Brands, P.-M. Beier, and I. Stahl. *Electrostatic Separation*. Wiley-VCH Verlag GmbH & Co. KGaA, 2016.

[9] A. Geletu, A. Hoffmann, M. Kläppel, and P. Li. An inner-outer approximation approach to chance constrained optimization. *SIAM Journal on Optimization*, 27(3):1834–1857, 2017.

[10] T. Gonzalez Grandon, H. Heitsch, and R. Henrion. A joint model of probabilistic/robust constraints for gas transport management in stationary networks. *Computational Management Science*, 14:443–460, 2017.

[11] C. Gotzes, H. Heitsch, R. Henrion, and R. Schultz. On the quantification of nomination feasibility in stationary gas networks with random load. *Mathematical Methods of Operations Research*, 84(2):427–457, 2016.

[12] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[13] J. Kacerovský, J. Brabec, F. Mach, and P. Karban. Experimental electrostatic separator for charged particles of plastic mixture. In *2016 ELEKTRO*, pages 523–526, 2016.

[14] D. E. Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1997.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, 2012.

[16] S. Lenhart and J. Workman. *Optimal Control Applied to Biological Models*. Chapman & Hall/CRC Mathematical and Computational Biology. Taylor & Francis, 2007.

[17] F. Mach, L. Adam, J. Kacerovský, P. Karban, and I. Doležel. Evolutionary algorithm-based multi-criteria optimization of triboelectrostatic separator. *Journal of Computational and Applied Mathematics*, 270:134–142, 2014.

[18] T. Poggio, K. Kawaguchi, Q. Liao, B. Miranda, L. Rosasco, X. Boix, J. Hidary, and H. Mhaskar. Theory of deep learning iii: explaining the non-overfitting puzzle. *???*, ???(???):???, ???

[19] K. Ragaert, L. Delva, and K. V. Geem. Mechanical and chemical recycling of solid plastic waste. *Waste Management*, 69:24 – 58, 2017.

[20] F. Shan, L. Zhang, and X. Xiao. A smoothing function approach to joint chance-constrained programs. *Journal of Optimization Theory and Applications*, 163(1):181–199, 2014.

[21] P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109(3):475–494, 2001.

[22] W. van Ackooij, A. Frangioni, and W. de Oliveira. Inexact stabilized Benders' decomposition approaches with application to chance-constrained problems with finite support. *Computational Optimization and Applications*, 65(3):637–669, Dec 2016.

[23] W. van Ackooij and R. Henrion. Gradient formulae for nonlinear probabilistic constraints with Gaussian and Gaussian-like distributions. *SIAM Journal on Optimization*, 24:1864–1889, 2014.

[24] M. Wang and D. Bertsekas. Stochastic first-order methods with random constraint projection. *SIAM Journal on Optimization*, 26(1):681–717, 2016.

[25] M. M. Wiecek and G. M. Dranichak. *Robust Multiobjective Optimization for Decision Making Under Uncertainty and Conflict*, chapter 4, pages 84–114. 2016.

[26] G. Wu, J. Li, and Z. Xu. Triboelectrostatic separation for granular plastic waste recycling: A review. *Waste Management*, 33(3):585 – 597, 2013.

[27] W. Xie and S. Ahmed. On quantile cuts and their closure for chance constrained optimization problems. *Mathematical Programming*, Sep 2017.

A Properties of q

In this section, we will show that q defined in (2) is Lipschitz continuous and if the index from (10) is unique, then q is (locally) differentiable. We assume that $g(\cdot,\xi_i)$ is Lipschitz continuous for all i. Fix any $\bar{x}$. Due to (2), there is a disjoint partition I, J_1 and J_2 of $\{1,\dots,S\}$ such that

$$\begin{aligned} g(\bar{x},\xi_i) &= q(\bar{x}) \text{ for all } i \in I,\\ g(\bar{x},\xi_i) &< q(\bar{x}) \text{ for all } i \in J_1,\\ g(\bar{x},\xi_i) &> q(\bar{x}) \text{ for all } i \in J_2. \end{aligned}$$

Due to the assumed continuity, there is a neighborhood of $\bar{x}$ such that for all y from this neighborhood, there exist an index $i(y) \in I$ such that

$$g(y,\xi_{i(y)}) = q(y). \tag{28}$$

But since $i(y) \in I$, we have

$$|q(y)-q(x)| = |g(y,\xi_{i(y)}) - g(\bar{x},\xi_{i(y)})| \le L\|\bar{x}-y\|,$$

where L is the Lispchitz constant of $g(\cdot,\xi_{i(y)})$. Thus, q is Lipschitz continuous. If the index in (10), then the index in (28) is unique and q. Thus, q differentiable at $\bar{x}$ whenever $g(\cdot,\xi_{i(\bar{x})})$ is differentiable at the same point.

