

CEopt: A MATLAB Package for Non-convex Optimization with the Cross-Entropy Method

Americo Cunha Jr
Rio de Janeiro State University

Marcos Vinicius Issa
Rio de Janeiro State University

Julio Cesar Basilio
Rio de Janeiro State University

José Geraldo Telles Ribeiro
Rio de Janeiro State University

Abstract

This paper introduces **CEopt** (<https://ceopt.org>), a MATLAB tool leveraging the Cross-Entropy method for non-convex optimization. Due to the relative simplicity of the algorithm, it provides a kind of transparent “gray-box” optimization solver, with intuitive control parameters. Unique in its approach, **CEopt** effectively handles both equality and inequality constraints using an augmented Lagrangian method, offering robustness and scalability for moderately sized complex problems. Through select case studies, the package’s applicability and effectiveness in various optimization scenarios are showcased, marking **CEopt** as a practical addition to optimization research and application toolsets.

Keywords: Non-convex optimization, Cross-entropy method, MATLAB optimization tool.

1. Introduction

Optimization underpins numerous advances across many disciplines like machine learning, control systems, and scientific computing, aiming universally to maximize outcomes or minimize costs under given constraints (Bonnans, Gilbert, Lemarechal, and Sagastizábal 2009). Challenges in this domain fall into two primary categories: convex and non-convex. While convex optimization benefits from a more straightforward approach to finding global solutions, non-convex optimization introduces significant challenges. The last category is characterized by a landscape peppered with local optima and abrupt discontinuities, complicating the search for global optima and often necessitating more advanced, nuanced optimization strategies to effectively navigate these complexities (Nocedal and Wright 2006).

1.1. Convexity and beyond in optimization

Convex optimization, characterized by the convexity of the objective function and constraints, holds a privileged position in the optimization landscape. When an optimization problem can be elegantly cast into a convex formulation, the path to a solution is often smooth and well-paved (Boyd and Vandenberghe 2004; Bertsekas 2009). In this sense, the linear programming, quadratic programming, and semi-definite programming paradigms have revolutionized countless applications like: control theory (Oliveira and Krstic 2022); image reconstruction (Vijina

and Jayasree 2020); statistical learning (Hastie, Tibshirani, and Friedman 2016); etc.

However, the reality of optimization problems is not always so conveniently convex. Many real-world challenges defy the assumptions of convexity (Bertsekas 2016). Consider, for instance, the vast domains of data science and machine learning (Kroese, Botev, Taimre, and Vaisman 2019; Brunton and Kutz 2022); inverse problems (Kaipio and Somersalo 2004; Tarantola 2005; Quaranta, Lacarbonara, and Masri 2020); structural optimization and mechanical design (Gonçalves, Lopez, and Miguel 2015; Wolszczak, Lonkwic, Cunha Jr, Litak, and Molski 2019); simulation-based optimization (Cunha Jr, Soize, and Sampaio 2015; Cunha Jr 2021), uncertainty quantification (Tenorio 2017; Soize 2017) and beyond. These domains teem with complex, multidimensional landscapes where non-convexity reigns supreme. The formulation of these problems often strays from convexity, leading to scenarios where the journey toward the optimal solution becomes treacherous.

Non-convex optimization introduces a myriad of complexities. The landscape becomes speckled with numerous local optima, leaving traditional gradient-based methods vulnerable to converging at suboptimal solutions. Moreover, non-differentiability, discontinuities, and other irregularities further obstruct the optimization path. In the face of such formidable challenges, the choice of an appropriate solver becomes of paramount importance (Bonnans *et al.* 2009).

To offer readers an overview of optimization solvers available for practical and general purpose use, we will now explore various software tools commonly referenced in literature and utilized across diverse computational platforms.

1.2. Solvers for optimization problems

Decades of research and development have cultivated a vibrant ecosystem of solvers for convex optimization (Bertsekas 2015), flourishing within the open-source community and beyond, offering seamless integration into widely used numerical packages and platforms:

- Python’s SciPy (Virtanen *et al.* 2020) is a fundamental library for scientific computing, providing a rich collection of optimization tools.
- Julia’s JuMP (Lubin, Dowson, Dias Garcia, Huchette, Legat, and Vielma 2023) offers a powerful modeling language for optimization problems.
- AMPL (Fourer 1996), a comprehensive and highly flexible modeling system for solving large-scale optimization problems.
- MATLAB’s Optimization Toolbox (MathWorks Inc. 2024b) provides a vast suite of algorithms for linear, nonlinear, and binary optimization.
- CVX (Grant and Boyd 2008, 2014), a MATLAB and Python tool for disciplined convex programming, streamlining the process of solving convex optimization problems.
- R’s ROI (Theußl, Schwendinger, and Hornik 2020; Hornik, Meyer, Schwendinger, and Theußl 2023) package integrates a wide range of optimization techniques, making them accessible to the R environment.
- PETSc for Fortran (Balay *et al.* 2023) brings robust numerical tools for parallel scientific computations, including optimization routines.

- GNU Scientific Library (GSL) ([Galassi et al. 2018](#)) caters to C/C++ programmers with a collection of mathematical routines, including optimization functions.
- NLOpt ([Johnson 2007](#)) provides a library for nonlinear optimization, supporting a multitude of programming languages.

For those tackling convex optimization challenges, this assortment of tools often represents the end of their search. These solvers enable efficient and reliable optimization processes, circumventing significant obstacles.

Conversely, non-convex optimization, with its intrinsic complexity, demands a broader spectrum of solutions ([Bazaraa, Sherali, and Shetty 2006](#)). This domain benefits from metaheuristic and evolutionary algorithms capable of overcoming the hurdles posed by multiple local optima and intricate solution spaces.

Software implementations for non-convex optimization include metaheuristic methods such as Genetic Algorithms (GA), Simulated Annealing (SA), and Particle Swarm Optimization (PSO). These methods, which can navigate through multimodal landscapes to seek global optima, are embodied in various platforms catering to different computational needs:

- MATLAB’s Global Optimization Toolbox ([MathWorks Inc. 2024a](#)) provides an integrated environment for applying GA, SA, and PSO, making it a powerful suite for users facing non-convex challenges.
- Python’s rich ecosystem, in addition to SciPy ([Virtanen et al. 2020](#)), also includes libraries like DEAP for evolutionary algorithms ([Fortin, De Rainville, Gardner, Parizeau, and Gagné 2012](#)) or PySwarms for particle swarm optimization ([Miranda 2018](#)).
- For R users, packages such as GA ([Scrucca 2013](#)) and PSO ([Bendtsen 2022](#)) offer tools to implement genetic and particle swarm optimization techniques, respectively, within the R programming environment, enabling statisticians and data scientists to incorporate sophisticated optimization routines in their analyses.

Despite the utility of these metaheuristic methods in exploring vast and complex solution spaces, they often operate as “black-box” solvers. This means that while they are capable of finding solutions to difficult optimization problems, the internal workings and decision processes of these algorithms often is opaque, making it challenging for users to interpret the optimization process or to adjust strategies with precision.

This backdrop of existing non-convex optimization software sets the stage for the introduction of the Cross-Entropy (CE) method and its MATLAB implementation dubbed **CEopt**. The subsequent sections will explore the theoretical underpinnings of the CE method, the architecture and features of **CEopt**, and its practical application across various domains, highlighting its potential to fill a significant gap in the non-convex optimization landscape.

1.3. The Cross-Entropy method

The CE method embodies a systematic and lucid strategy for optimization, grounded in the robust principles of probability theory ([Rubinstein and Kroese 2004](#)). This method transforms deterministic optimization challenges into stochastic problems centered on the estimation of

rare event probabilities, as represented in Figure 1. Through the technique of adaptive importance sampling, CE meticulously refines its pursuit of the global optimum across iterations. Such an approach not only deepens our comprehension of the intricate optimization terrain but also guarantees a focused and efficient exploration process. The probabilistic underpinnings of CE provide a flexible framework capable of addressing a broad spectrum of optimization dilemmas. From continuous to combinatorial and mixed problems, CE delineates a clear and methodical pathway to uncovering solutions, demonstrating its adaptability and broad applicability (Rubinstein 2005; Kroese, Porotsky, and Rubinstein 2006).

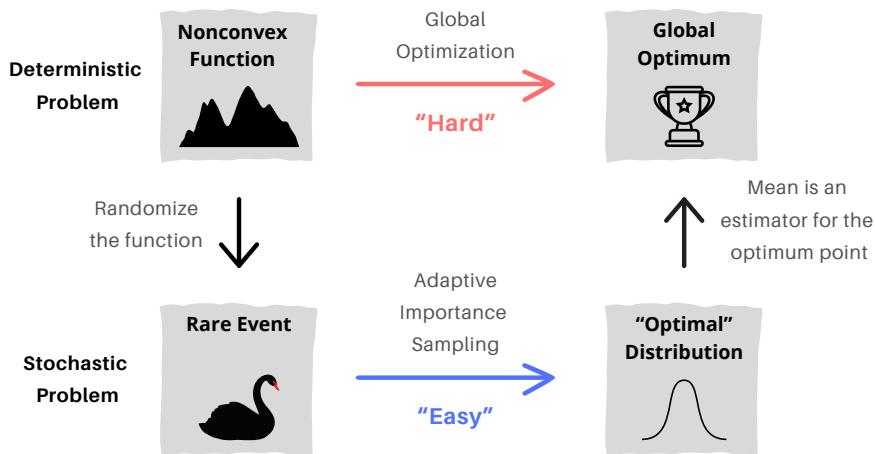


Figure 1: Cross-Entropy Method Explained: This method turns a complex optimization problem into a manageable task of estimating a rare event. By using adaptive importance sampling, it iteratively focuses on making the global optimum more likely to find. The mean of the “optimal” distribution estimates the best solution, showcasing how CE simplifies tough optimization challenges into more approachable ones with a probabilistic strategy.

Contrasting with the often opaque “black-box” optimization methods, CE boasts simplicity and transparency. It operates with a minimal set of control parameters, each serving a clear and defined purpose. These include the number of samples per iteration, the size of an elite subset of samples for learning, the maximum iteration count, convergence tolerances, and smoothing parameters to enhance the optimization process. The CE method’s working mechanism, which is reviewed in this paper, is both elegant and understandable.

Imagine optimization as a journey through a vast landscape, with peaks and valleys representing objective function values. In this context, CE operates as a skilled navigator, charting a course to reach the optimal summit. CE begins by sampling the objective function domain according to a carefully chosen probability distribution, akin to casting a net over the landscape to catch points. Among the sampled points, CE identifies the ones that yield better results (those with lower objective function values). These elite points form the basis for learning and improvement. CE then engages in a stochastic learning process, striving to understand the landscape by updating its probability distribution.

In the CE method, the probability distribution plays a dual role, with its mean representing the method’s current estimate of the optimal point. As the optimization journey unfolds,

this mean undergoes a gradual shift in the direction of the global optimum, akin to a climber closing in on a summit. Simultaneously, the distribution's variance undergoes reduction, symbolizing the method's growing confidence in the vicinity where the optimal solution likely resides. This decrease in variance effectively narrows the method's exploration towards the true optimal location. In the ideal limit, the probability distribution converges to a delta function precisely centered at the global optimum.

The learning process of CE hinges on the dynamic adjustment of two pivotal parameters: the mean and the variance. These crucial updates draw wisdom from the elite samples, which represent the points that have consistently exhibited superior performance during the optimization process. The mean's adaptation aligns it more closely with the direction of the global optimum, resembling a navigator fine-tuning their course. Meanwhile, the variance diminishes in response to the method's heightened confidence, particularly in the vicinity surrounding the presumed optimal solution. This intricate interplay, finely tuning both the mean and variance, stands as a cornerstone of CE's effectiveness, guaranteeing its progressive convergence towards the global optimum across successive iterations.

1.4. Paper contribution and organization

In this paper, we introduce **CEopt** (<https://ceopt.org>), a MATLAB package developed to leverage the Cross-Entropy (CE) method for continuous non-convex optimization challenges. We detail the architecture and software engineering principles that underpin **CEopt**, designed with the goal of ensuring easy integration into diverse optimization workflows.

Through a variety of case studies, from simple to more complex scenarios, we demonstrate the utility of **CEopt**. Whether for refining machine learning models, enhancing structural optimization, or venturing into new research areas, **CEopt** provides a valuable tool for researchers and practitioners navigating the intricate landscapes of non-convex optimization.

Due to the solid probabilistic foundations and relative simplicity of CE method algorithm, **CEopt** represents a step forward in optimization software by offering a kind of “gray-box” MATLAB solver. The control parameters of the method transparent and user-friendly, addressing a frequent issue encountered with more opaque “black-box” solutions. Additionally, **CEopt** features a refined framework for addressing both equality and inequality constraints through an augmented Lagrangian method. While this provides robustness against problems laden with constraints, it is important to recognize that **CEopt**, like any tool, is not a panacea for any optimization problem, specially in large dimensions. It is, however, a significant aid in the toolkit of those tackling non-convex optimization problems, offering clarity and control where such qualities are often sought but not always found.

