Chance Constrained Optimization: Bi-level reformulation

Master Research Project

Anoush Azar-Pey supervised by Saeed Masiha
INDY Lab, EPFL

June 6, 2025

Contents

1 Chance Constrained Optimization

1.1 Context

Chance Constrained Optimization (CCO) [KHVR20]: A way of modeling uncertainty and controlling rare events in optimization. It differs from deterministic optimization, such as Linear or Quadratic Programming. Chance constraints are modeled by what is called a *chance function* $g(x, Z)$, parametrized by the decision variable x, where Z is a random variable that follows some known distribution. They have been used in many applications, such as reinforcement learning [Cho+17], telecommunications [Med98] and in power system design [GX19].

Definition 1 (Chance Constrained Optimization Problem)

$$\min_{x \in \mathcal{X}} f(x) \ \ s.t. \ \mathbb{P}\{g(x, Z) \leq 0\} \geq 1 - \delta \tag{1}$$

where $\mathcal{X}$ is a closed convex subset of $\mathbb{R}^d$, $Z \sim P_Z$ is a random variable, $f(x)$ is our objective function, $g(x, Z)$ our chance function *and $1 - \delta$ our safety probability level, typically close to 1.*

We assume f and g to be *convex functions* w.r.t. x [ROC70].

1.2 Objectives

In this project, we aim to design two simple chance constrained optimization algorithms and to compare them with previous work on this topic [LMA21; Zha+24]. The main steps will be:

1. *Bi-level* problem reformulation by introducing a scalar auxiliary variable.
2. Develop two simple methods: a *first-order* and a *zeroth-order* method.
3. Numerical experiments and comparison with [LMA21] and [Zha+24].

2 Bi-level reformulation

First, we reformulate the chance constraint using the notion of $(1 - \delta)$-quantile.

Definition 2 (Quantile) *Let X be a random variable, $F_X(t)$ its Cumulative Distribution Function and $\delta \in [0, 1]$ our level. X's $(1 - \delta)$-quantile[Hab96] denoted $Q_{1-\delta}(X)$ is defined as:*

$$Q_{1-\delta}(X) = \inf\{t \in \mathbb{R} : F_X(t) \geq 1 - \delta\}. \tag{2}$$

Intuitively, it can be interpreted as the inverse of the CDF[1] of a random variable, i.e. $F_X^{-1}(1 - \delta)$.

We can rewrite our previously defined chance constraint using quantiles [LMA21]:

$$\mathbb{P}\{g(x, Z) \leq 0\} \geq 1 - \delta \iff Q_{1-\delta}(g(x, Z)) \leq 0. \tag{3}$$

Intuitively, this means that the left $1 - \delta$ proportion of the mass of $g(x, Z)$'s density should lie below 0. Using this equivalence, it follows that our main (Problem 1) can be rewritten as *Bi-level problem*:

$$\begin{cases} \min\limits_{x \in \mathcal{X}} & f(x) \\ \text{s.t.} & \mathbb{P}\{g(x, Z) \leq 0\} \geq 1 - \delta \end{cases} \longleftrightarrow \begin{cases} \min\limits_{x \in \mathcal{X}} & f(x) \\ \text{s.t.} & s \leq 0 \\ & s = Q_{1-\delta}(g(x, Z)) \end{cases} . \tag{4}$$

Challenge $s = Q_{1-\delta}(g(x, Z))$ is a non-convex non-smooth constraint. To overcome it, we introduce the following notion.

[1] CDF stands for Cumulative Distribution Function, which describes the probability that a random variable takes a value less than or equal to a given point.

2.1 Superquantile reformulation

2.1.1 Definition of Superquantile

Definition 3 (Superquantile) *Let X be a random variable, $\delta \in [0,1]$ our level and $Q_{1-\delta}(X)$ our quantile at the level $1-\delta$. X's $(1-\delta)$-superquantile, denoted $SQ_{1-\delta}(X)$ is defined as:*

$$SQ_{1-\delta}(X) = \mathbb{E}\{X \mid X \geq Q_{1-\delta}(X)\} \tag{5}$$

which can be equivalently defined as (from [LMA21])

$$SQ_{1-\delta}(X) = \frac{1}{\delta}\int_{1-\delta}^{1} Q_p(X)dp. \tag{6}$$

Superquantiles can be intuitively interpreted as the average mass of high values of a density function.

2.1.2 Variational formulation for Superquantile

For a given random variable X and a level δ, we will estimate these two quantities with the following convex minimization problem:

Proposition 1 *[LMA21]*

$$SQ_{1-\delta}(X) = \min_{s\in\mathbb{R}} s + \frac{1}{\delta}\mathbb{E}[\max\{X-s,0\}]. \tag{7}$$

Moreover, the left endpoint of the solution set of (7) is $Q_{1-\delta}(X)$.

Intuitively, s is shifting X's density to the left or to the right such that a $1-\delta$ proportion of its values lie below 0.

2.2 Bi-level reformulation

Using (Proposition 1), we introduce the following *jointly convex* function $G_\delta(x,s)$, involving the auxiliary variable s:

$$G_\delta(x,s) = s + \frac{1}{\delta}\mathbb{E}_{P_Z}[\max\{g(x,Z)-s,0\}] \tag{8}$$

and further rewrite our problem as:

$$\begin{cases} \min\limits_{x\in\mathcal{X}} & f(x) \\ \text{s.t.} & s \leq 0 \\ & s = Q_{1-\delta}(g(x,Z)) \end{cases} \longleftrightarrow \begin{cases} \min\limits_{x\in\mathcal{X}} & f(x) \\ \text{s.t.} & s \leq 0 \\ & s \in S(x) = \arg\min\limits_{s\in\mathbb{R}} G_\delta(x,s) \end{cases}. \tag{9}$$

2.3 Penalty method

We will assume that $S(x) = \arg\min_{s\in\mathbb{R}} G_\delta(x,s)$ is a singleton, i.e. it contains a unique solution, that we will denote $s^*(x)$, leading to the following reformulation of the CCO:

$$\min_{x\in\mathcal{X}} f(x) \text{ s.t. } s^*(x) \leq 0. \tag{10}$$

Penalty method: We begin by reformulating the optimization problem in (10) through its mini-max reformulation, expressed as:

$$\min_{x\in\mathcal{X}} \max_{\lambda\geq 0} f(x) + \lambda s^*(x). \tag{11}$$

However, the function $\max_{\lambda\geq 0} f(x) + \lambda s^*(x)$ is non-smooth with respect to x. To address this, we introduce a regularization term that smooths the objective:

$$\min_{x\in\mathcal{X}} \max_{\lambda\geq 0} P(x,\lambda) = \min_{x\in\mathcal{X}} \max_{\lambda\geq 0} \left[f(x) + \lambda s^*(x) - \frac{\mu}{2}\lambda^2\right]. \tag{12}$$

Here, the added μ-regularization term $-\frac{\mu}{2}\lambda^2$ serves to soften the effect of the λ-penalized constraint $\lambda s^*(x)$. The first-order optimality condition states

$$\frac{\partial P(x,\lambda)}{\partial \lambda} = s^*(x) - \mu\lambda = 0 \iff \lambda = \frac{s^*(x)}{\mu}. \tag{13}$$

Since $\lambda \geq 0$ we take $\lambda = \left[\frac{s^*(x)}{\mu}\right]_+$ and finally rewrite (Problem 10) as the simpler:

$$\min_{x\in\mathcal{X}} F(x,s^*) = \min_{x\in\mathcal{X}} f(x) + s^*(x)\left[\frac{s^*(x)}{\mu}\right]_+ \tag{14}$$

with $F(x,s^*)$ being our *objective function*, $s^*(x)$ denoting $\arg\min_{s\in\mathbb{R}} G_\delta(x,s)$, $[\cdot]_+$ the $\max(\cdot,0)$ or $\mathrm{ReLU}(\cdot)$ function and $f(x)$ being the function to minimize specific to the instance of the problem we are trying to solve.