B Detailed algorithm for updating the quantile

In Section 3.2, we mentioned the basic ideas of how to efficiently update the quantile. Since the details are rather technical and not particularly interesting, we summarize them here in the Appendix.

At iteration k we know the sorting permutation π of $\{1,\dots,S\}$ which sorts z^k into s^k and we compute the sorting permutation φ of I^{k+1} which sorts $g_i^k := g(x^{k+1},\xi_i)$, $i \in I^{k+1}$ into h_j^k. Namely, we have

$$\begin{aligned} s_i^k &= z_{\pi(i)}^k, \quad \pi(i) \in \{1,\dots,S\},\\ h_j &= g_{\varphi(j)}, \quad \varphi(j) \in I^{k+1}. \end{aligned} \tag{29}$$

According to (13), we want to update z^k only on minibatch I^{k+1}. Then we have

$$z_i^{k+1} = \begin{cases} g_i^k = g(x^{k+1},\xi_i) & \text{if } i \in I^{k+1},\\ z_i^k & \text{otherwise.} \end{cases} \tag{30}$$

Thus, we have two sorted arrays s^k and h^k from (29) and we need to merge them together.

To this aim, we will employ three indices: i will run on s^k, j will run on h^k and finally l will run on s^{k+1}. We initialize all indices to $i = j = l = 1$. In every iteration, we first check whether $\pi(i) \in I^{k+1}$. If this is the case, $z_{\pi(i)}^k$ was replaced by $g_{\pi(i)}$ in (30). Thus, we are not interested in $s_i^k = z_{\pi(i)}^k$, we increase i by one and repeat. In the opposite case of $\pi(i) \notin I^{k+1}$ we set

$$s_l^{k+1} = \begin{cases} s_i^k & \text{if } s_i^k \le h_j^k,\\ h_j^k & \text{otherwise.} \end{cases}$$

In other words, we insert to s_l^{k+1} the minimum of s_i^k and h_j^k. Since both s^k and h are sorted, s^{k+1} is sorted as well. Now, we need to obtain a permutation ϕ mapping z^{k+1} into its sorted variant s^{k+1}, namely it needs to satisfy

$$s_l^{k+1} = z_{\phi(l)}^{k+1}. \tag{31}$$

Due to (29), this can be simply obtained by

$$\phi(l) = \begin{cases} \pi(i) & \text{if } s_i^k \leq h_j^k, \\ \varphi(j) & \text{otherwise.} \end{cases}$$

Since s_l^{k+1} has been filled, we increase l by one. Finally, if s_l^{k+1} was filled by s_i^k, we increase i by one and if s_l^{k+1} was filled by h_j^k, we increase j by one. In both cases, we repeat the whole precedure. After sorting z^{k+1}, the quantile can be computed as

$$q^{k+1} = s_{\text{index}}^{k+1},$$

where index $= \lceil \mathrm{S}(1-\varepsilon) \rceil$.

To summarize, the procedure above takes as input the sorted array s^k from the previous iteration and the permutation π satisfying (29). It then replaces values g on indices I^{k+1} and returns the sorted array s^{k+1} and the permutation satisfying (31). These outputs can be directly used as inputs for the next iteration and thus, the sorting can be done efficiently. Note that it is not even necessary to store the unsorted array z^k.

This whole procedure is written down in Algorithm B.1. We made two small changes to the procedure described above. First, in steps 20-24 we needed to take care about indices overflowing the arrays. Second, it may be time-consuming to check whether $\pi(i) \in I^{k+1}$. For this reason, in step 3 we assume that $I^{k+1} = \{i_{\text{low}}, \ldots, i_{\text{high}}\}$ and then insert the randomness into selection minibatches by permuting ξ as describing in Section 3.3.

C Backpropagation

In this short section, we describe how the well-known method backpropagation [12] computes derivatives. Consider a general function

$$F(u) := \sum_{t=0}^{T} f_t(u_t, x_t),$$

where x_0 is given and we have

$$x_{t+1} = h_t(u_t, x_t).$$

Since for every u, we can uniquely compute x, we want to compute the derivative of the objective with respect to u. We have

$$\frac{\partial F}{\partial u_t} = \nabla_u f_t(u_t, x_t) + \sum_{s=t+1}^{T} \nabla_x f_s(u_s, x_s) \frac{\partial x_s}{\partial u_t} = \nabla_u f_t(u_t, x_t) + \sum_{s=t+1}^{T} \nabla_x f_s(u_s, x_s) \frac{\partial x_s}{\partial x_{s-1}} \cdots \frac{\partial x_{t+2}}{\partial x_{t+1}} \frac{\partial x_{t+1}}{\partial u_t}$$