This work contributes to the non-convex optimization field by presenting **CEopt** as a pragmatic and accessible tool, enhancing the ability of users to engage with and solve complex optimization problems. While **CEopt** is a notable addition to the optimization toolbox, we acknowledge its role as part of a broader ensemble of methods necessary to address the wide array of challenges presented by non-convex optimization scenarios.

The remainder of this paper is organized as follows:

- **Historical development of the CE method:** A journey through the evolution and milestones of the CE method, setting the stage for its current applications.
- **Theoretical foundation of the CE method:** An exploration of the probabilistic principles and mechanisms that constitute the backbone of the CE method.
- **MATLAB Implementation: The CExpt package:** A walkthrough into the code architecture, highlighting its modular design, user-friendliness, and integration capabilities.
- **Numerical experiments:** A compilation of empirical studies demonstrating **CExpt**'s versatility and power across a spectrum of optimization scenarios.
- **Final remarks:** Concluding observations and reflections on **CExpt**'s contribution to the field of optimization, along with a glimpse into potential expansions of the code.

2. Historical development of the CE method

The Cross-Entropy (CE) method has evolved from its humble beginnings in rare event estimation to becoming a versatile tool for non-convex optimization. This section explores the historical development of CE, tracing its journey from its pioneering work in rare event estimation by Reuven Rubinstein¹ to its prominent role in modern applications, theoretical advancements, and software implementations. A timeline illustrating key milestones in the CE method's evolution can be found in Figure 2.

The roots of the CE method can be traced back to the pioneering work of Rubinstein (Rubinstein 1997) about rare event estimation problems in the context of Monte Carlo simulations. Here, the problem is addressed using an adaptive importance sampling algorithm, allowing a significant speed-up on the estimation process.

A key turning point in the history of CE was the realization that an adaptive importance sampling strategy could be extended to tackle optimization problems (Rubinstein 1999). By introducing a controlled randomness in the design variables and minimizing the Kullback–Leibler divergence between a zero-variance distribution at the global optimum and the importance sampling distribution, CE could efficiently explore complex and non-convex landscapes to find high-quality optimal solutions.

This discovery opened the door to a wide range of applications in optimization problems, in several areas of knowledge, such as clustering and classification (Mannor, Peleg, and Rubinstein 2005; Kroese, Rubinstein, and Taimre 2007); multi-objective optimization (Bekker

¹Reuven Rubinstein (1938-2012) was an Israeli scientist renowned for his substantial contributions to Monte Carlo simulation, applied probability, stochastic modeling, and stochastic optimization. He authored over one hundred papers and six books during his distinguished career. Rubinstein was recognized as the founder of pioneering methods such as the score function method, stochastic counterpart method, and cross-entropy method, which have widespread applications in combinatorial optimization and simulation. In 2010, he received the INFORMS Simulation Society's Lifetime Professional Achievement Award and, in 2011, the Lifetime Professional Award from the Operations Research Society of Israel (ORSIS) in recognition of his fundamental contributions to the fields of simulation and operations research.

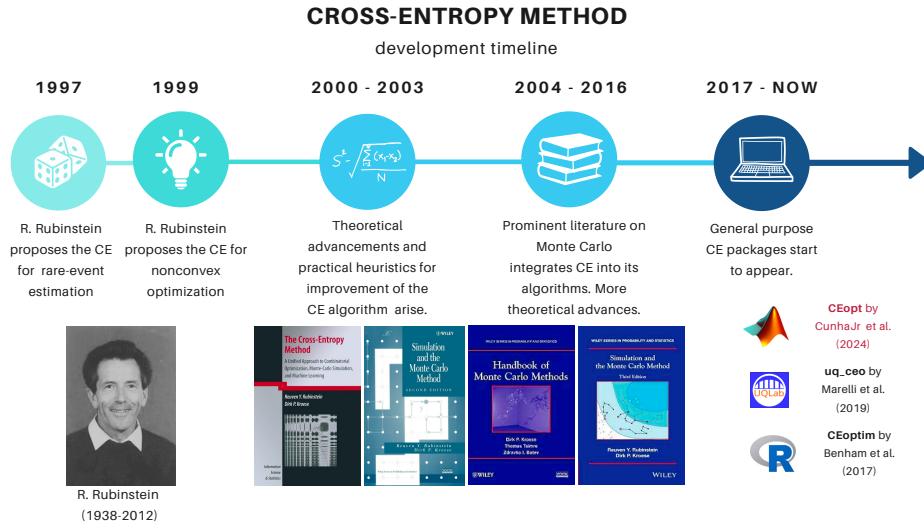


Figure 2: Cross-Entropy Method Timeline: Beginning with R. Rubinstein’s initial proposal in 1997 for rare-event estimation, this timeline tracks the evolution of the method into non-convex optimization and beyond. Highlighting key theoretical advancements and the development of CE software, it illustrates the method’s increasing significance in the field of optimization.

and Aldrich 2011; Haber, Beruvides, Quiza, and Hernandez 2017); vehicle routing problem (Chepuri and Homem-de Mello 2005); signal optimization (Maher, Liu, and Ngoduy 2013); structural design (Issa, Cunha Jr, Soeiro, and Pereira 2018, 2023; Farahmand-Tabar, Ashtari, and Babaei 2023); reliability assessment (Kroese and Hui 2007; Chan, Papaioannou, and Straub 2023); energy harvesting optimization (Cunha Jr 2021); portfolio optimization (Durán Fernández, Figueroa Mora, and Maldonado 2023); calibration of dynamical systems models for structural dynamics and finite element simulation (Dantas, Cunha Jr, and Silva 2019a; Dantas, Cunha Jr, Soeiro, Cayres, and Weber 2019b; Raqueti, Teloli, da Silva, Busseta, and Cunha Jr 2023). Additionally, during the COVID-19 pandemic, CE was employed to calibrate models used for predicting the spread of the virus (Cunha Jr, Barton, and Ritto 2023), aiding in decision-making processes.

The CE method’s journey was accompanied by the development of theoretical guarantees and insights, such as the convergence properties of CE for global optimization. Landmark theorems (Homem-de Mello and Rubinstein 2002; Asmussen, Kroese, and Rubinstein 2005; Margolin 2005; de Mello 2007) established its legitimacy as a robust optimization and rare event estimation technique. This theoretical foundation reassured practitioners of CE’s effectiveness in solving complex problems.

In 2004, the publication of the book *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning* by Rubinstein and Kroese (2004) marked a significant milestone in the CE method’s history. This comprehensive book provided a detailed account of the CE method’s principles and practical applications, solidifying its position as a powerful optimization technique.

In the years following its inception, the Cross-Entropy (CE) method gained widespread recognition and accessibility through influential publications such as (De Boer, Kroese, Mannor,

and Rubinstein 2005; Rubinstein and Kroese 2007, 2016; Kroese, Rubinstein, Cohen, Porotsky, and Taimre 2013; Botev, Kroese, Rubinstein, and L’Ecuyer 2013). Its significance is underscored by its inclusion in authoritative references like the *Monte Carlo Handbook* by Kroese, Taimre, and Botev (2011), solidifying its reputation as a valuable tool in the realm of Monte Carlo simulations.

The CE method’s practicality was greatly enhanced by its incorporation into software packages. Notable implementations include **CEoptim** in R (Benham, Duan, Kroese, and Liquet 2017), and the function *uq_ceo* in the UQLib (Moustapha, Lataniotis, Wiederkehr, Wagner, Wicaksono, Marelli, and Sudret 2022) module of UQlab package (Marelli and Sudret 2014), available in MATLAB and Python. However, these implementations do not support complex constraint handling. **CEopt** differentiates itself by including robust features for managing both equality and inequality nonlinear constraints, making it particularly useful for engineering and control systems where such constraints are critical. Additionally, **CEopt** offers advanced stopping criteria, expanding its applicability in sophisticated optimization tasks.

Today, the CE method keep finding applications in cutting-edge fields such as robotics (Xu, Fan, Ma, and Wang 2024) and power grids (Zhao, Chen, Liu, Cheng, Xie, Hu, and Wang 2024). Its ability to handle complex, non-convex optimization problems aligns well with the challenges posed by modern machine learning models, where finding optimal hyperparameters and model architectures is essential.

3. Theoretical foundation of the CE method

The Cross-Entropy method provides a robust framework for comprehending its operational principles and effectiveness. At its essence, CE harnesses the concept of importance sampling, a fundamental technique in probability theory and Monte Carlo methods. This concept assumes a pivotal role in CE’s capacity to efficiently explore and converge towards optimal solutions. In this section, we present a comprehensive overview of the CE method’s theory, elucidating how it employs an adaptive importance sampling strategy to yield high-quality solutions for non-convex optimization challenges.

3.1. Deterministic optimization problem

The primary problem of interest here is to find an optimal value that minimizes a certain scalar function. This problem can be compactly represented as

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathcal{X}} F(\mathbf{x}), \quad (1)$$

where the (cost) objective function $\mathbf{x} \in \mathcal{X} \mapsto F(\mathbf{x}) \in \mathbb{R}$ depends nonlinearly on the design variable vector $\mathbf{x} = (x_1, \dots, x_n)$. The latter is defined within an admissible set $\mathcal{X} \subset \mathbb{R}^n$ that describes the constraints to be satisfied by its components. The common instances in which we deal with this problem are:

$$\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x}_l \preceq \mathbf{x} \preceq \mathbf{x}_u\}, \quad (2)$$

$$\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x}_l \preceq \mathbf{x} \preceq \mathbf{x}_u, H_i(\mathbf{x}) = 0, i = 1, \dots, N_E\}, \quad (3)$$

or

$$\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x}_l \preceq \mathbf{x} \preceq \mathbf{x}_u, H_i(\mathbf{x}) = 0, i = 1, \dots, N_E, G_j(\mathbf{x}) \leq 0, j = 1, \dots, N_I\}. \quad (4)$$

In all cases, the components of \mathbf{x} have lower and upper bounds defined by \mathbf{x}_ℓ and \mathbf{x}_u , respectively, and the generalized inequality \preceq indicates that each component of the left vector is less than or equal to the respective component of the right vector. The first scenario also incorporates the simple case where $\mathcal{X} = \mathbb{R}^n$, by admitting the possibility of infinity bound limits. In the last two scenarios, it is also necessary to satisfy a set of (usually nonlinear) constraints, defined by the functions $\mathbf{x} \in \mathcal{X} \mapsto H_i(\mathbf{x}) \in \mathbb{R} (j = 1, \dots, N_E)$, in case of equality conditions, and by $\mathbf{x} \in \mathcal{X} \mapsto G_j(\mathbf{x}) \in \mathbb{R} (j = 1, \dots, N_I)$, for the inequality conditions.

For numerical implementation purposes, when constraints apply – admissible sets (3) and (4) – it is necessary to transform the problem (1) into an equivalent bounded unconstrained problem – admissible set (2) – seeking to minimize a modified function $\mathbf{x} \in \mathcal{X} \mapsto \tilde{F}(\mathbf{x}) \in \mathbb{R}$.

In the CE literature, various strategies have been proposed for defining the modified objective function $\tilde{F}(\mathbf{x})$, with penalization techniques commonly favored (Rubinstein and Kroese 2004). Nonetheless, due to its superior stability and enhanced ability to manage a broad spectrum of constraints, we opted for the augmented Lagrangian approach (Conn, Gould, and Toint 1991, 1997), which is formulated as

$$\tilde{F}(\mathbf{x}) = F(\mathbf{x}) + \sum_{i=1}^{N_E} \left(\lambda_i^E H_i(\mathbf{x}) + \frac{1}{2} \nu H_i(\mathbf{x})^2 \right) - \sum_{j=1}^{N_I} \lambda_j^I s_j \ln(s_j - G_j(\mathbf{x})), \quad (5)$$

where λ_i^E and λ_j^I are the non-negative Lagrange multiplier estimates for equality and inequality constraints, respectively; $\nu > 0$ acts as a penalty factor; and s_j , the non-negative shifts, are given by

$$s_j = \frac{\lambda_j^I}{\nu}. \quad (6)$$

This augmented Lagrangian transformation facilitates the handling of both equality and inequality constraints within an unconstrained optimization framework by substituting $F(\mathbf{x})$ with $\tilde{F}_k(\mathbf{x})$. The latter represents a sequentially indexed version of $\tilde{F}(\mathbf{x})$, derived from an incrementally adjusted penalty ν_k . In this iterative process, a solution \mathbf{x}_k is computed and used to update the Lagrange multipliers according to

$$\begin{aligned} \lambda_i^E &\leftarrow \lambda_i^E + \nu_k H_i(\mathbf{x}_k) \\ \lambda_j^I &\leftarrow \max(0, \lambda_j^I + \nu_k G_j(\mathbf{x}_k)), \end{aligned} \quad (7)$$

ensuring that the penalty factor is progressively increased, i.e., $\nu_{k+1} \geq \nu_k$. This systematic augmentation of the Lagrangian underscores a commitment to refining constraint handling, thereby enhancing the CE method's applicability to complex optimization scenarios.

3.2. Optimization through rare event estimation

The core concept behind the CE method is to link the non-convex optimization problem defined by (1) with a rare event estimation problem. By employing an optimized sampling strategy for the latter, it is possible to solve the former through stochastic calculations.