3 Algorithms

3.1 First-order method

In order to minimize our F function, we will first use a *first-order method*, the *gradient descent algorithm*. We therefore need to compute the partial derivatives $\frac{\partial F\{x,s^*(x)\}}{\partial x}$ and $\frac{\partial}{\partial x}s^*(x)$.

$$\begin{aligned} \frac{\partial F\{x,s^*(x)\}}{\partial x} &= \nabla f(x) + \frac{\partial s^*(x)}{\partial x}\left[\frac{s^*(x)}{\mu}\right]_+ + s^*(x)\begin{cases}\frac{\partial s^*(x)}{\mu} & \text{if } s^*(x)>0\\ 0 & \text{otherwise}\end{cases} \\ &= \nabla f(x) + 2\frac{\partial s^*(x)}{\partial x}\left[\frac{s^*(x)}{\mu}\right]_+. \end{aligned} \tag{15}$$

We do not have a closed-form formula for $s^*(x)$ because it results from a minimization problem. We know that $s^*(x)$ is a minimizer of $G_\delta(x,s)$, therefore we have:

$$\begin{aligned} \frac{\partial G_\delta\{x,s^*(x)\}}{\partial s} = 0 &\implies \frac{d}{dx}\left[\frac{\partial G_\delta\{x,s^*(x)\}}{\partial s}\right] = 0 \\ &\iff \frac{\partial^2}{\partial x\partial s}G_\delta\{x,s^*(x)\} + \frac{\partial}{\partial x}s^*(x)\frac{\partial^2}{\partial s^2}G_\delta\{x,s^*(x)\} = 0 \\ &\iff \frac{\partial}{\partial x}s^*(x) = -\frac{\partial^2}{\partial x\partial s}G_\delta\{x,s^*(x)\}\left[\frac{\partial^2}{\partial s^2}G_\delta\{x,s^*(x)\}\right]^{-1}. \end{aligned} \tag{16}$$

We know that $\frac{\partial^2}{\partial s^2}G_\delta\{x,s^*(x)\}$ is *invertible* because $s^*(x)$ is unique.

We introduce our first algorithm, that we will refer to as `Algorithm 1`:

Algorithm 1 Our first-order algorithm

```
x ← x_0
i ← 0
while i < # iters do
    Sample from the Z distribution a fixed number of times
    Estimate $s^*(x)$ by minimizing $G_\delta(x,s)$ over $s$ using GD with $\frac{\partial}{\partial s}G_\delta(x,s)$
    if $s^* \leq 0$: $s^*$ is a feasible solution, else $s^*$ is unfeasible
    Update $x$ using GD with $\frac{\partial}{\partial x}F(x,s^*)$, i.e. $x \leftarrow x - \gamma\frac{\partial}{\partial x}F(x,s^*)$
    i ← i + 1
end while
```

3.2 Zeroth-order method

3.2.1 2-point estimator

Then, we introduce *zeroth-order methods*[NS17]. They estimate the first-order gradient of a function via the following procedure and subsequent gradient descent step:

$$\begin{aligned} &\text{Sample } u \text{ from } \{u : \|u\|_2 = 1\} \\ &\hat{\nabla}_k h(x) := \frac{h(x+uk) - h(x-uk)}{2k} \cdot u \\ &x_{t+1} \leftarrow x_t - \eta \hat{\nabla}_k h(x_t). \end{aligned} \tag{17}$$

In our case we estimate the gradient of our function F (not G):

$$\hat{\partial}_k F\{x, s^*(x)\} := \frac{F\{x+uk, s^*(x+uk)\} - F\{x-uk, s^*(x-uk)\}}{2k} \cdot u. \tag{18}$$

We pick the *learning rate* η as our k parameter. We compute $s^*(x \pm up)$ the same way we compute $s^*(x)$, i.e. using GD with $\frac{\partial}{\partial s} G_\delta(x, s)$.

We introduce our first algorithm, that we will refer to as `Algorithm 2`:

Algorithm 2 Our zeroth-order algorithm (2-point estimator)

```
$x \leftarrow x_0$
$i \leftarrow 0$
while $i <$ # iters do
    Sample from the $Z$ distribution a fixed number of times
    Estimate $s^*(x)$ by minimizing $G_\delta(x, s)$ over $s$ using GD with $\frac{\partial}{\partial s} G_\delta(x, s)$
    if $s^* \leq 0$: $s^*$ is a feasible solution, else $s^*$ is unfeasible
    Sample $k$, the direction of the perturbation for the zeroth-order method
    Estimate $s^*(x+uk)$ and $s^*(x-uk)$ by minimizing $G_\delta(x \pm uk, s)$ over $s$ using GD with $\frac{\partial}{\partial s} G_\delta(x \pm uk, s)$
    Approximate the gradient of $F$ with $\hat{\partial}_k F\{x, s^*(x)\}$
    Update $x$ using GD with $\hat{\partial}_k F\{x, s^*(x)\}$, i.e. $x \leftarrow x - \gamma \hat{\partial}_k F\{x, s^*(x)\}$
    $i \leftarrow i + 1$
end while
```

As demonstrated in our numerical experiments (Section 4), the *2-point estimator* performs well for uni-variate problems due to the inherent uni-dimensionality of the search space. However, this approach may exhibit suboptimal performance in higher-dimensional problems, where the infinite-dimensional nature of the search space renders directional exploration insufficient for accurate gradient estimation.

In the following sections, we propose enhancements to Algorithm 2. For uni-variate problems, we employ the *2-point estimator* introduced earlier, while for multi-dimensional problems, we introduce a refined *4-point estimator* to address the challenges of the former in higher-dimensional optimization.

3.2.2 4-point estimator

We propose a *4-point estimator* specifically designed for bi-dimensional problems. This method leverages orthogonal perturbations to span the full 2D space, ensuring robust gradient estimation. The algorithm proceeds as follows:

1. Sample a random perturbation direction δ_1 uniformly on the unit circle.
2. Generate a second perturbation δ_2 orthogonal to δ_1 (i.e., $\delta_1 \perp \delta_2$).

3. Evaluate the objective function at four perturbed points: $x \pm \delta_1$ and $x \pm \delta_2$.

The gradient estimate is calculated independently for each pair of symmetric perturbations (δ_1 and δ_2) using finite difference formulas. The final estimator combines these two directional gradients using averaging to produce a more accurate gradient approximation in $\mathbb{R}^2$. This orthogonal sampling strategy ensures that the estimator can capture curvature information in all directions, overcoming the directional bias inherent in the *2-point* approach.