Based on this expression, define

$$a_t := \sum_{s=t+1}^{T} \nabla_x f_s(u_s, x_s) \frac{\partial x_s}{\partial x_{s-1}} \cdots \frac{\partial x_{t+2}}{\partial x_{t+1}}.$$

Then with $a_T = 0$, we have the chaining relation

$$a_t = a_{t+1} \frac{\partial x_{t+2}}{\partial x_{t+1}} + \nabla_x f_{t+1}(x_{t+1}, u_{t+1}) = a_{t+1} \nabla_x h_{t+1}(u_{t+1}, x_{t+1}) + \nabla_x f_{t+1}(x_{t+1}, u_{t+1})$$

and the derivative can be computed as

$$\frac{\partial F}{\partial u_t} = \nabla_u f_t(u_t, x_t) + a_t \frac{\partial x_{t+1}}{\partial u_t} = \nabla_u f_t(u_t, x_t) + a_t \nabla_u h_t(x_t, u_t).$$

Note that the function value $F(u)$ can be computed in one forward swipe and the Jacobian $\nabla J(u)$ can be computed in one backward swipe.

Algorithm (Auxiliary) B.1 For finding the quantile q^{k+1} from (14)

Input: sorted arrays s^k and h^k with the corresponding permutations (29)
Output: sorted array s^{k+1} with the corresponding permutation (31)

$i = j = k \leftarrow 1$
while true do
  if $\pi(i) \geq i_{\text{low}}$ **and** $\pi(i) \leq i_{\text{high}}$ **then** ▷ Value was replaced, ignore it
    $i \leftarrow i+1$
    if $i > S$ **then break while**
  else
    if $s_i^k \leq h_j$ **then** ▷ Add s_i^k to the new array
      $s_l^{k+1} \leftarrow s_i^k$
      $\phi(l) \leftarrow \pi(i)$
      $l \leftarrow l+1$, $i \leftarrow i+1$
      if $i > S$ **then break while**
    else ▷ Add h_j to the new array
      $s_l^{k+1} \leftarrow h_j^k$
      $\phi(l) \leftarrow \varphi(j)$
      $l \leftarrow l+1$, $j \leftarrow j+1$
      if $j > i_{\text{high}} - i_{\text{low}} + 1$ **then break while**
    end if
  end if
end while
if $i < S$ **then** ▷ Handle the remaining terms
  Array h^k has been inserted to s^{k+1}. Insert the remaining of s^k to s^{k+1} in a similar way as above.
else
  Array s^k has been inserted to s^{k+1}. Insert the remaining of h^k to s^{k+1} in a similar way as above.
end if
$q^{k+1} \leftarrow s_{\text{index}}^{k+1}$

D Computation of Derivatives

Similarly, the objective function and the constraints can be discretized into

$$\begin{aligned} f(u) &:= f(u,x(u)) := \Delta t \sum_{t=0}^{T-1} \left(p_t u_t x_t - d_t u_t^2 x_t^2 - c_t u_t \right), \\ g(u) &:= g(u,x(u)) := x_T. \end{aligned} \tag{32}$$

In this section, we compute the derivatives for functions (32) form the fishing application. Since the derivatives of functions from the separator application can be computed in an identical way, we omit it here. Using the backpropagation method from Section C, for $t = 0, \ldots, T-1$ we can compute the derivative by

$$\frac{\partial f}{\partial u_t} = \Delta t \left(p x_t - 2 d u_t x_t^2 - c \right) - a_t \Delta t x_t,$$

where $a_T = a_{T-1} = 0$ and for $t = 0, \ldots, T-2$ we set

$$a_t = a_{t+1} \left(1 + \Delta t \left(r - \frac{2}{K} r x_{t+1} - u_{t+1} \right) \right) + \Delta t \left(p u_{t+1} - 2 d u_{t+1}^2 x_{t+1} \right).$$

Note that this is a discretized version of the adjoint equation which can be obtained by the techniques from optimal control theory [6]. Note that the computation above contains only vectors while a naive application of the chain rule would result in $\nabla_u f(u,x(u)) + \nabla_x f(u,x(u)) \frac{\partial x}{\partial u}$, where $\frac{\partial x}{\partial u}$ is a matrix. Similarly, for the terminal state function g we have

$$\frac{\partial g}{\partial u_t} = -b_t \Delta t x_t,$$

where $b_T = 0$ and

$$b_{T-1} = b_T\left(1+\Delta t\left(r-\frac{2}{K}rx_T-u_T\right)\right)+1,$$
$$b_t = b_{t+1}\left(1+\Delta t\left(r-\frac{2}{K}rx_{t+1}-u_{t+1}\right)\right).$$