This stochastic reinterpretation of the optimization problem occurs within a probabilistic space denoted as $(\Omega, \Sigma, \mathcal{P})$, comprising a sample space Ω , a σ -field Σ , and a probability measure \mathcal{P} . Within this framework, any random vector $\mathbf{X} = (X_1, \dots, X_n)$ is characterized

by a distribution $P(d\mathbf{x})$ in \mathbb{R}^n , with a density $\mathbf{x} \mapsto f(\mathbf{x}, \cdot)$ with respect to $d\mathbf{x}$. We denote the expected value of \mathbf{X} concerning the density $f(\mathbf{x}, \cdot)$ as $\mathbb{E}_f\{\mathbf{X}\}$, and its variance as $\mathbb{V}_f\{\mathbf{X}\}$.

To introduce randomness into the problem (1), we transform the vector of design variables, \mathbf{x} , into a random vector, \mathbf{X} , governed by a probability distribution characterized by the density $f(\mathbf{x}, \mathbf{u})$. Here, \mathbf{u} represents a vector with carries the probability distribution hyperparameters. For instance, if the mean vector $\boldsymbol{\mu} \in \mathbb{R}^n$ and the standard deviation vector $\boldsymbol{\sigma} \in \mathbb{R}^n$ are the hyperparameters of $f(\mathbf{x}, \mathbf{u})$, then $\mathbf{u} = (\boldsymbol{\mu}, \boldsymbol{\sigma})$.

Denoting the global minimum of problem (1) as $\gamma^* = F(\mathbf{x}^*)$, we can note that the random event $F(\mathbf{X}) \leq \gamma^*$ is impossible since the objective function cannot yield values lower than γ^* . Consequently, $\mathcal{P}\{F(\mathbf{X}) \leq \gamma^*\} = 0$. If we relax the reference value to a scalar $\gamma > \gamma^*$, $F(\mathbf{X}) \leq \gamma$ typically signifies a rare event when γ is chosen to be close to γ^* , i.e.,

$$\mathcal{P}\{F(\mathbf{X}) \leq \gamma\} \approx 0 \text{ if } \gamma \approx \gamma^*. \quad (8)$$

Therefore, the rare event $F(\mathbf{X}) \leq \gamma$ occurring for $\gamma \approx \gamma^*$ is intrinsically linked to the optimization problem (1). It involves realizations (samples) of \mathbf{X} that either directly solve the optimization problem or are in close proximity to the optimal solution. Consequently, by estimating this rare probability through stochastic simulation, we theoretically obtain optimal or near-optimal values (high-quality solutions) for the optimization problem.

Utilizing the indicator function

$$\mathbb{1}_{\mathcal{A}}(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} \in \mathcal{A} \\ 0, & \text{if } \mathbf{x} \notin \mathcal{A}, \end{cases} \quad (9)$$

we can express the probability in (8) as

$$\mathcal{P}\{F(\mathbf{X}) \leq \gamma\} = \mathbb{E}_f\{\mathbb{1}_{F(\mathbf{X}) \leq \gamma}\}, \quad (10)$$

and a practical and unbiased method to estimate (10) is given by the sample mean estimator

$$\mathbb{E}_f\{\mathbb{1}_{F(\mathbf{X}) \leq \gamma}\} \approx \frac{1}{N_s} \sum_{k=0}^{N_s} \mathbb{1}_{F(\mathbf{X}_k) \leq \gamma}, \quad (11)$$

where \mathbf{X}_k ($k = 1, \dots, N_s$) represents i.i.d. statistical realizations of \mathbf{X} drawn from $f(\mathbf{x}, \mathbf{u})$. However, it is essential to note that using the sample mean (11) as an estimator for a rare event like (8) can be problematic, as it can be computationally expensive and often impractical in real-world applications. For instance, consider a scenario where one million of samples (which is common in rare events) are necessary, and each evaluation of $F(\mathbf{X})$ takes approximately 1 second. The total computational time needed for the estimation would be on the order of 10^6 seconds, or roughly 11.6 days. Even with parallelization strategies to distribute the workload across multiple processors or machines, completing such a computation in a reasonable amount of time can still be challenging for very small probabilities.

In contrast, adaptive importance sampling offers an alternative approach that can significantly reduce the computational burden by focusing computational effort on the most promising regions of the sample space. This approach is presented in the next section.

3.3. Rare event estimation using adaptive importance sampling

Rare event estimation using adaptive importance sampling (AIS) is a cornerstone in rare event simulation methodologies, particularly in the development of the Cross-Entropy method. The primary goal of AIS is to efficiently estimate the probability of rare events by designing a suitable importance sampling (IS) density that focuses computational effort on the regions of interest in the sample space. By allocating the majority of its mass near the rare event region, AIS optimizes the sampling process for improved accuracy and efficiency.

In this sense, consider then a generalization of the rare event probability (10) given by

$$\ell = \mathbb{E}_f \{ S(\mathbf{X}) \} = \int_{\mathbb{R}^n} S(\mathbf{x}) f(\mathbf{x}, \cdot) d\mathbf{x}, \quad (12)$$

where $S(\mathbf{X}) \geq 0$ denotes a certain performance function that depends on the random vector \mathbf{X} . For example, in the problem (10) we have $S(\mathbf{X}) = \mathbf{1}_{F(\mathbf{X}) \leq \gamma}$.

Aiming to reduce the computational cost of the estimation process of (12), we will consider a reformulation of the problem in the form

$$\ell = \int_{\mathbb{R}^n} S(\mathbf{x}) \frac{f(\mathbf{x}, \cdot)}{g(\mathbf{x}, \cdot)} g(\mathbf{x}, \cdot) d\mathbf{x}, \quad (13)$$

where the IS density $g(\mathbf{x}, \cdot)$ is such that $g(\mathbf{x}, \cdot) = 0 \Rightarrow S(\mathbf{x}) f(\mathbf{x}, \cdot) = 0$, i.e., g dominates Sf . Thus, the estimation problem is alternatively seen as

$$\ell = \mathbb{E}_g \left\{ S(\mathbf{x}) \frac{f(\mathbf{x}, \cdot)}{g(\mathbf{x}, \cdot)} \right\}, \quad (14)$$

where now the sampling is performed with respect to the IS distribution g , which ideally is better suited for the rare event estimation. In this novel setting, the estimation can be done with the aid of the estimator

$$\hat{\ell} = \frac{1}{N_s} \sum_{k=1}^{N_s} H(\mathbf{X}_k) \frac{f(\mathbf{X}_k, \cdot)}{g(\mathbf{X}_k, \cdot)}, \quad (15)$$

where the i.i.d. samples \mathbf{X}_k ($k = 1, \dots, N_s$) now are obtained according to $g(\mathbf{x}, \cdot)$.

The IS density g directly affects the efficiency of $\hat{\ell}$, so a key challenge here, crucial for achieving accurate estimates of rare event probabilities, is to find the density g which minimizes the variance of the estimator $\hat{\ell}$, i.e.,

$$g^*(\mathbf{x}, \cdot) = \arg \min_g \mathbb{V}_g \left\{ \hat{\ell} \right\}, \quad (16)$$

which due to the i.i.d. nature of the samples is equivalent to

$$g^*(\mathbf{x}, \cdot) = \arg \min_g \mathbb{V}_g \left\{ S(\mathbf{X}) \frac{f(\mathbf{X}, \cdot)}{g(\mathbf{X}, \cdot)} \right\}, \quad (17)$$

a nonlinear program which has as analytic solution

$$g^*(\mathbf{x}, \cdot) = \frac{S(\mathbf{x}) f(\mathbf{x}, \cdot)}{\int_{\mathbb{R}^n} S(\mathbf{x}) f(\mathbf{x}, \cdot) d\mathbf{x}}, \quad (18)$$

since for $g = g^*$ we have

$$\mathbb{V}_{g^*} \left\{ S(\mathbf{X}) \frac{f(\mathbf{X}, \cdot)}{g(\mathbf{X}, \cdot)} \right\} = \mathbb{V}_{g^*} \left\{ \int_{\mathbb{R}^n} S(\mathbf{x}) f(\mathbf{x}, \cdot) d\mathbf{x} \right\} = \mathbb{V}_{g^*} \{\ell\} = 0. \quad (19)$$

However, implementing this minimization strategy is not feasible in practical problems, because the optimal IS density depends on ℓ , which is not known a priori. In fact, the stochastic simulation is done to get this value!

To address this challenge, an alternative approach is to restrict the search space for the IS density to a parametric family that shares similarities with the original distribution of \mathbf{X} , i.e., if $\mathbf{X} \sim f(\mathbf{x}, \mathbf{u})$ for a hyperparameter vector \mathbf{u} we choose $g(\mathbf{x}, \cdot) = f(\mathbf{x}, \mathbf{v})$ for a reference hyperparameter vector \mathbf{v} . This approach ensures that the IS density remains within a feasible and computationally tractable domain.

Thus, the IS estimator (15) becomes

$$\hat{\ell} = \frac{1}{N_s} \sum_{k=1}^{N_s} S(\mathbf{X}_k) \frac{f(\mathbf{X}_k, \mathbf{u})}{f(\mathbf{X}_k, \mathbf{v})}, \quad (20)$$

with i.i.d. samples \mathbf{X}_k ($k = 1, \dots, N_s$) drawn from $f(\mathbf{x}, \mathbf{v})$, so that we replace the variance minimization problem in (17) for the nonlinear program

$$\begin{aligned} \mathbf{v}^* &= \arg \min_{\mathbf{v}} \mathbb{V}_{\mathbf{v}} \left\{ S(\mathbf{X}) \frac{f(\mathbf{X}, \mathbf{u})}{f(\mathbf{X}, \mathbf{v})} \right\} \\ &= \arg \min_{\mathbf{v}} \mathbb{E}_{\mathbf{v}} \left\{ S(\mathbf{X})^2 \frac{f(\mathbf{X}, \mathbf{u})^2}{f(\mathbf{X}, \mathbf{v})^2} \right\} - \mathbb{E}_{\mathbf{v}} \left\{ S(\mathbf{X}) \frac{f(\mathbf{X}, \mathbf{u})}{f(\mathbf{X}, \mathbf{v})} \right\}^2 \\ &= \arg \min_{\mathbf{v}} \int_{\mathbb{R}^n} S(\mathbf{x})^2 \frac{f(\mathbf{x}, \mathbf{u})^2}{f(\mathbf{x}, \mathbf{v})^2} f(\mathbf{x}, \mathbf{v}) d\mathbf{x} - \left(\int_{\mathbb{R}^n} S(\mathbf{x}) \frac{f(\mathbf{x}, \mathbf{u})}{f(\mathbf{x}, \mathbf{v})} f(\mathbf{x}, \mathbf{v}) d\mathbf{x} \right)^2 \quad (21) \\ &= \arg \min_{\mathbf{v}} \int_{\mathbb{R}^n} S(\mathbf{x})^2 \frac{f(\mathbf{x}, \mathbf{u})}{f(\mathbf{x}, \mathbf{v})} f(\mathbf{x}, \mathbf{u}) d\mathbf{x} - \left(\int_{\mathbb{R}^n} S(\mathbf{x}) f(\mathbf{x}, \mathbf{u}) d\mathbf{x} \right)^2 \\ &= \arg \min_{\mathbf{v}} \int_{\mathbb{R}^n} S(\mathbf{x})^2 \frac{f(\mathbf{x}, \mathbf{u})}{f(\mathbf{x}, \mathbf{v})} f(\mathbf{x}, \mathbf{u}) d\mathbf{x} \\ &= \arg \min_{\mathbf{v}} \mathbb{E}_{\mathbf{u}} \left\{ S(\mathbf{X})^2 \frac{f(\mathbf{X}, \mathbf{u})}{f(\mathbf{X}, \mathbf{v})} \right\}, \end{aligned}$$

which is numerically solved with aid of the stochastic counterpart program

$$\hat{\mathbf{v}}^* = \arg \min_{\mathbf{v}} \frac{1}{N_s} \sum_{k=1}^{N_s} S(\mathbf{X}_k)^2 \frac{f(\mathbf{X}_k, \mathbf{u})}{f(\mathbf{X}_k, \mathbf{v})}, \quad (22)$$

with i.i.d. samples \mathbf{X}_k ($k = 1, \dots, N_s$) drawn from $f(\mathbf{x}, \mathbf{u})$, that is known in advance.

By confining the search space to a known parametric family, this approach streamlines the practical implementation of the estimation problem (13). Consequently, it also facilitates the resolution of the original optimization problem (1). However, this method necessitates solving a nonlinear program numerically, introducing a level of complexity that can be mitigated by employing a strategy based on Kullback–Leibler divergence minimization, as elucidated in the subsequent section.