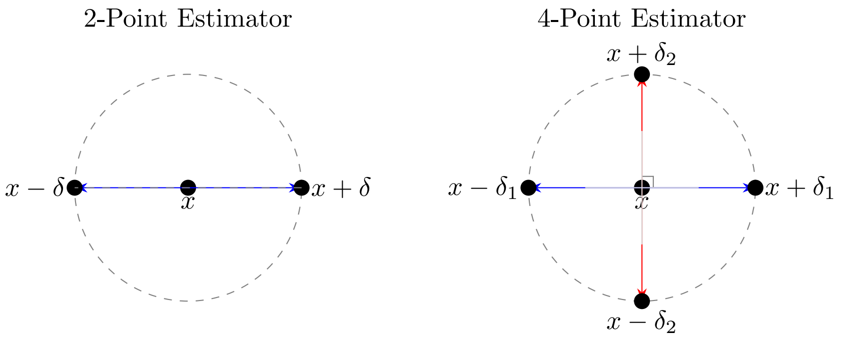

Figure 1: Gradient estimation methods in 2D space: Left: 2-point estimator with symmetric perturbations $\pm\delta$ in a single direction, spanning only a line. Right: 4-point estimator with orthogonal perturbations $\pm\delta_1$ and $\pm\delta_2$, enabling full 2D span through linear combinations.

3.2.3 Stochastic perturbations

A common challenge in zeroth-order optimization algorithms is premature convergence to suboptimal solutions, often caused by inappropriate perturbation magnitudes. Specifically, fixed perturbation scales may be insufficiently large to escape local optima or excessively large, leading to noisy updates. To address this limitation, we introduce a stochastic scaling factor α sampled independently at each iteration. This scalar is multiplied with the perturbation vector to dynamically adjust its amplitude, enabling adaptive exploration of the search space. By sampling $\alpha \sim \mathcal{U}\left(\frac{1}{a}, a\right)^2$, for example, with $a = 3/2$, the algorithm balances exploration (larger α) and exploitation (smaller α) while maintaining computational efficiency.

3.2.4 Update clipping

Zeroth-order gradient estimation can produce unstable updates due to high variance in the objective function $f(x)$ or in the chance function $g(x, Z)$. In particular, when Z exhibits significant variability or the function f is ill-conditioned, the estimated updates may have unbounded norms, leading to numerical instability. To mitigate this, we take inspiration from [Zha+20] and adopt the technique of *update clipping*. This method enforces an upper bound on the norm of the update vector $\|\Delta x\|$, ensuring that $\|\Delta x\| \leq C$, where C is a user-defined clipping threshold. If the computed update exceeds this bound, it is projected onto the ℓ_2-ball of radius C. This regularization strategy prevents extreme parameter shifts while preserving the algorithm's convergence guarantees.

4 Numerical Experiments

We will now introduce multiple examples to assess the performance of `Algorithm` 1 and `Algorithm` 2 in comparison with several known algorithms such as:

- The *Bundle algorithm* from [LMA21].

[2] $\mathcal{U}$ denotes the uniform distribution.

- 3 algorithms from [Zha+24]: Conformal Predictive Programming - Mixed Integer Programming (CPP-MIP), Conformal Predictive Programming - Karush Kuhn Tucker (CPP-KKT) and the Sample Average Approximation (SAA).

4.1 Example 1

We start by introducing a simple example as follows: we pick

$$f(x) = (x-2)^2 \tag{19}$$

as our convex objective function, and [3]

$$g(x, Z) = xZ - 1, \quad Z \sim \mathcal{N}(1, 1). \tag{20}$$

We know that the global minimum of f is $x = 2$, but we designed our constraint so that this point is unfeasible. We can derive the analytical optimal point for this problem by computing $\{x : xZ - 1 \leq 0\}$ and picking x^* from this set that minimizes f:

$$x^* = \frac{1}{\Phi^{-1}(1-\delta)+1} \quad \text{where } \Phi \text{ is the CDF of } \mathcal{N}(0,1). \tag{21}$$

To determine whether a solution is *optimal* we use the sub-optimality [LMA21] metric:

$$\frac{|f(x_k) - f(x^*)|}{|f(x^*)|}. \tag{22}$$

To determine whether a given point is feasible, we use the *Empirical coverage* metric [Zha+24]:

$$\mathrm{EC}(x) = \mathbb{E}_Z\left[\mathbf{1}\{g(x,Z) \leq 0\}\right]. \tag{23}$$

We choose $1-\delta = 0.95$ as our safety probability level.

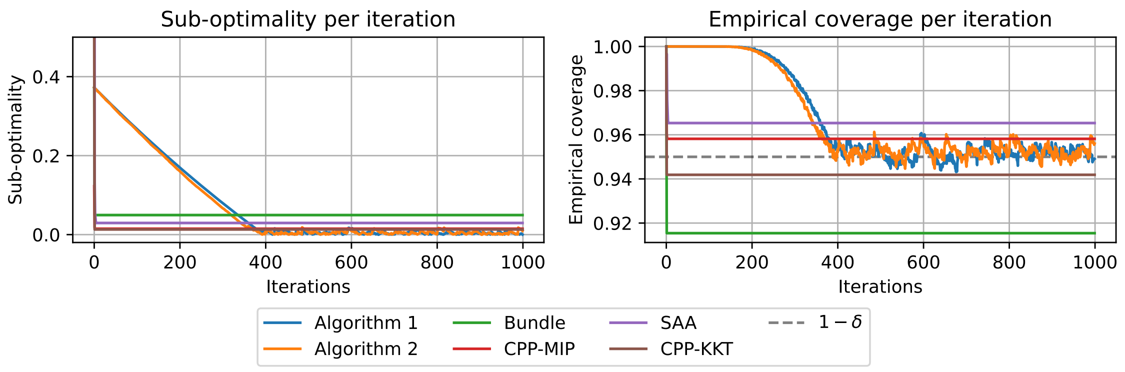

Figure 2: Convergence of all algorithms

In figure 4.1, we plot, for all algorithms, on the left their sub-optimality per iteration and on the right their corresponding empirical coverage. Table 4.1 summarizes the performance of Algorithms 1 and 2, and the *Bundle algorithm* by reporting the average sub-optimality and empirical coverage over their last 10 iterations. For `CPP-MIP`, `SAA`, and `CPP-KKT`, we instead report these metrics only for their final iteration, as these methods typically converge in very few steps.

Interestingly, on this simple problem, we observe that Algorithms 1 and 2 are the best performing. The `Bundle algorithm`'s solution is unfeasible, since its empirical coverage is below our $1-\delta$ safety probability treshold and the solutions of `CPP-MIP`, `SAA`, `CPP-KKT` have a higher sub-optimality.

[3]$\mathcal{N}$ denotes the normal (Gaussian) distribution.