E Differences between the batch and stochastic gradient descents

In Section 5.1 we showed that in problem (1) we can replace the random variable p_t by its expectation $\mathbb{E}p_t$. These two problems are equivalent when the batch gradient descent is applied. However, this is no longer true for the stochastic gradient descent. To show this, we proposed the following experiment. We fixed the optimized design and randomly generated one set with $\text{size}_{\text{minibatch}}$ scenarios. Then we considered the same set where p_t was replaced by its expectation $\mathbb{E}p_t$. When we computed the difference in the values of the objective f, the differences were negligible. However, the difference in Jacobians of the objective ∇f was rather big. The value of the gradient is showed in Figure 5 (left) while the relative difference between both settings is depicted in Figure 5 (right). Due to this difference, the two problems above are no longer equivalent when the stochastic gradient descent is used.

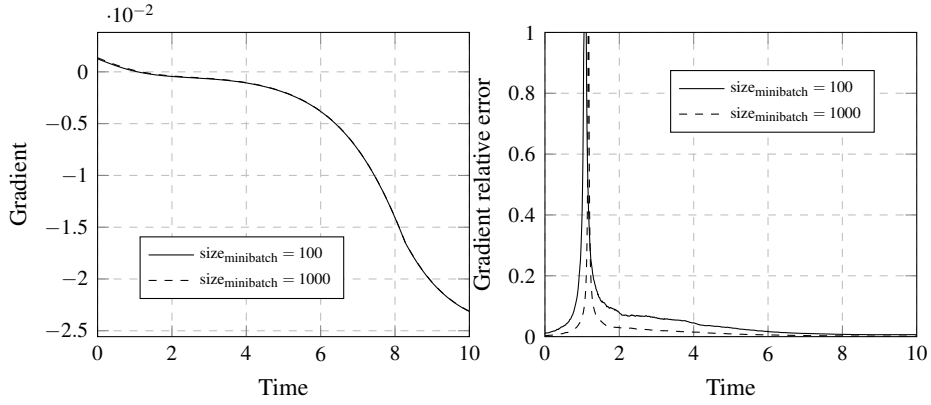

Figure 5: The gradient (left) and the relative error in gradient when the random prices are replaced by their expectation (right). Both figures are for a fixed design.

F Additional information for the numerical part

In this section, we provide additional information about parameter choices from the numerical section.

F.1 Application 1: Optimal control of fish population

The random variables followed the $r \sim N(1, 0.01)$, $K \sim N(2, 0.25)$ and $x_0 \sim U(\frac{K}{4}, K)$ distributions where r was truncated at 0.01 and K was truncated at 1. The random prices were initialized in $\bar{p}_t = 2$, $\bar{d}_t = 0.1$, $\bar{c}_t = 1.5$ and then followed the process

$$\begin{pmatrix} p_0 \\ d_0 \\ c_0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0.1 \\ 1.5 \end{pmatrix}, \qquad \begin{pmatrix} p_{t+1} \\ d_{t+1} \\ c_{t+1} \end{pmatrix} = \begin{pmatrix} p_t \\ d_t \\ c_t \end{pmatrix} + N\left(\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \frac{0.01}{n}\begin{pmatrix} 1 & 0.025 & 0.5 \\ 0.025 & 0.05 & 0.025 \\ 0.5 & 0.025 & 1 \end{pmatrix}\right).$$

F.2 Application 2: Optimal control of electrostatic separator

For the random variables, we loosely followed [13]. For the positively changed particles, we had $m \sim N(6 \cdot 10^{-6}\text{kg}, (2 \cdot 10^{-6}\text{kg})^2)$ and $Q \sim N(5.5 \cdot 10^{-11}\text{C}, (2.2 \cdot 10^{-11}\text{C})^2)$ while for the negatively charged

particles, we had $m \sim N(8 \cdot 10^{-6}\text{kg}, (1.5 \cdot 10^{-6}\text{kg})^2)$ and $Q \sim N(-5 \cdot 10^{-11}\text{kg}, (2 \cdot 10^{-11}\text{kg})^2)$. The mass was truncated at $1e-7$kg while there was no truncation for the charge. The initial position followed $s_x \sim U(-0.05\text{m}, 0.05\text{m})$ and $s_y \sim U(0\text{m}, 0.05\text{m})$. The separator was 1m tall. The middle bin was located at $[-0.025\text{m}, 0.025\text{m}]$.