3.4. Estimating rare events via Kullback–Leibler divergence minimization

The Kullback–Leibler (KL) divergence, often referred to as relative entropy or the cross-entropy, serves as a measure of statistical distance between two probability distributions. Specifically, it quantifies how one distribution g diverges from another distribution f . Mathematically, the KL divergence between g and f is defined as

$$\mathcal{D}_{\text{KL}}(g \parallel f) = \int_{\mathbb{R}^n} g(\mathbf{x}, \cdot) \ln \frac{g(\mathbf{x}, \cdot)}{f(\mathbf{x}, \cdot)} d\mathbf{x}, \quad (23)$$

which is equivalent to

$$\mathcal{D}_{\text{KL}}(g \parallel f) = \int_{\mathbb{R}^n} g(\mathbf{x}, \cdot) \ln g(\mathbf{x}, \cdot) d\mathbf{x} - \int_{\mathbb{R}^n} g(\mathbf{x}, \cdot) \ln f(\mathbf{x}, \cdot) d\mathbf{x}. \quad (24)$$

It is crucial to recognize that while KL divergence is commonly referred to as a “distance”, it does not fully adhere to the criteria of a true distance function in the metric space sense. Notably, it lacks symmetry, meaning $\mathcal{D}_{\text{KL}}(g \parallel f) \neq \mathcal{D}_{\text{KL}}(f \parallel g)$, and it does not satisfy the triangle inequality, indicated by $\mathcal{D}_{\text{KL}}(g \parallel f) > \mathcal{D}_{\text{KL}}(g \parallel z) + \mathcal{D}_{\text{KL}}(z \parallel f)$. However, KL divergence is always non-negative, represented as $\mathcal{D}_{\text{KL}}(g \parallel f) \geq 0$, with equality occurring only when g is identical to f , i.e., $\mathcal{D}_{\text{KL}}(g \parallel f) = 0 \Leftrightarrow g = f$. In this context, KL divergence remains a valuable tool for quantifying the difference between two probability distributions.

Here we aim to utilize the KL divergence, denoted as $\mathcal{D}_{\text{KL}}(g^* \parallel f)$, to measure the divergence between the optimal IS distribution $g^*(\mathbf{x}, \mathbf{u}) \propto S(\mathbf{x}) f(\mathbf{x}, \mathbf{u})$, defined in (16), and the actual IS distribution $f(\mathbf{x}, \mathbf{v})$ underlying the stochastic program (22). The objective is to adaptively adjust the hyperparameter vector \mathbf{v} to bring $f(\mathbf{x}, \mathbf{v})$ closer and closer to $g^*(\mathbf{x}, \mathbf{u})$.

This strategy entails solving the optimization problem expressed as

$$\begin{aligned} \mathbf{v}^* &= \arg \min_{\mathbf{v}} \mathcal{D}_{\text{KL}}(g^*(\cdot, \mathbf{u}) \parallel f(\cdot, \mathbf{v})) \\ &= \arg \min_{\mathbf{v}} \int_{\mathbb{R}^n} g^*(\mathbf{x}, \mathbf{u}) \ln g^*(\mathbf{x}, \mathbf{u}) d\mathbf{x} - \int_{\mathbb{R}^n} g^*(\mathbf{x}, \mathbf{u}) \ln f(\mathbf{x}, \mathbf{v}) d\mathbf{x} \\ &= \arg \min_{\mathbf{v}} \int_{\mathbb{R}^n} S(\mathbf{x}) f(\mathbf{x}, \mathbf{u}) \ln (S(\mathbf{x}) f(\mathbf{x}, \mathbf{u})) d\mathbf{x} - \int_{\mathbb{R}^n} S(\mathbf{x}) f(\mathbf{x}, \mathbf{u}) \ln f(\mathbf{x}, \mathbf{v}) d\mathbf{x} \\ &= \arg \min_{\mathbf{v}} - \int_{\mathbb{R}^n} S(\mathbf{x}) f(\mathbf{x}, \mathbf{u}) \ln f(\mathbf{x}, \mathbf{v}) d\mathbf{x} \\ &= \arg \max_{\mathbf{v}} \int_{\mathbb{R}^n} S(\mathbf{x}) \ln f(\mathbf{x}, \mathbf{v}) f(\mathbf{x}, \mathbf{u}) d\mathbf{x} \\ &= \arg \max_{\mathbf{v}} \mathbb{E}_{\mathbf{u}} \{S(\mathbf{X}) \ln f(\mathbf{X}, \mathbf{v})\}. \end{aligned} \quad (25)$$

This can be accomplished using the stochastic program

$$\hat{\mathbf{v}}^* = \arg \max_{\mathbf{v}} \frac{1}{N_s} \sum_{k=1}^{N_s} S(\mathbf{X}_k) \ln f(\mathbf{X}_k, \mathbf{v}), \quad (26)$$

where \mathbf{X}_k ($k = 1, \dots, N_s$) represents i.i.d. realizations obtained according to $f(\mathbf{x}, \mathbf{u})$.

For the optimization problem (1), we are interested in the indicator function $S(\mathbf{X}) = \mathbb{1}_{F(\mathbf{X}) \leq \gamma}$. In this case, (26) simplifies to

$$\begin{aligned}\hat{\mathbf{v}}^* &= \arg \max_{\mathbf{v}} \frac{1}{N_s} \sum_{k=1}^{N_s} \mathbb{1}_{F(\mathbf{X}_k) \leq \gamma} \ln f(\mathbf{X}_k, \mathbf{v}) \\ &= \arg \max_{\mathbf{v}} \sum_{\mathbf{X}_k \in \mathcal{E}} \ln f(\mathbf{X}_k, \mathbf{v}),\end{aligned}\quad (27)$$

where the elite samples set \mathcal{E} comprises the samples \mathbf{X}_k for which $F(\mathbf{X}_k) \leq \gamma$. It is evident that the solution to this problem is the classic maximum likelihood estimator (MLE).

This approach offers a significant advantage over the method presented in the previous section (22), as it often admits an analytic solution in typical scenarios. Specifically, this is the case when the random vector \mathbf{X} comprises independent random variables with continuous distributions in the natural exponential family or discrete distributions with finite support.

For instance, consider a scenario where each component of \mathbf{X} is independent and follows a multivariate truncated Gaussian distribution, i.e., $\mathbf{X} \sim \mathcal{T}\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}, \mathbf{x}_\ell, \mathbf{x}_u)$. Here, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ represent the mean and standard deviation, respectively, while \mathbf{x}_ℓ and \mathbf{x}_u define the lower and upper bounds. In this case, the MLE for the mean $\boldsymbol{\mu}$ is given by

$$\hat{\boldsymbol{\mu}} = \frac{1}{N_{\mathcal{E}}} \sum_{\mathbf{X}_k \in \mathcal{E}} \mathbf{X}_k, \quad (28)$$

while the MLE for the standard deviation $\boldsymbol{\sigma}$ is computed as

$$\hat{\boldsymbol{\sigma}} = \sqrt{\text{diag} \left(\frac{1}{N_{\mathcal{E}}} \sum_{\mathbf{X}_k \in \mathcal{E}} (\mathbf{X}_k - \hat{\boldsymbol{\mu}})(\mathbf{X}_k - \hat{\boldsymbol{\mu}})^T \right)}, \quad (29)$$

where $N_{\mathcal{E}}$ denotes the number of elements in \mathcal{E} .

3.5. The classic CE algorithm

In this section, we present an overview of the CE algorithm for optimization, emphasizing its connection to the theory of rare event estimation through KL divergence minimization. For sake of simplicity and efficiency, the multivariate truncated Gaussian distribution is used for all the random variables in \mathbf{X} , assumed as mutually independent.

A natural approach to tackling the optimization problem (1) from this perspective is to solve the stochastic program (27) to obtain $\hat{\mathbf{v}}^*$. Subsequently, the distribution $f(\mathbf{x}, \hat{\mathbf{v}}^*)$ can be used to generate realizations that provide approximate solutions to the optimization problem. The mean of these samples, defined in (28), can serve as an estimator for the global optimum \mathbf{x}^* .

However, it is crucial to highlight that if the associated rare probability $F(\mathbf{X}_k) \leq \gamma$ is exceedingly small, the distribution $f(\mathbf{x}, \hat{\mathbf{v}}^*)$ may not efficiently produce realizations that approximate \mathbf{x}^* . This occurs because many of the samples \mathbf{X}_k in $\mathbb{1}_{F(\mathbf{X}_k) \leq \gamma}$ will yield zero, leading to a mean value $\hat{\boldsymbol{\mu}}$ far from \mathbf{x}^* . One potential solution is to employ an iterative multi-level process, which instead of using just one distribution to estimate the optimum, employs a sequence of distributions $f(\mathbf{x}, \hat{\mathbf{v}}_0), f(\mathbf{x}, \hat{\mathbf{v}}_1), \dots, f(\mathbf{x}, \hat{\mathbf{v}}_t), \dots$ that are sequentially calculated, producing a

sequence of approximations $\hat{\boldsymbol{\mu}}_0, \hat{\boldsymbol{\mu}}_1, \dots, \hat{\boldsymbol{\mu}}_t, \dots$ for the optimum \boldsymbol{v}^* . Each step in this process involves a pair of estimators $(\hat{\gamma}_t, \hat{\boldsymbol{v}}_t)$ for the pair $(\gamma^*, \boldsymbol{v}^*)$.

Ideally, these estimators converge, $(\hat{\gamma}_t, \hat{\boldsymbol{v}}_t) \rightarrow (\gamma^*, \boldsymbol{v}^*)$, in a way that $f(\mathbf{x}, \hat{\boldsymbol{v}}_t) \rightarrow \delta(\mathbf{x} - \mathbf{x}^*)$, where $\delta(\mathbf{x} - \mathbf{x}^*)$ denotes Dirac's delta distribution, with all its mass centered at \mathbf{x}^* . In practice, the iterative process is terminated once a convergence criterion is met. In such scenarios, $f(\mathbf{x}, \hat{\boldsymbol{v}}_t)$ progressively moves toward \mathbf{x}^* while also shrinking.

This estimation process is multi-level because both γ_t and \boldsymbol{v}_t are sequentially updated. To achieve this, a rarity parameter $0 < \rho < 1$ is introduced, and γ_t is defined as the $(1 - \rho)$ -quantile for $F(\mathbf{X})$ with $\mathbf{X} \sim f(\mathbf{x}, \boldsymbol{v}_{t-1})$. Formally, γ_t is characterized by

$$\mathcal{P}\{F(\mathbf{X}) \leq \gamma\} \geq \rho \text{ and } \mathcal{P}\{F(\mathbf{X}) > \gamma\} \geq 1 - \rho, \quad (30)$$

and it is estimated using the order statistic

$$\hat{\gamma}_t = F_{(N_{\mathcal{E}})}, \quad (31)$$

where $F_{(1)} \leq F_{(2)} \leq \dots \leq F_{(N_s)}$ represents the ordered values of $F(\mathbf{x})$ calculated at the samples $\mathbf{X}_k (k = 1, \dots, N_s)$ from $f(\mathbf{x}, \boldsymbol{v}_{t-1})$. In this estimator, $N_{\mathcal{E}} = \text{ceil}(\rho N_s)$ denotes the size of the elite set $\mathcal{E}_t = \{\mathbf{X}_k \mid F(\mathbf{X}_k) \leq \gamma_t\}$.

Subsequently, $\hat{\boldsymbol{v}}_t$ is obtained from

$$\hat{\boldsymbol{v}}_t = \arg \max_{\boldsymbol{v}} \sum_{\mathbf{X}_k \in \mathcal{E}_t} \ln f(\mathbf{X}_k, \boldsymbol{v}). \quad (32)$$

For a multivariate Gaussian distribution, $\hat{\boldsymbol{v}}_t = (\tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\sigma}}_t)$ is computed analytically using the formulas (28) and (29).

Finally, these vectors are updated according to the smoothing schemes

$$\hat{\boldsymbol{\mu}}_t = \alpha \tilde{\boldsymbol{\mu}}_t + (1 - \alpha) \hat{\boldsymbol{\mu}}_{t-1}, \quad (33)$$

$$\hat{\boldsymbol{\sigma}}_t = \beta_t \tilde{\boldsymbol{\sigma}}_t + (1 - \beta_t) \hat{\boldsymbol{\sigma}}_{t-1}, \quad (34)$$

where $0 < \alpha \leq 1$ is the smoothing parameter and

$$\beta_t = \beta - \beta \left(1 - \frac{1}{t}\right)^q, \quad (35)$$

with $\beta \geq 0$ and $q \in \mathbb{N}$.

The convergence of the iterative process is controlled by the criterion

$$\|\hat{\boldsymbol{\sigma}}_t - \hat{\boldsymbol{\sigma}}_{t-1}\|_{w rms} \leq 1, \quad (36)$$

where the weighted root-mean-square norm of $\mathbf{x} \in \mathbb{R}^n$ is computed as follows

$$\|\mathbf{x}\|_{w rms} = \sqrt{\frac{1}{n} \sum_{j=1}^n (w_j x_j)^2}, \quad (37)$$

where w_j represents the error weights defined by

$$w_j = \frac{1}{\text{atol}_j + |x_j| \text{rtol}}. \quad (38)$$

Here, atol_j and rtol represent the absolute and relative tolerances, respectively. The normalization provided by the weights in (37) ensures that a weighted norm of approximately 1 in (36) indicates convergence. This convergence criterion, commonly employed in high-quality differential equation solvers [Hindmarsh, Brown, Grant, Lee, Serban, Shumaker, and Woodward \(2005\)](#); [Shampine and Reichelt \(1997\)](#), ensures robust error control.