Algorithm	Sub-optimality	Empirical Coverage
Algorithm 1	0.0012	0.9494
Algorithm 2	0.0133	0.9575
Bundle	0.0495	0.9154
CPP-KKT	0.013	0.9428
CPP-MIP	0.0149	0.9590
SAA	0.0294	0.9657

Table 1: Algorithm performance: average of the last iterations

4.2 Example 2

4.2.1 Example 2.1

Now we present a bi-variate problem from [LMA21]. The objective function

$$f(x) = \frac{1}{2}(x-a)^T Q(x-a) \text{ with } a = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, Q = \begin{pmatrix} 5.5 & 4.5 \\ 4.5 & 5.5 \end{pmatrix} \tag{24}$$

is quadratic with $Q \succeq 0$ and thus convex [ROC70]. The chance function is

$$\begin{aligned} g(x,Z) = Z^T W(x) Z + w^T Z \quad \text{with } W(x) &= \begin{pmatrix} x_1^2 + 0.5 & 0 \\ 0 & |x_2 - 1|^3 + 0.2 \end{pmatrix} \\ w &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ Z \sim \mathcal{N}(\mu, \Sigma) \quad \text{with } \mu &= \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma = \begin{pmatrix} 20 & 0 \\ 0 & 20 \end{pmatrix}. \end{aligned} \tag{25}$$

As in the paper, we choose an unusually low probability level $1 - \delta \approx 0.033$.

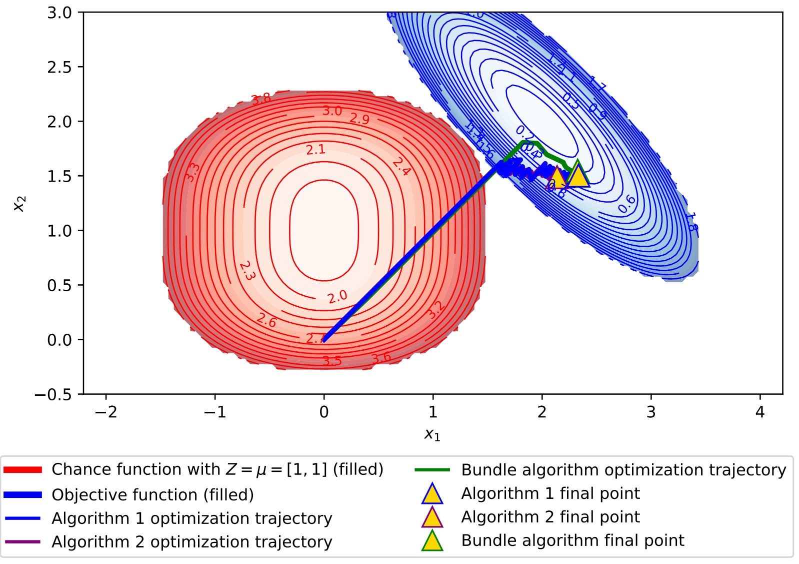

Figure 3: Optimization trajectories of Algorithms 1 and 2, and the `Bundle algorithm`, starting from $[0, 0]$

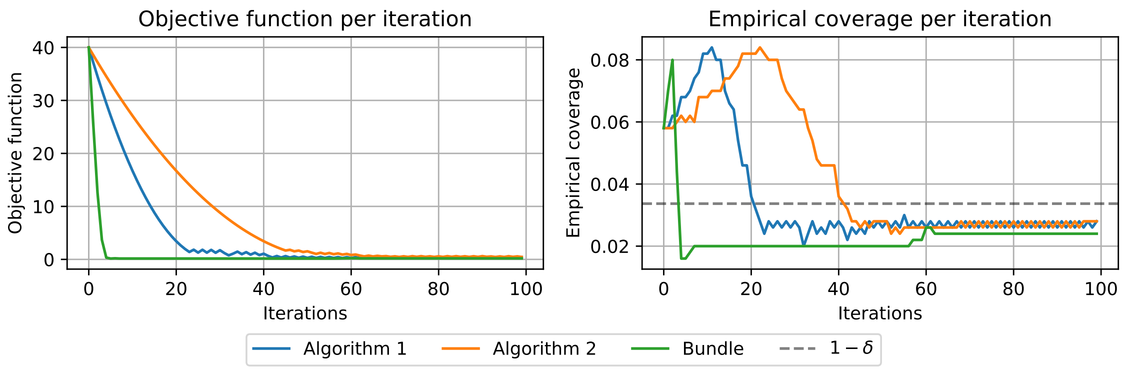

Figure 4: Convergence of Algorithms 1 and 2, and the `Bundle algorithm`

Algorithm	**Objective function**	**Empirical coverage**
Algorithm 1	0.3273	0.0270
Algorithm 2	0.5245	0.0274
Bundle	0.1872	0.0240

Table 2: Algorithm performance: average of the last iterations

In figure 4.2.3, we plot the two-dimensional optimization trajectory of Algorithms 1 and 2, and the `Bundle algorithm`. The optimization trajectory represents the path taken by the decision variable across all iterations of the optimization process. We depict their final point by a triangle and represent the objective and chance function using a level plot.

We can see that the optimization trajectories and final points are similar for all three algorithms. Algorithm 1 and the `Bundle algorithm` have close final points while Algorithms 1 and 2 have the same trajectory. It is important to note that none of the solutions satisfy the feasibility conditions.

4.2.2 Example 2.2

We slightly change the parameters of the original problem to further test and compare the different algorithms. The objective function is now

$$f(x) = \frac{1}{2}(x-a)^T Q(x-a) \text{ with } a = \begin{pmatrix} -2 \\ -3 \end{pmatrix}, Q = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix} \tag{26}$$

and is still quadratic with $Q \succeq 0$. The chance function is

$$\begin{aligned} g(x,Z) = Z^T W(x) Z + w^T Z \quad \text{with } W(x) &= \begin{pmatrix} (x_1-2)^2+1 & 0 \\ 0 & |x_2+1|^3 - 0.4 \end{pmatrix} \\ w &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ Z \sim \mathcal{N}(\mu, \Sigma) \quad \text{with } \mu &= \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma = \begin{pmatrix} 20 & 0 \\ 0 & 20 \end{pmatrix}. \end{aligned} \tag{27}$$

$1-\delta$ remains unchanged.

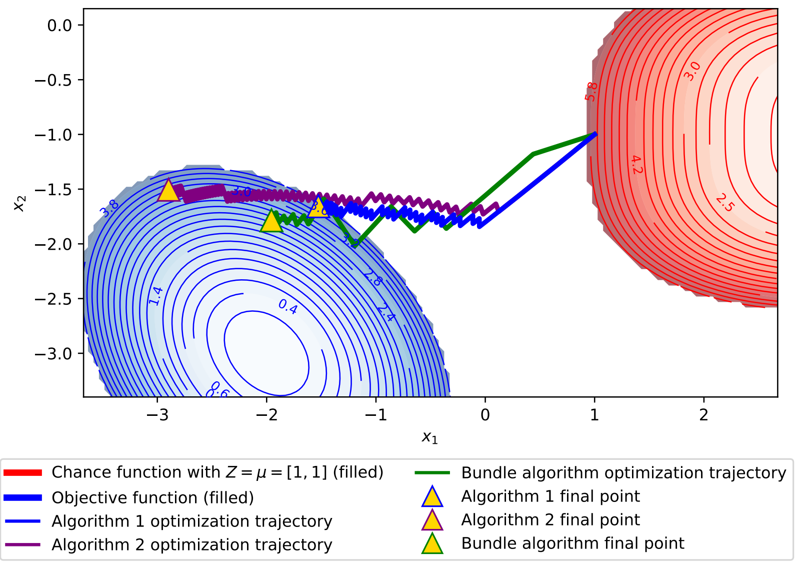

Figure 5: Optimization trajectories of Algorithms 1 and 2, and the `Bundle algorithm`, starting from $[1, -1]$