This algorithm is summarized as follows:

1. **Initialization:** Begin by defining the necessary parameters: the number of total samples N_s , the number of elite samples $N_{\mathcal{E}}$ (which must be less than N_s), an absolute tolerance atol , a relative tolerance rtol , the maximum number of iteration levels t_{\max} , the family of probability distributions $f(\cdot, \mathbf{v})$, an initial vector of hyperparameters $\hat{\mathbf{v}}_0$, the smoothing parameters α , β , and q , and set the iteration level counter to zero, i.e., $t = 0$.
2. **Update Iteration Level:** Increment the iteration level counter by 1, i.e., $t = t + 1$.
3. **Sample Generation:** Generate N_s independent and identically distributed (iid) samples from $f(\cdot, \mathbf{v}_{t-1})$, denoted by $\mathbf{X}_1, \dots, \mathbf{X}_{N_s}$.
4. **Objective Function Evaluation:** Evaluate the objective function $F(\mathbf{x})$ at each sample $\mathbf{X}_1, \dots, \mathbf{X}_{N_s}$, sort the results as $F_{(1)} \leq \dots \leq F_{(N_s)}$, and define the elite sample set \mathcal{E}_t containing the $N_{\mathcal{E}}$ best points, i.e., those associated with the minimum values.
5. **Update Estimators:** Update the estimators $\hat{\gamma}_t$ and $\hat{\mathbf{v}}_t$ using the elite sample set. Use order statistic estimator from (31) for $\hat{\gamma}_t$ and maximum likelihood estimators from (32) for $\hat{\mathbf{v}}_t$. If necessary, apply a scheme for smooth updating.
6. **Iteration:** Repeat steps 2 through 5 of this algorithm until a stopping criterion is met. This can be when the maximum number of iterations t_{\max} is reached, or when the weighted root-mean-square norm of the difference vector $\hat{\boldsymbol{\sigma}}_t - \hat{\boldsymbol{\sigma}}_{t-1}$ falls below or equals 1, signaling convergence as per the defined criteria. For a comprehensive discussion on various stopping strategies, including those based on alternative convergence measures and practical considerations, refer to Section 4.2.

A schematic representation of the above CE algorithm is presented in Figure 3.

3.6. Remarks about CE method

The CE algorithm possesses several noteworthy characteristics that contribute to its appeal and effectiveness in solving optimization problems. These features include:

- *Simplicity:* The CE algorithm is intuitively understandable and has only a few control parameters, namely N_s , $N_{\mathcal{E}}$, t_{\max} , atol and rtol . Each of these parameters has a clear interpretation, making the algorithm easy to implement and tune.
- *Robustness:* Theoretical results establish that, under typical conditions, the CE algorithm is guaranteed to converge to the global optimum if the optimization problem has a single global extremum. This robustness ensures its reliability in a variety of optimization scenarios.

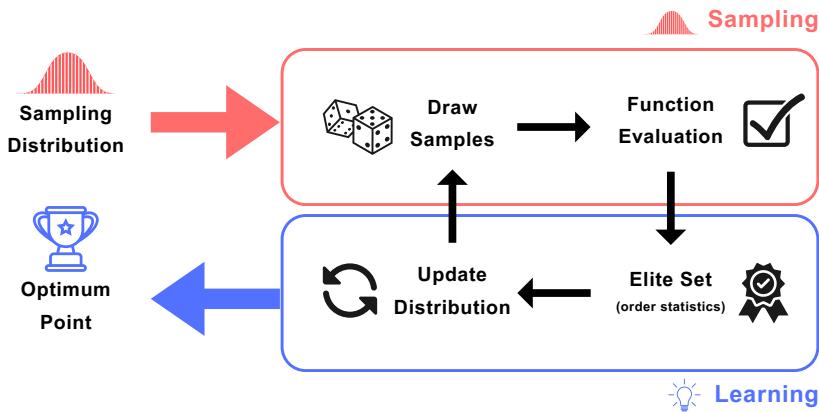


Figure 3: Cross-Entropy Method Flowchart: This diagram shows the CE method's steps, from starting with a sample distribution to finding the best solution. It involves sampling, function evaluation, refining the distribution with the best samples, and repeating these steps until the optimal solution is found.

- *Efficiency:* The CE algorithm often exhibits fast convergence rates compared to traditional global search metaheuristics such as genetic algorithms. This efficiency makes it particularly suitable for problems where computational resources are limited or where rapid optimization is desired.
- *Generality:* Unlike some optimization methods that require specific properties of the objective function, the CE algorithm is applicable to a wide range of non-convex optimization problems. It can handle functions that are non-differentiable, discontinuous, or lack other regularity conditions.
- *Extensibility:* The theoretical framework underlying the CE algorithm is general and can be applied to optimization problems of any finite dimension. While computational cost and the curse of dimensionality may limit its practical applicability in high-dimensional spaces, the method remains versatile and adaptable.
- *Ease of Implementation:* Thanks to its simplicity and the availability of the MATLAB black-box package presented in this paper, implementing the CE algorithm for a particular optimization problem is straightforward. This package offers a robust and modern implementation of CE, making it easy to apply in various optimization tasks.

The mathematical foundation of the CE algorithm is well-established, with a rigorous development of its formalism over the past decades. Theoretical theorems have been formulated to rigorously establish the conditions under which the algorithm converges. While the detailed mathematical analysis is beyond the scope of this paper, interested readers can refer to Rubinstein and Kroese (2004, 2016) for further exploration.

In conclusion, the CE algorithm's simplicity, robustness, efficiency, generality, extensibility, and ease of implementation make it a valuable tool for non-convex optimization problems.

Its principled approach and clear control parameters make it a recommendable choice for a wide range of optimization tasks.

4. MATLAB implementation: The **CEopt** package

The **CEopt** package is a MATLAB-based framework designed to solve non-convex optimization problems using the Cross-Entropy method. The development of this package is motivated by the need for an easy-to-use, efficient, modular and general purpose tool based on CE for both theoretical researchers and practitioners in the field of optimization. The **CEopt** package closely mirrors MATLAB's black box function `ga` (Genetic Algorithm), in its user-friendliness and functionality, making it a valuable tool for solving a wide range of optimization problems without requiring in-depth knowledge of the underlying algorithmic intricacies.

4.1. Code architecture and modularity

The **CEopt** package is structured to maximize code modularity and readability, adopting a design that segregates the optimization process into distinct, logically organized functions, as can be seen in Figure 4. This modular architecture not only enhances maintainability but also facilitates the understanding and potential customization of the code. The main function, `CEopt`, serves as the entry point for the optimization process, encapsulating the core CE method logic and iteratively refining the solution until convergence criteria are met or computational limits are reached.

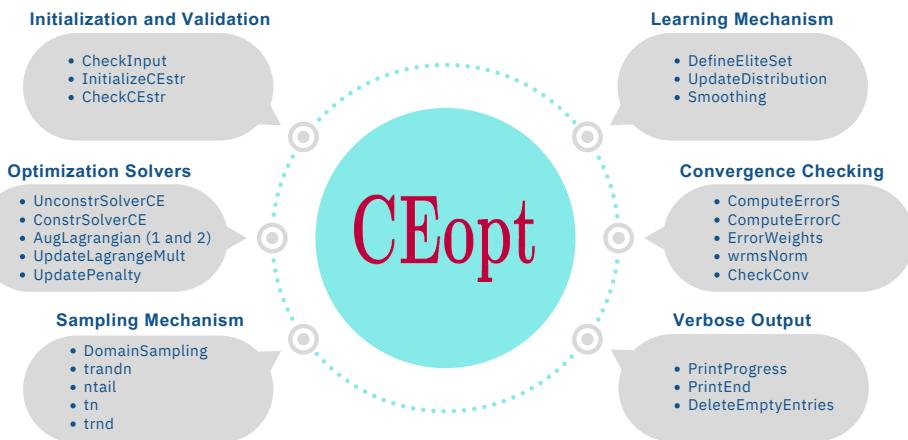


Figure 4: **CEopt** MATLAB Implementation: This diagram outlines the key components of the **CEopt** package, starting with input checks and setup, through to optimization solvers for various problem types. It highlights the sampling and learning processes used to refine solutions, along with methods for convergence verification and detailed progress reporting, illustrating the systematic approach embedded in **CEopt** for solving optimization tasks.

Key components of the package include:

- *Initialization and Validation*: Sets up the initial parameters and structures and checks user inputs for consistency and correctness, ensuring robust and error-free execution.

- *Optimization Solvers:* Offers both constrained and unconstrained solvers, providing versatile options for tackling a wide range of optimization problems.
- *Sampling Mechanism:* Generates samples from a truncated Gaussian distribution, focusing the search on promising regions of the solution space.
- *Learning Mechanism:* Identifies and processes elite samples to update the distribution parameters, guiding the algorithm towards optimal solutions.
- *Convergence Checking:* Monitors the optimization process, assessing convergence based on predefined criteria and computational bounds.
- *Verbose Output:* Delivers concise progress updates and diagnostic information, enhancing the optimization process transparency and ease of analysis.

4.2. Ease of use and functionality

Operating the **CEopt** package is straightforward: users need only to provide the necessary input parameters and invoke the **CEopt** function to commence the optimization process. The simplicity of use does not compromise the method's power or versatility, making it accessible to a wide audience of optimization practitioners across various domains of knowledge. The basic syntax for invoking **CEopt** with all inputs and outputs is given by:

```
[Xopt, Fopt, ExitFlag, CEstr] = CEopt(fun, xmean0, sigma0, lb, ub, nonlcon, CEstr)
```

Below is a detailed explanation of the inputs and outputs of the **CEopt** package:

- Input Parameters
 - **fun:** The objective function handle to be minimized or maximized;
 - **xmean0:** Initial mean of the sampling distribution;
 - **sigma0:** Initial standard deviation of the sampling distribution;
 - **lb** and **ub:** Lower and upper bounds for the design variables, defining a rectangular feasible region to search the solution;
 - **nonlcon:** Handle for nonlinear constraint functions (optional);
 - **CEstr:** A struct containing various settings and parameters for customizing the CE method's behavior and some features of **CEopt** implementation (optional).
- Output Parameters
 - **Xopt:** The optimal solution found by the algorithm;
 - **Fopt:** The value of the objective function at the optimal solution;
 - **ExitFlag:** An integer flag indicating the optimization process's termination reason;
 - **CEstr:** Updated **CEstr** struct, also containing diagnostic information about the algorithm execution and the final state of the optimization process.

The `ExitFlag` serves as a crucial indicator of the outcome of the optimization process in the **CEopt** package. It provides concise feedback on why the optimization procedure terminated, allowing users to quickly assess the success of the optimization and understand any issues that may have arisen. Here is a detailed explanation of the possible values for `ExitFlag` and their implications:

- **1 - Maximum Number of Iterations Reached:** The optimization process stopped because it hit the maximum limit of iterations specified by the user through the `MaxIter` field in the `CEstr` structure. This may indicate that more iterations are needed to converge to a solution or that the optimization is stalled;
- **2 - Solution Stalled:** The algorithm terminated because there was no significant change in the objective function value over a predefined number of iterations (`MaxStall`). This suggests that the algorithm might have converged to a global optimum solution or is trapped in a local minimum;
- **3 - Maximum Number of Function Evaluations Reached:** This flag indicates that the optimization ended because the total number of objective function evaluations exceeded the user-defined maximum (`MaxFcount`). Like flag 1, this might suggest the need for higher limits or a review of the optimization strategy;
- **4 - Objective Function Tolerance Achieved:** The optimization successfully concluded because the range of objective function values across iterations fell below the specified tolerance (`TolFun`). For constrained problems, it also implies that constraint violations are small. This is typically a sign of successful convergence;
- **5 - Standard Deviation Convergence:** The process terminated due to the small variation in the standard deviation of the sampling distribution (`sigma`), and, for constrained optimizations, constraint violations are also small. This indicates that the population of solutions is converging towards a point, suggesting a potential optimum solution has been found by the algorithm;
- **6 - Minimum Function Value Criterion Met:** This outcome means the optimization found a solution where the objective function value is equal to or better than the target minimum function value set by the user (`MinFval`). It is a clear indication of success, especially in applications where a specific performance benchmark is targeted.

Understanding the meaning behind each `ExitFlag` value is essential for interpreting the results of the optimization process. It not only provides insights into the behavior and performance of the CE algorithm but also guides users in making decisions about potential next steps. Depending on the `ExitFlag` received, users may adjust algorithm parameters, extend computational limits, or refine their problem formulation seeking better results in next runs.

The `CEstr` structure is central to the customization and configuration of the optimization process in the **CEopt** package. It offers users a high degree of control over the behavior of the Cross-Entropy optimization algorithm, enabling the adjustment of various parameters to best fit the specific requirements of their optimization problem. Furthermore, it encapsulates a rich tapestry of historical insights into the optimization's evolutionary path, granting users the capacity to meticulously scrutinize the algorithm's trajectory, identify any emergent issues,

and strategically refine parameters for optimized outcomes. Below is a detailed description of all possible fields within the **CEstr** structure:

- **Verbose:** A boolean flag that, when set to `true`, enables the display of detailed information about the optimization process at each iteration on screen. This includes the current iteration number, objective function value, and other diagnostic information;
- **isConstrained:** A boolean flag indicating whether the optimization problem includes constraints. This influences the choice of optimization strategy within algorithm;
- **isVectorized:** A boolean flag indicating whether the objective function `fun` and the nonlinear constraints function `nonlcon` (if provided) are vectorized. Vectorized functions are capable of taking multiple input samples at once and returning their corresponding output values in a single call, which can significantly enhance computational efficiency;
- **Nvars:** Indicates the number of design variables in the optimization problem. This is set automatically based on the input parameters;
- **EliteFactor:** Determines the proportion of samples considered as elite. These are the best samples from the current iteration used to update the sampling distribution;
- **Nsamp:** The total number of samples to be drawn at each iteration. It affects the breadth of the search and can influence both the speed and the quality of convergence;
- **MaxIter:** Specifies the maximum number of iterations to be performed by the algorithm. This serves as a stopping criterion to prevent the algorithm from running indefinitely;
- **MaxStall:** Defines the maximum number of iterations without any improvement in the objective function value that the algorithm will tolerate before terminating. This helps in avoiding unnecessary computations when the iteration has converged to the global optimum or is stuck in a local minimum;
- **MaxFcount:** Sets an upper limit on the total number of objective function evaluations. This is another stopping criterion that ensures efficient use of computational resources;
- **MinFval:** Establishes a threshold for the objective function value. If the algorithm finds a solution with an objective function value below this threshold, it will terminate, assuming an acceptable solution has been found;
- **TolAbs:** An absolute tolerance level for the convergence criterion based on the standard deviation. It helps in determining when the changes in the solution vector between iterations are sufficiently small to consider the solution as converged;
- **TolRel:** A relative tolerance level for the convergence criterion based on the standard deviation, similar to **TolAbs**, but relative to the size of the solution vector. It provides a scale-invariant measure of convergence;
- **TolCon:** Specifies the tolerance for constraint violations. It allows for slight violations in the constraints, facilitating convergence in problems where strict adherence to constraints is challenging to achieve;

- **TolFun**: Indicates the tolerance for changes in the objective function value. If the change in function values between iterations is below this threshold, the process terminate;
- **alpha**: Smoothing parameter used for updating the mean of the sampling distribution. It controls the balance between exploration and exploitation in the search space;
- **beta**: Similar to **alpha**, but for the standard deviation of the sampling distribution. It adjusts the spread of the distribution, affecting the diversity of the samples generated;
- **q**: An exponent used in the dynamic update formula for **beta**. It allows to change the smoothing effect over time, potentially improving convergence properties;
- **NonlconAlgorithm**: Specifies the algorithm used for handling nonlinear constraints, such as the augmented Lagrangian method. It allows for the selection of the most appropriate method based on the nature of the constraints;
- **InitialPenalty**: Sets the initial penalty value for the augmented Lagrangian when handling constrained problems. It affects the initial weight given to constraint violations;
- **PenaltyFactor**: Determines the factor by which the penalty parameter is increased during the optimization process. This is relevant in the context of methods that adjust penalties dynamically to enforce constraint satisfaction;
- **xmean**: Stores the history of mean values of the sampling distribution over iterations. This array provides insight into how the central tendency of the sampled solutions evolves as the optimization progresses;
- **xmedian**: Captures the history of median values of the sampling distribution. Similar to **xmean**, but offers a different statistical perspective on the distribution's central tendency;
- **xbest**: Contains the best sample (solution) found until that iteration. This field is particularly useful for observing the progression towards the optimal solution across iterations;
- **Fmean**: Records the history of mean objective function values of the elite set at each iteration. It helps in assessing the overall performance improvement of the algorithm;
- **Fmedian**: Similar to **Fmean**, but tracks the median of the objective function values among the elite samples. This provides an alternative measure of central tendency, which is less sensitive to outliers;
- **Fbest**: Maintains the history of the best objective function value found up to each iteration. This field is crucial for understanding the optimization process's efficacy and convergence behavior;
- **sigma**: Keeps a record of the standard deviation of the sampling distribution over iterations. Variations in **sigma** reflect changes in the diversity of the sampled solutions and the algorithm's exploration-exploitation balance;
- **ErrorS**: Logs the history of the standard deviation error measure. This field is instrumental in monitoring the convergence of the sampling distribution's spread;

- **ErrorC:** Provides the history of the constraint violation error measure for constrained problems. It indicates how well the solutions satisfy the constraints over time;
- **iter:** Tracks the total number of iterations performed during the optimization process. Fundamental for managing loop execution and assessing computational effort;
- **stall:** Records the number of iterations without significant progress. A high stall count may indicate convergence to a suboptimal solution or necessitate parameter adjustment;
- **Fcount:** Counts the total number of objective function evaluations conducted. This metric is valuable for evaluating the computational cost of the optimization process;
- **ConvergenceStatus:** Reflects the outcome of the optimization process with regard to convergence. This boolean field is set to **true** if the algorithm successfully meets the convergence criteria, such as reaching a satisfactory solution within specified tolerances or achieving adequate constraint satisfaction in the case of constrained problems. Conversely, it is set to **false** if the optimization process terminates due to reaching computational limits (like the maximum number of iterations or function evaluations) without satisfying the convergence conditions.

This comprehensive list of fields within the **CEstr** structure demonstrates the flexibility and depth of customization available in the **CEopt** package. Users can fine-tune these parameters to adapt the optimization process to their specific needs, enhancing the package's utility across a broad spectrum of optimization problems. For additional information regarding code installation, licensing, and more, please visit the **CEopt** website at <https://ceopt.org>.

In the following section, we present several case studies that demonstrate the **CEopt** package's application across a diverse range of non-convex optimization problems. These examples are designed to showcase the versatility, effectiveness, and broad applicability of **CEopt** in addressing complex optimization challenges in various domains. Through these illustrative examples, readers will gain insights into how **CEopt** can be seamlessly integrated into practical scenarios, highlighting its value and utility in research and industry settings alike.

5. Numerical experiments

This section of the paper is dedicated to showcasing the robust capabilities of the **CEopt** package through a series of detailed numerical experiments. Each example has been carefully selected to illustrate the package's ability to efficiently solve a variety of complex, non-convex optimization problems that are commonly encountered in both research and industrial applications. By demonstrating **CEopt** in action across different scenarios, we aim to highlight its adaptability, precision, and effectiveness. These case studies not only validate the theoretical aspects discussed in previous sections but also provide practical insights that can be directly applied in diverse fields such as machine learning, engineering design, and computational finance. The following examples will demonstrate the implementation specifics, the ease of use of **CEopt**, and its performance in achieving optimal solutions under challenging conditions.

5.1. Example 1: A Gaussian mixture in 1D

For pedagogical reasons, we begin with a straightforward example featuring a Gaussian mix-

ture function. This case study is designed to demonstrate the ease of using **C***E***opt** as a black-box optimization command. The objective function considered here is a simple Gaussian mixture given by

$$F(x) = -0.8 \exp(-(x-2)^2) - 0.5 \exp(-(x+2)^2) + 1. \quad (39)$$

Despite its simplicity, this function has illustrative properties, allowing us to demonstrate the effectiveness of the CE method in navigating complex landscapes to locate minima, making it an ideal candidate to showcase the capabilities of **C***E***opt**. The MATLAB code provided below automates the optimization process using **C***E***opt**, treating it as a black box that requires minimal user intervention for parameter setup.

```

MainCEoptExample1.m
1 clc; clear; close all;
2
3 disp(' ----- ')
4 disp(' MainCEoptExample1.m ')
5 disp(' ----- ')
6
7 % objective function
8 F = @(x) -0.8*exp(-(x-2).^2) - 0.5*exp(-(x+2).^2)+1;
9
10 % bound for design variables
11 lb = -5;
12 ub = 5;
13
14 % initialize mean and std. dev. vectors
15 mu0 = (ub+lb)/2;
16 sigma0 = (ub-lb)/6;
17
18 % CE optimizer
19 tic
20 [Xopt,Fopt,ExitFlag,CEobj] = CEopt(F,mu0,sigma0,lb,ub)
21 toc

```

Figure 5 provides a detailed visualization of the optimization process. It shows not only the Gaussian mixture function but also the sequence of Gaussian distributions employed by the CE method. The adaptive nature of this sampling is clearly visible in the transition from broad, orange-colored distributions early in the search process to darker, nearly black distributions as the search converges towards the optimum. The focused distributions towards the end of the search effectively pinpoint the location of the minimum, marked by a black dot on the plot.

The inset within Figure 5 further magnifies the final stages of this optimization process, illustrating the precision with which the CE method narrows down the search distributions. This detailed view highlights the method's ability to finely tune the sampling strategy as it homes in on the optimum, showcasing the dynamic adaptation that is central to the CE method's approach.

5.2. Example 2: The Peaks function in 2D

Now we delve into the optimization of a intricate two-dimensional function characterized by

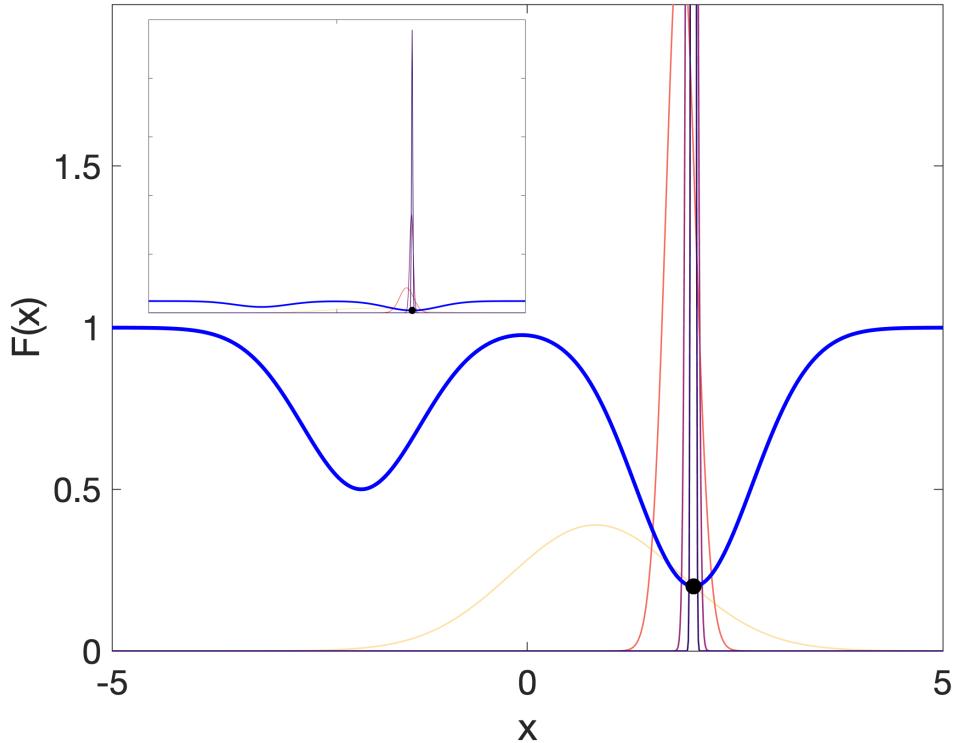


Figure 5: Adaptive Sampling in CE Optimization. The plot showcases the Cross-Entropy method optimizing a one-dimensional Gaussian mixture function (thick blue line). Gaussian distributions, from orange to dark near-black, visualize the adaptive sampling strategy. Early in the search, broader red distributions cover wider areas, while focused near-black distributions towards the end pinpoint the optimum, marked by the black dot. The inset magnifies the final stages, highlighting the precision of the narrowing search distributions.

its complex landscape of multiple local maxima and minima, the Peaks function:

$$\begin{aligned} F(x_1, x_2) = & 3(1 - x_1)^2 \exp\left(-x_1^2 - (x_2 + 1)^2\right) \\ & - 10 \left(\frac{x_1}{5} - x_1^3 - x_2^5\right) \exp\left(-x_1^2 - x_2^2\right) - \frac{1}{3} \exp\left(-(x_1 + 1)^2 - x_2^2\right). \end{aligned} \quad (40)$$

This function presents a challenge for many optimization algorithms due to its convoluted surface features, demanding robust strategies for navigation and convergence.

For the optimization process, the function $F(x_1, x_2)$ is implemented in MATLAB as an auxiliary function and transformed into a function handle, a standard procedure for managing complex objective functions with **CEopt**. The following MATLAB code demonstrates how users can customize the parameters of the CE method via an auxiliary structure. This customization enables users to adjust aspects such as iteration count, sample size, and elite sample proportion, adapting the optimization process to their specific requirements. Initial conditions for the mean and standard deviation are set randomly within a broad range to ensure comprehensive exploration of the potential landscape.