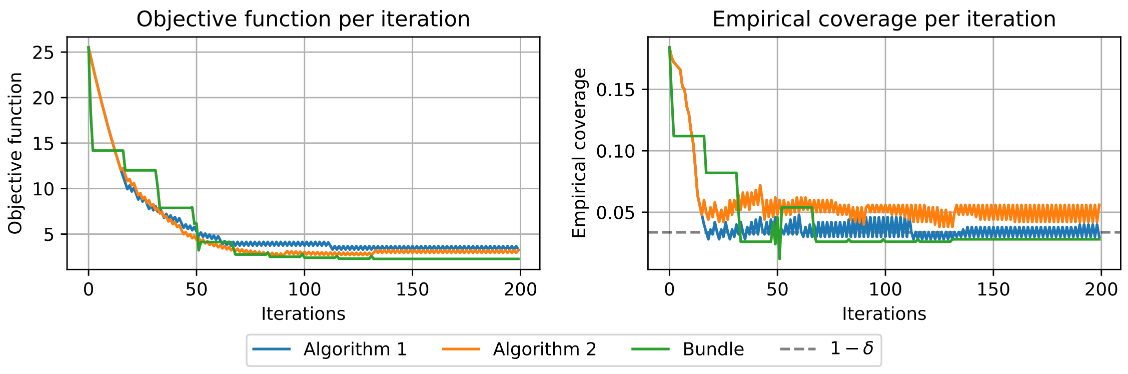

Figure 6: Convergence of Algorithms 1 and 2, and the `Bundle algorithm`

Algorithm	**Objective function**	**Empirical coverage**
Algorithm 1	3.4888	0.0350
Algorithm 2	3.0806	0.0490
Bundle	2.2702	0.0280

Table 3: Algorithm performance: average of the last iterations

In this new problem, we now notice that the optimization trajectories are initially similar but differ in their final point. The trajectory of Algorithm 2 stands out the most from the other two. This time, Algorithms 1 and 2 yield feasible solutions whereas the *Bundle algorithm*'s solution lies just below the $1 - \delta$ threshold.

4.2.3 Example 2.3

Again, we modify the parameters of the original problem. The objective function is now

$$f(x) = \frac{1}{2}(x-a)^T Q(x-a) \text{ with } a = \begin{pmatrix} 1 \\ -1 \end{pmatrix}, Q = \begin{pmatrix} 3 & 2 \\ 2 & 3 \end{pmatrix} \tag{28}$$

and is still quadratic with $Q \succeq 0$. The chance function is

$$\begin{aligned} g(x, Z) = Z^T W(x) Z + w^T Z \quad \text{with } W(x) &= \begin{pmatrix} (x_1+2)^2 & 0 \\ 0 & |x_2-3|^3 \end{pmatrix} \\ w &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ Z \sim \mathcal{N}(\mu, \Sigma) \quad \text{with } \mu &= \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma = \begin{pmatrix} 20 & 0 \\ 0 & 20 \end{pmatrix}. \end{aligned} \tag{29}$$

$1-\delta$ remains unchanged.

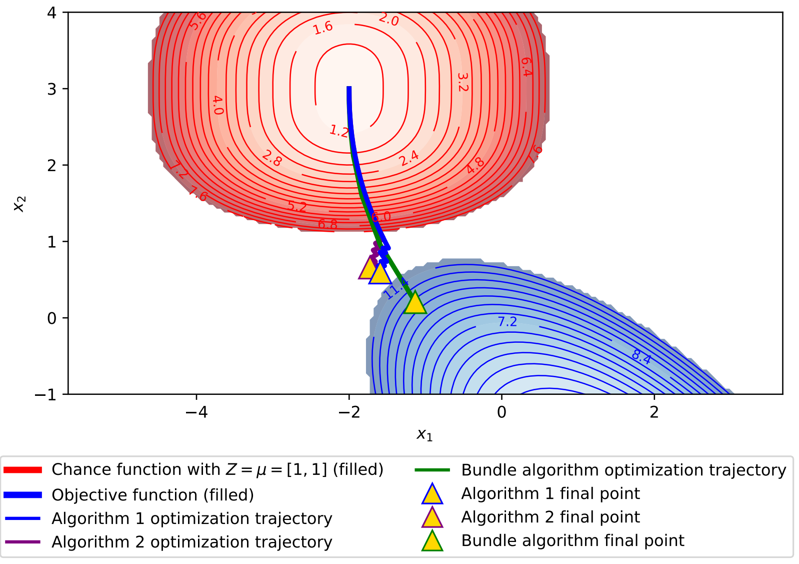

Figure 7: Optimization trajectories of Algorithms 1 and 2, and the `Bundle algorithm`, starting from $[-2, 3]$

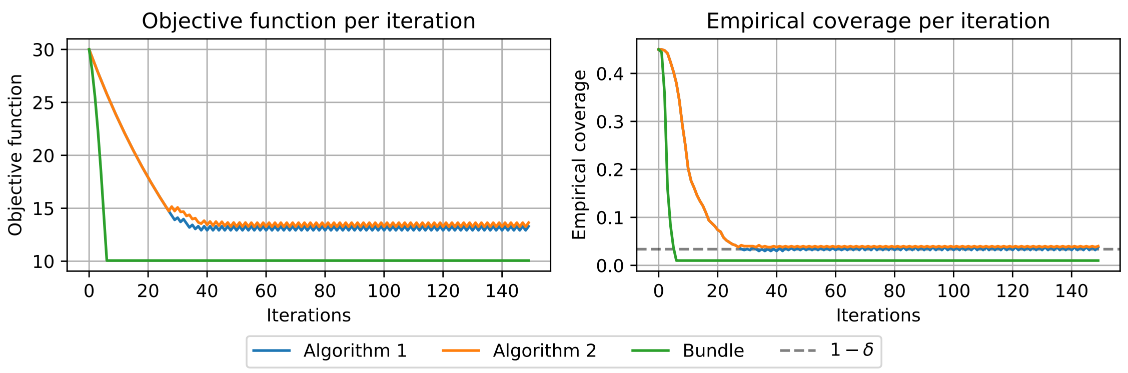

Figure 8: Convergence of Algorithms 1 and 2 and the `Bundle algorithm`

Algorithm	**Objective function**	**Empirical coverage**
Algorithm 1	13.1069	0.0350
Algorithm 2	13.4495	0.0390
Bundle	10.0605	0.0100

Table 4: Algorithm performance: average of the last iterations

Finally, the results are similar and the observations are consistent with those on the previous problem. The empirical coverage of Algorithms 1 and 2 lie close to the $1-\delta$ threshold whereas the `Bundle algorithm`'s solution is unfeasible.

4.3 Example 3

In this uni-variate problem from [Zha+24], our objective function and chance constraint are

$$f(x) = x^3 e^x \qquad g(x, Z) = 50Ze^x - 5 \quad \text{with } Z \sim \text{Exp}\left(\frac{1}{3}\right). \tag{30}$$

The problem also has deterministic constraint: $x^3 + 20 \leq 0 \iff x \leq (-20)^{1/3}$, which we verify is satisfied but do not take into account in our gradients. Through analytical derivation, we obtain the following optimal solution:

$$x^* = -\log(10) - \log\{F_Z^{-1}(1-\delta)\}. \tag{31}$$

For this problem, we have $1 - \delta = 0.9$.

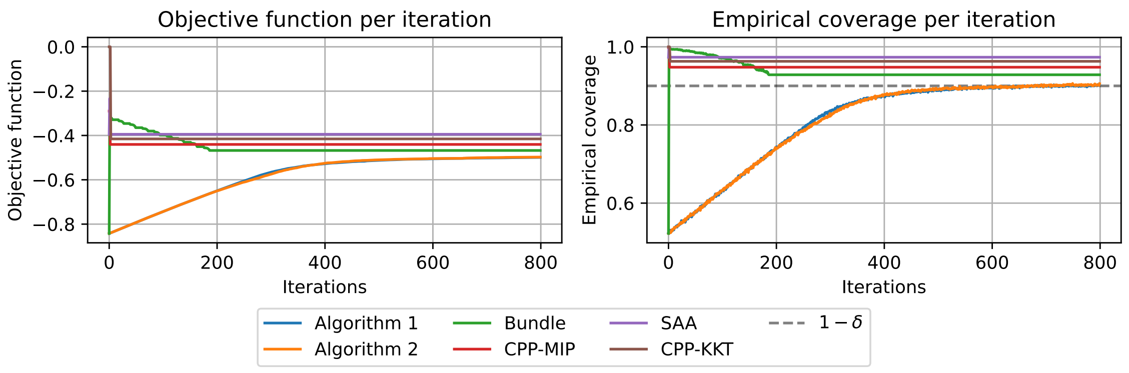

Figure 9: Convergence of all algorithms

Algorithm	**Objective function**	**Empirical coverage**
Algorithm 1	−0.499	0.9020
Algorithm 2	−0.4978	0.9030
Bundle	−0.4677	0.9284
CPP-KKT	−0.4155	0.9624
CPP-MIP	−0.4404	0.9477
SAA	−0.3947	0.9733

Table 5: Algorithm performance: average of the last iterations

Interestingly, we see that Algorithms 1 and 2, and the `Bundle algorithm`, while converging in many more steps, outperform all algorithms from [Zha+24]. All solutions are feasible.

4.4 Example 4

This multivariate resource-allocation problem from [Zha+24], our objective function and chance constraint are

$$f(x) = c \cdot x \qquad g(x, Z) = Z - Ax \quad \text{with } Z \sim \text{lognormal}\left(0, \frac{1}{4}\right) \times 3 \in \mathbb{R}^3$$
$$c = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \quad A = \begin{pmatrix} 3 & 12 & 2 \\ 10 & 3 & 5 \\ 5 & 3 & 15 \end{pmatrix}. \tag{32}$$

We notice that unlike previous problems, the output of our chance function is no longer a scalar but a vector of size s. We call them *Joint Chance Constrained Optimization* (JCCO) problems and can define them as

$$\min_{x \in \mathcal{X}} f(x) \text{ s.t. } \mathbb{P}\{g_i(x, Z) \leq 0, \forall i \in \{1, \dots, s\}\} \geq 1 - \delta. \tag{33}$$

Intuitively, this means that we want the constraint to be fulfilled for each element of the vector. Hence, we need to define new methods to solve JCCO problems. The problem's paper has two propositions: *Union Bounding* and *Pointwise Maximum*. We propose two similar methods. For the first one we sum the constraint vector i.e.

$$g_{\text{sum}}(x, Z) := \sum_{i=1}^{s} g_i(x, Z). \tag{34}$$

For the second, we only keep the maximum element of this vector, meaning that we want the largest element from the vector to satisfy the constraint, i.e.

$$g_{\max}(x, Z) := \max_{i \in \{1, \dots, s\}} g_i(x, Z). \tag{35}$$

For both of these methods, we also define a new δ:

$$\delta_{\text{sum,max}} := \delta / s^2. \tag{36}$$

For this problem, we have $\delta = 0.1$ and $s = 3$, which gives $1 - \delta_{\text{sum,max}} \approx 0.989$, the highest safety probability level we have tested.

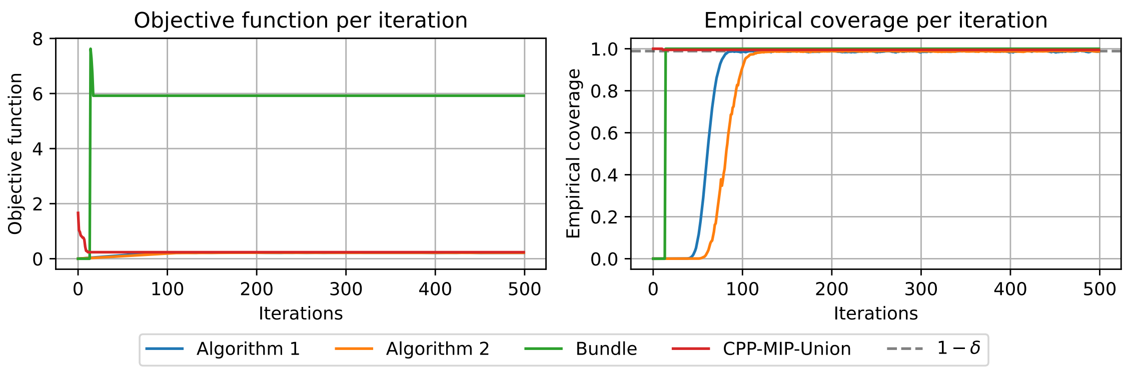

Figure 10: Convergence of Algorithms 1, 2, the `Bundle algorithm` and `CPP-MIP` using the `sum` method

Algorithm	Objective function	Empirical coverage
Algorithm 1	0.2141	0.9876
Algorithm 2	0.217	0.9881
Bundle	5.9203	1.0000
CPP-MIP-Union	0.237	0.9948

Table 6: Algorithm performance: average of the last iterations

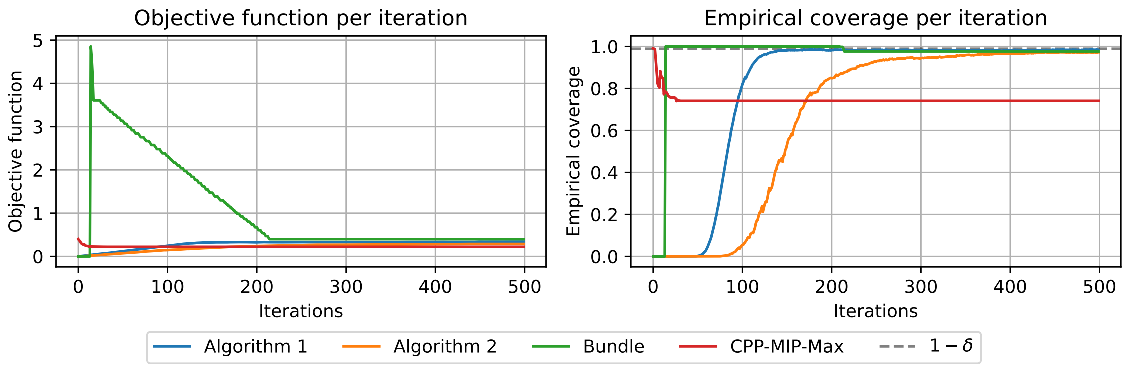

Figure 11: Convergence of Algorithms 1, 2, the `Bundle algorithm` and `CPP-MIP` using the `max` method

Algorithm	Objective function	Empirical coverage
Algorithm 1	0.3413	0.9845
Algorithm 2	0.2872	0.9716
Bundle	0.3991	0.9764
CPP-MIP-Max	0.221	0.7409

Table 7: Algorithm performance: average of the last iterations

We observe that Algorithms 1, 2 and the `Bundle algorithm` require more iterations to converge under the `max` method than under the `sum` method. These first two converge to an optimal feasible solution whereas the other algorithms yield either sub-optimal or unfeasible results. The `Bundle algorithm` works best under the `max` method and inversely `CPP-Union` seems to work better than `CPP-Max`.