`MainCEoptExample2.m`

```

1 clc; clear; close all;
2
3 disp(' ----- ')
4 disp(' MainCEoptExample2.m ')
5 disp(' ----- ')
6
7 % objective function
8 F = @(x)PeaksFunc(x);
9
10 % bound for design variables
11 lb = [-3; -3];
12 ub = [ 3;  3];
13
14 % initialize mean and std. dev. vectors
15 mu0 = lb + (ub-lb).*rand(2,1);
16 sigma0 = 5*(ub-lb);
17
18 % define parameters for the CE optimizer
19 CEstr.isVectorized = 1;           % Vectorized function
20 CEstr.EliteFactor = 0.1;          % Elite samples percentage
21 CEstr.Nsamp = 50;                % Number of samples
22 CEstr.MaxIter = 80;              % Maximum of iterations
23 CEstr.TolAbs = 1.0e-2;           % Absolute tolerance
24 CEstr.TolRel = 1.0e-2;           % Relative tolerance
25 CEstr.alpha = 0.7;               % Smoothing parameter
26 CEstr.beta = 0.8;                % Smoothing parameter
27 CEstr.q = 10;                   % Smoothing parameter
28
29 % CE optimizer
30 tic
31 [Xopt,Fopt,ExitFlag,CEstr] = CEopt(F,mu0,sigma0,lb,ub,[],CEstr)
32 toc
33
34 % objective function
35 function F = PeaksFunc(x)
36     x1 = x(:,1);
37     x2 = x(:,2);
38     F = 3*(1-x1).^2.*exp(-x1.^2 - (x2+1).^2) ...
39         - 10*(x1/5 - x1.^3 - x2.^5).*exp(-x1.^2 - x2.^2) ...
40         - (1/3)*exp(-(x1+1).^2 - x2.^2);
41 end

```

The CE algorithm initiates with a broad search area, indicated by the initial approximation settings, and methodically narrows this area under the guidance of elite samples. Figure 6 visually represents the trajectory of the CE iterations as they converge towards the global optimum. The iterations, marked by red diamonds, illustrate the positions sampled by CE method, providing a visual narrative of the algorithm's exploration and exploitation of the landscape. The ultimate convergence to the global optimum, marked by a red cross, underscores the efficacy of the CE method in navigating through and distinguishing between multiple peaks to identify the best solution.

5.3. Example 3: A collection of 2D algebraic benchmark problems

This section evaluates the **CEopt** solver's efficacy across a diverse set of 24 two-dimensional

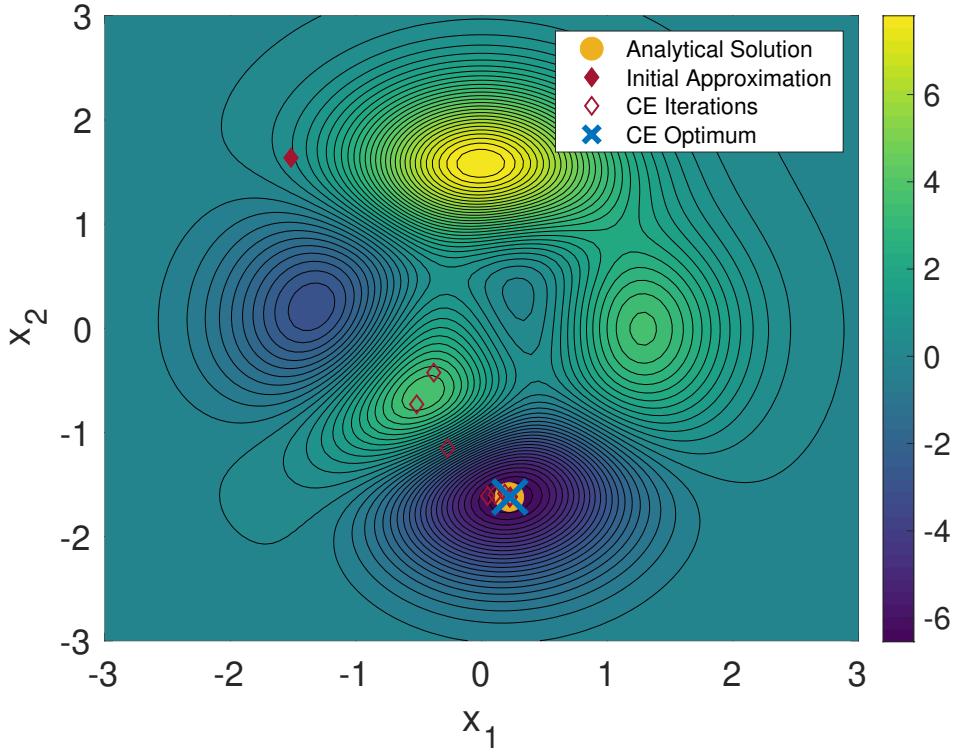


Figure 6: Visualization of the Peaks Function Optimization. The color gradient from blue to yellow represents function values from low to high. Key points include the initial approximation (full red diamond), the trajectory of CEopt iterations (open red diamonds), the final CE solution (red cross), and the analytical solution (yellow circle).

benchmark problems listed in Table 1. These benchmarks, known for their complex landscapes and intricate topological features, rigorously test the solver beyond the conventional limits of convex optimization methods. They encompass a wide range of mathematical models, each designed to challenge optimization algorithms in unique ways. Functions such as Ackley and Rastrigin test the solver’s global search capabilities through their multi-modal landscapes, while the steep gradients and narrow valleys of the Rosenbrock and Dixon-Price functions probe its precision and convergence in more confined spaces.

The optimization trajectories and final outcomes are comprehensively depicted in Figures 7 and 8, which illustrate the **CEopt** solver’s path through complex optimization landscapes. These figures provide a visual account of the solver’s strategy, highlighting its dynamic adaptability and robust performance across varied challenges. **CEopt** demonstrated exceptional performance, reliably achieving or closely approximating global optima across most benchmarks. The solver showcased notable resilience against deceptive local minima, particularly in complex landscapes like the Eggholder and Cross-in-Tray, where it effectively navigated through intricate terrain to locate near-optimal solutions. While not every run achieved the global optimum due to inherent landscape challenges, the results were consistently near-optimal, underscoring the solver’s capability to yield high-quality solutions.

Table 1: Benchmark functions for unconstrained optimization testing. The functions range from simple unimodal landscapes to complex multimodal challenges, representing typical problems encountered in various scientific and engineering disciplines.

| Function Name | Function Formula $F(x_1, x_2)$ | Domain for (x_1, x_2) |
|------------------|--|--|
| Ackley | $-20 \exp\left(-0.2\sqrt{0.5(x_1^2 + x_2^2)}\right) - \exp(0.5(\cos 2\pi x_1 + \cos 2\pi x_2)) + 20 + e$ | $[-32.768, 32.768] \times [-32.768, 32.768]$ |
| Beale | $(1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2$ | $[-4.5, 4.5] \times [-4.5, 4.5]$ |
| Booth | $(x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$ | $[-10, 10] \times [-10, 10]$ |
| Bukin N.6 | $100\sqrt{ x_2 - 0.01x_1^2 } + 0.01 x_1 + 10 $ | $[-15, -5] \times [-3, 3]$ |
| Cross-in-tray | $-0.0001 \left(\left \sin(x_1) \sin(x_2) \exp\left(100 - \frac{\sqrt{x_1^2 + x_2^2}}{\pi} \right) \right + 1 \right)^{0.1}$ | $[-10, 10] \times [-10, 10]$ |
| Dixon-Price | $(x_1 - 1)^2 + 2(2x_2^2 - x_1)^2$ | $[-10, 10] \times [-10, 10]$ |
| Easom | $-\cos(x_1) \cos(x_2) \exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2)$ | $[-100, 100] \times [-100, 100]$ |
| Eggholder | $-(x_2 + 47) \sin(\sqrt{ x_2 + \frac{x_1}{2} + 47 }) - x_1 \sin(\sqrt{ x_1 - (x_2 + 47) })$ | $[-512, 512] \times [-512, 512]$ |
| Goldstein-Price | $(1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1 x_2 + 3x_2^2)) \times (30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1 x_2 + 27x_2^2))$ | $[-2, 2] \times [-2, 2]$ |
| Griewank | $1 + \sum_{i=1}^2 \frac{x_i^2}{4000} - \prod_{i=1}^2 \cos\left(\frac{x_i}{\sqrt{i}}\right)$ | $[-600, 600] \times [-600, 600]$ |
| Himmelblau's | $(x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$ | $[-5, 5] \times [-5, 5]$ |
| Holder Table | $- \sin(x_1) \cos(x_2) \exp(1 - \sqrt{x_1^2 + x_2^2}/\pi) $ | $[-10, 10] \times [-10, 10]$ |
| Lévi N.13 | $\sin^2(3\pi x_1) + (x_1 - 1)^2(1 + \sin^2(3\pi x_2)) + (x_2 - 1)^2(1 + \sin^2(2\pi x_2))$ | $[-10, 10] \times [-10, 10]$ |
| Matyas | $0.26(x_1^2 + x_2^2) - 0.48x_1 x_2$ | $[-10, 10] \times [-10, 10]$ |
| McCormick | $\sin(x_1 + x_2) + (x_1 - x_2)^2 - 1.5x_1 + 2.5x_2 + 1$ | $[-1.5, 4] \times [-3, 3]$ |
| Rastrigin | $10 \times 2 + \sum_{i=1}^2 [x_i^2 - 10 \cos(2\pi x_i)]$ | $[-5.12, 5.12] \times [-5.12, 5.12]$ |
| Rosenbrock | $\sum_{i=1}^{2-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$ | $[-5, 10] \times [-5, 10]$ |
| Schaffer N.2 | $0.5 + \frac{\sin^2(x_1^2 - x_2^2) - 0.5}{[1 + 0.001(x_1^2 + x_2^2)]^2}$ | $[-100, 100] \times [-100, 100]$ |
| Schaffer N.4 | $0.5 + \frac{\cos^2(\sin x_1^2 - x_2^2) - 0.5}{[1 + 0.001(x_1^2 + x_2^2)]^2}$ | $[-100, 100] \times [-100, 100]$ |
| Shekel | $-\sum_{i=1}^{10} \frac{1}{c_i + \sum_{j=1}^2 (x_j - A_{ij})^2}$ | $[0, 10] \times [0, 10]$ |
| Sphere | $\sum_{i=1}^2 x_i^2$ | $[-5, 5] \times [-5, 5]$ |
| Styblinski-Tang | $\frac{1}{2} \sum_{i=1}^2 (x_i^4 - 16x_i^2 + 5x_i)$ | $[-5, 5] \times [-5, 5]$ |
| Three-hump Camel | $2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1 x_2 + x_2^2$ | $[-5, 5] \times [-5, 5]$ |
| Zakharov | $\sum_{i=1}^2 x_i^2 + \left(\sum_{i=1}^2 0.5 i x_i \right)^2 + \left(\sum_{i=1}^2 0.5 i x_i \right)^4$ | $[-5, 10] \times [-5, 10]$ |

Note: For the Shekel function, matrix $A = \begin{bmatrix} 4 & 1 & 8 & 6 & 3 & 2 & 5 & 8 & 6 & 7 \\ 4 & 1 & 8 & 6 & 3 & 2 & 5 & 8 & 6 & 7 \end{bmatrix}$ and vector $c = \frac{1}{10} \times [1, 2, 2, 4, 4, 6, 3, 7, 5, 5]$ are used.

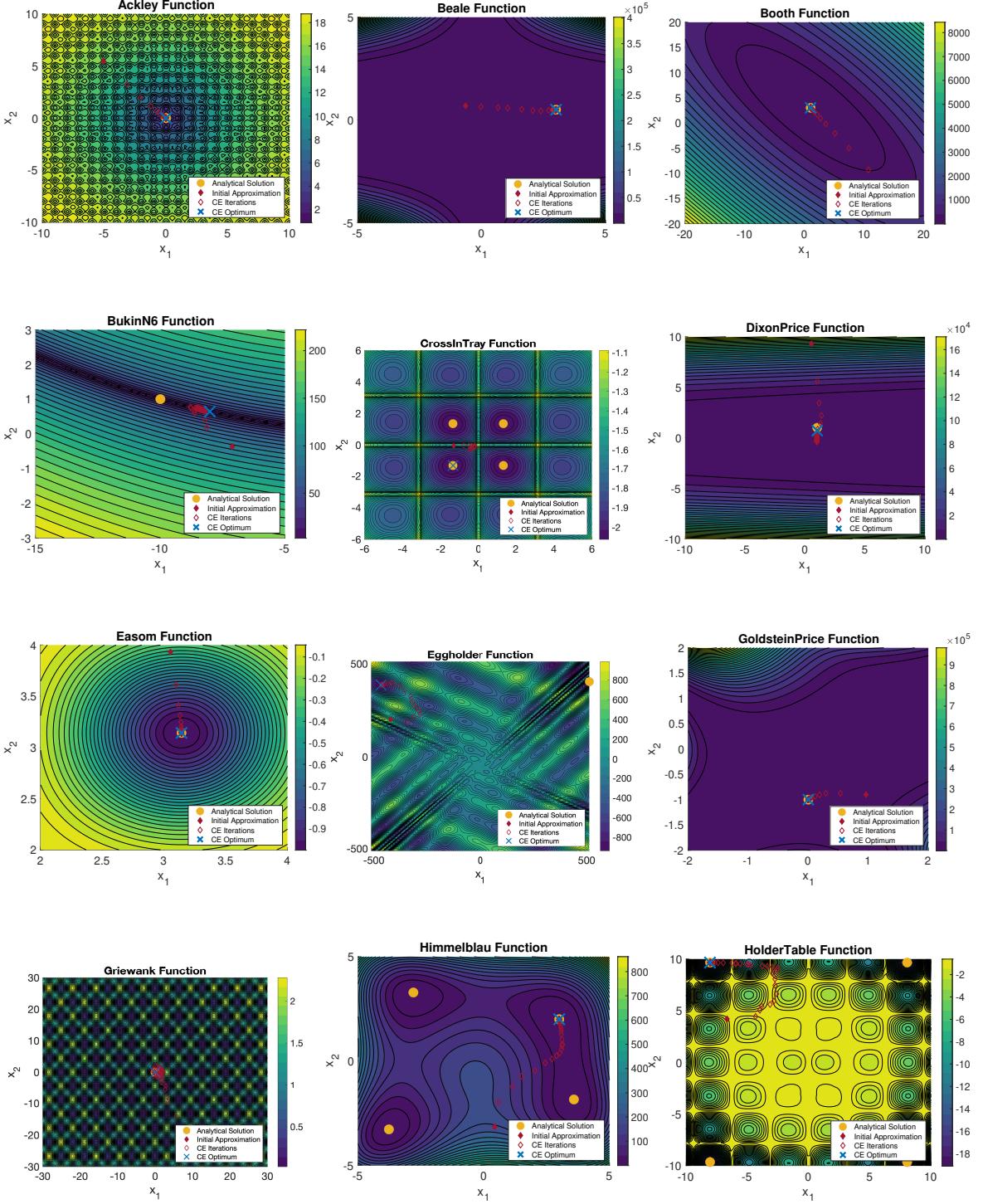


Figure 7: Optimization landscapes and paths for the first set of benchmark functions. Each subplot shows the optimization process, with legends indicating initial points, progression paths, and final solutions, highlighting the solver's effectiveness across diverse landscapes.