5 Conclusion

We proposed a bi-level reformulation of the chance constrained optimization problem and relaxed it into an unconstrained formulation as given in (14). Based on this relaxation, we developed both a *first-order method* and a *zeroth-order method* for solving the problem. These methods were implemented and tested on five distinct problems, encompassing both uni-dimensional and multi-dimensional cases. Our experiments indicate that the two methods exhibit similar behavior in terms of solution quality. Although both methods have longer run-times compared to other evaluated algorithms, they consistently yield nearly feasible and competitive solutions in different scenarios. Notably, the *zeroth-order method*, despite its higher computational cost, is particularly valuable for problems with non-differentiable objective functions.

6 References

[Cho+17] Yinlam Chow et al. *Risk-Constrained Reinforcement Learning with Percentile Risk Criteria.* 2017. arXiv: `1512.01629 [cs.AI]`. URL: `https://arxiv.org/abs/1512.01629`.

[GX19] Xinbo Geng and Le Xie. *Data-driven Decision Making with Probabilistic Guarantees (Part 2): Applications of Chance-constrained Optimization in Power Systems.* 2019. arXiv: `1904.06755 [math.OC]`. URL: `https://arxiv.org/abs/1904.06755`.

[Hab96] Shelby J. Haberman. "Quantiles". In: *Advanced Statistics: Description of Populations.* New York, NY: Springer New York, 1996, pp. 367–404. ISBN: 978-1-4757-4417-0. DOI: `10.1007/978-1-4757-4417-0_7`. URL: `https://doi.org/10.1007/978-1-4757-4417-0_7`.

[KHVR20] Willem K. Klein Haneveld, Maarten H. van der Vlerk, and Ward Romeijnders. "Chance Constraints". In: *Stochastic Programming: Modeling Decision Problems Under Uncertainty.* Cham: Springer International Publishing, 2020, pp. 115–138. ISBN: 978-3-030-29219-5. DOI: `10.1007/978-3-030-29219-5_5`. URL: `https://doi.org/10.1007/978-3-030-29219-5_5`.

[LMA21] Yassine Laguel, Jérôme Malick, and Wim Ackooij. *Chance constrained problems: a bilevel convex optimization perspective.* 2021. arXiv: `2103.10832 [math.OC]`. URL: `https://arxiv.org/abs/2103.10832`.

[Med98] E A Medova. "Chance-constrained stochastic programming forintegrated services network management". In: *Annals of Operations Research* 81.0 (June 1998), pp. 213–230.

[NS17] Yurii Nesterov and Vladimir Spokoiny. "Random Gradient-Free Minimization of Convex Functions". In: *Foundations of Computational Mathematics* 17.2 (2017), pp. 527–566. ISSN: 1615-3383. DOI: `10.1007/s10208-015-9296-2`. URL: `https://doi.org/10.1007/s10208-015-9296-2`.

[ROC70] R. TYRRELL ROCKAFELLAR. "Convex Functions". In: *Convex Analysis.* Princeton University Press, 1970, pp. 23–31. ISBN: 9780691015866. URL: `http://www.jstor.org/stable/j.ctt14bs1ff.8` (visited on 04/12/2025).

[Zha+20] Jingzhao Zhang et al. *Why gradient clipping accelerates training: A theoretical justification for adaptivity.* 2020. arXiv: `1905.11881 [math.OC]`. URL: `https://arxiv.org/abs/1905.11881`.

[Zha+24] Yiqi Zhao et al. *Conformal Predictive Programming for Chance Constrained Optimization.* 2024. arXiv: `2402.07407 [eess.SY]`. URL: `https://arxiv.org/abs/2402.07407`.

A Experimental settings

All experiments were conducted with our own code in Python on a 2019 laptop. We coded all algorithms without using any external library except from `NumPy` to easily manipulate arrays and matrices and `SciPy` for its inverse CDF functions. 1 and 2 running-times ranged from a few seconds for the simplest problems, up to an hour for high-dimensional problems, especially when using *4-Point estimation* (Section 3.2.2).

When comparing the algorithms, we ensure consistency by using closely matched parameters where applicable, while carefully tuning the remaining parameters for optimal performance. However, fundamental differences in the solvers' operation impose limitations. For instance, algorithms based on Conformal Predictive Programming [Zha+24] lack the concept of an initial point.

Let us introduce the parameters for each problem, using our solver:

`initial_x` is our initial value for the decision variable. `T` is the maximum number of iterations for our gradient descent algorithm used to minimize our G_δ function, i.e. to estimate s^*. It can stop early if it has converged to a solution before the end of all `T` iterations. `lr_GD1` is the corresponding learning rate parameter. `I` is the number of samples from $Z \sim P_Z$, used to approximate our chance function in our G_δ function (8). `lr_GD2` is the learning rate parameter for our main gradient descent algorithm, which updates our decision variable. `delta` is our δ safety probability level. `mu` is our μ-regularization term in our F function (14). `theta` is a scalar that represents the interval used to smooth the $\max(x, 0)$ function, from G_δ (8), into a C^2 function that we call h_θ. `update_clipping` is our update clipping parameter (Section 3.2.4) (used for both *first-order* and *zeroth-order* methods). We present the parameters that we used for each numerical experiment:

```
"Example-1": (
        initial_x=0.1, T=300, I=500, delta=1-0.95,
        lr_GD1=1e-2, lr_GD2=2e-4, theta=1e-2, mu=1e-2
    ),
"Example-2.1": (
        initial_x=[0,0], T=2000, I=500, delta=1-0.03368421, lr_GD1=1e0,
        lr_GD2=1e-2, mu=1e-3, theta= 1e-2, update_clipping=3
    ),
"Example-2.2": (
        initial_x=[1,-1], T=2000, I=500, delta=1-0.03368421, lr_GD1=3e0,
        lr_GD2=1.5e-1, mu=1e-3, theta= 1e-2, update_clipping=1/2
    ),
"Example-2.3": (
        initial_x=[-2,3], T=2000, I=500, delta=1-0.03368421, lr_GD1=3e0,
        lr_GD2=1.5e-1, mu=1e-3, theta= 1e-2, update_clipping=1/2
    ),
"Example-3": (
        initial_x=-5, T=500, I=500, delta = 0.1, lr_GD1=1e-1,
        lr_GD2=3e-4, theta=5e-2, mu=2, update_clipping=10
    ),
"Example-4": (
        initial_x=[3,3], T=500, I=500, delta=0.1, lr_GD1=1e-1,
        lr_GD2=1e-2, theta=1e-2, mu=1, update_clipping=3
    ),
"Example-5": (
        initial_x=[0,0,0], T=500, I=500, delta=0.1/(3**2), lr_GD1=1e-1,
        lr_GD2=3e-4, theta=1e-2, mu=2, update_clipping=5
    )
```

B Optimal solutions analytical derivation

B.1 Example 1 (Section 4.1)

As explained in the problem's section, we can derive the analytical optimal point for this problem by computing $\{x : xZ - 1 \leq 0\}$ and picking x^* from this set that minimizes f:

$$\begin{gathered}\mathbb{P}(xZ - 1 \leq 0) \geq 1 - \delta \iff \mathbb{P}\left(Z \leq \frac{1}{x}\right) = \Phi\left(\frac{1}{x} - 1\right) \geq 1 - \delta \\ x \leq \frac{1}{\Phi^{-1}(1-\delta) + 1} \quad \text{where } \Phi \text{ is the CDF of } \mathcal{N}(0,1)\end{gathered} \tag{37}$$

With $\delta = 0.05$, we have $x \leq 0.378092$. We know that f is minimized at $x = 2$, therefore we have to pick the highest value that satisfies the constraint. Hence, we have $x^* = \{\Phi^{-1}(1-\delta) + 1\}^{-1}$.

B.2 Example 3 (Section 4.3)

$$\begin{gathered}\mathbb{P}(50Ze^x - 5 \leq 0) = \mathbb{P}\left(Z \leq \frac{1}{10}e^{-x}\right) = F_Z\left(\frac{1}{10}e^{-x}\right) \geq 1 - \delta \\ x \leq -\log(10) - \log\{F_Z^{-1}(1-\delta)\}\end{gathered} \tag{38}$$

f is minimized at $x = -3$ and is decreasing on $]-\infty, -3]$. Anything above $(-20)^{1/3} \approx -2.714$ is also rejected by the deterministic constraint. Therefore we have $x^* = -\log(10) - \log\{F_Z^{-1}(1-\delta)\}$.

C Zoom on the final points of optimization trajectories

C.1 Example 2.1 (Section 4.2.1)

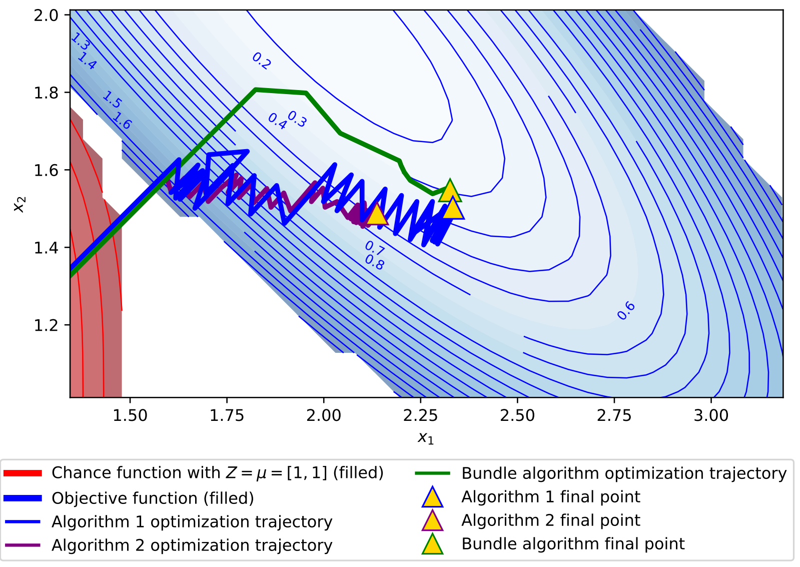

Figure 12: Zoom on the final points of optimization trajectories for 1, 2 and the `Bundle algorithm`

C.2 Example 2.2 (Section 4.2.2)

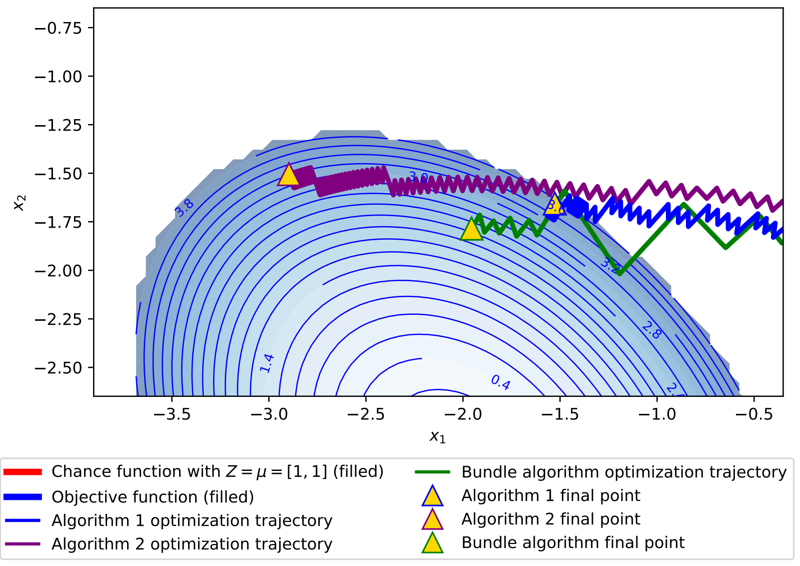

Figure 13: Zoom on the final points of optimization trajectories for 1, 2 and the `Bundle algorithm`

C.3 Example 2.3 (Section 4.2.3)

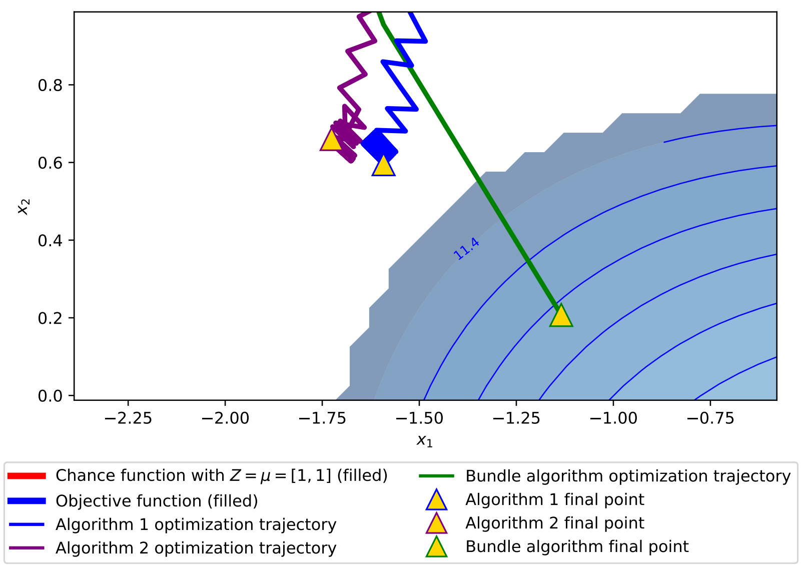

Figure 14: Zoom on the final points of optimization trajectories for 1, 2 and the `Bundle algorithm`