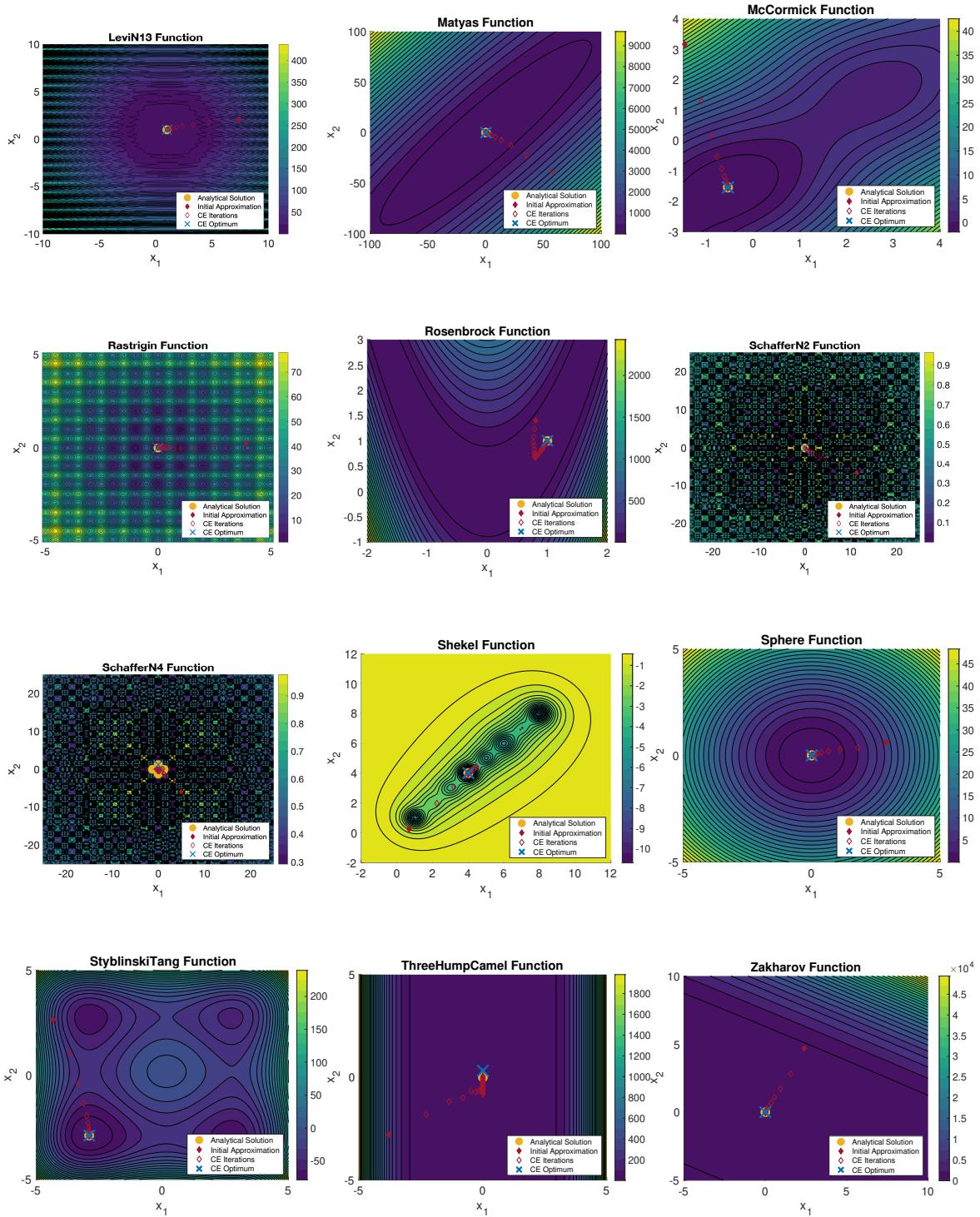


Figure 8: Continuation of optimization landscapes for the second set of benchmark functions. These visualizations demonstrate the solver's adaptability and strategic exploration, with detailed trajectories marking the course to optimal or near-optimal solutions.

In particular, the Bukin N.6 function posed a significant challenge with its steep ravines and a narrow global basin. **COpt** consistently reached near-optimal solutions, demonstrating the importance of stochastic methods in managing functions with extreme gradients and limited areas of attraction. The performance on the Eggholder function shown in Figure 9 further highlighted the solver’s capabilities; despite the complex landscape sometimes leading the solver away from the best optimum, **COpt** maintained a record of the best sampled solutions. This strategic retention of high-quality solutions significantly aids in achieving a superior final outcome, especially in landscapes where the global optimum is surrounded by challenging terrain.

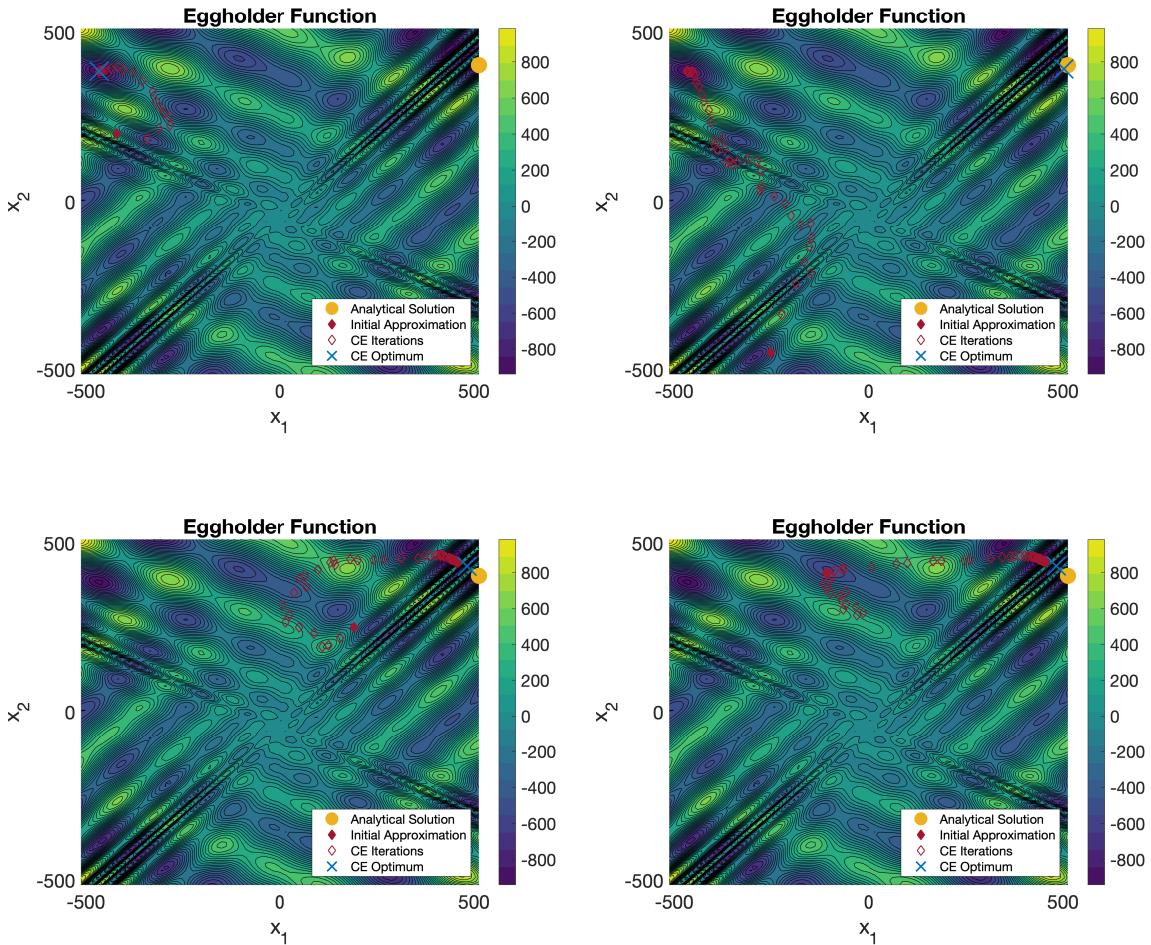


Figure 9: Illustration of multiple optimization runs on the Eggholder function using **COpt**. Each trajectory represents a separate run, highlighting how stochastic variation can lead to different paths towards the optimum. This figure showcases the adaptive nature of **COpt**, with each run potentially exploring new aspects of the landscape and improving the chances of locating the global optimum.

The optimization tests conducted using **COpt** covered a wide array of complex functions, each with its own set of peculiarities and challenges. Several key observations can be made:

1. Global Search Efficiency: For globally complex functions like Ackley and Griewank, which feature numerous local minima, **CEopt** effectively escaped local traps and converged towards global minima. This underscores the solver's ability to perform global searches efficiently, a crucial attribute for solving real-world problems where the landscape is often unpredictable and riddled with local optima.
2. Handling Sharp Gradients: In benchmarks like the Dixon-Price and Rosenbrock functions, which are known for their sharp, narrow valleys, **CEopt** demonstrated remarkable precision in navigating tight gradients without overshooting, a common issue in gradient-based methods. This precision is particularly advantageous in engineering applications where small deviations can lead to significantly different outcomes.
3. Robustness in Multimodal Landscapes: The solver's performance on multimodal landscapes such as those presented by the Rastrigin and Styblinski-Tang functions highlighted its robustness. Even in the face of severe oscillations and multiple local optima, **CEopt** managed to identify areas of interest and refine its search towards the optimum.
4. Specialized Challenges: Certain functions like the Shekel and Eggholder presented specialized challenges due to their peculiar landscapes. For instance, the Shekel function, with its dispersed peaks, tested the solver's ability to discern between closely spaced optima, while the Eggholder function, with its disjointed and irregular surface, required the solver to maintain a balance between exploration and exploitation.

The diversity of these tests not only demonstrates the solver's adaptability and efficiency across various types of optimization problems but also showcases its potential for application in a wide range of disciplines, from financial modeling to bioinformatics.

Due to space constraints, detailed MATLAB implementation for this test is not included in this document. However, the code is available on GitHub, file `MainCEoptExample3Ext.m`.

The subsequent examples will feature objective functions and/or constraints defined not by algebraic formulas but through alternative representations, providing an additional layer of complexity and challenge for the **CEopt** solver to navigate.

5.4. Example 4: Identification of a harmonic oscillator

This example demonstrates the application of the **CEopt** solver to the system identification of a damped harmonic oscillator without external forcing. The task is to estimate parameters of the second-order ordinary differential equation (ODE) that models the physical behavior of the system.

The harmonic oscillator is described by the ODE

$$\ddot{y}(t) + 2\zeta\omega_n\dot{y}(t) + \omega_n^2 y(t) = 0, \quad (41)$$

where ω_n is the natural frequency, and ζ is the damping ratio. The analytic solution, assuming initial conditions $y(0) = y_0$ and $\dot{y}(0) = v_0$, is

$$y(t) = A e^{-\zeta\omega_n t} \sin(\omega_d t + \phi), \quad (42)$$

with $\omega_d = \omega_n \sqrt{1 - \zeta^2}$, $A = \sqrt{y_0^2 + \left(\frac{v_0 + \zeta\omega_n y_0}{\omega_d}\right)^2}$, and $\phi = \tan^{-1}\left(\frac{y_0\omega_d}{v_0 + \zeta\omega_n y_0}\right)$.

Synthetic data for system identification is generated by computing the analytical solution and adding Gaussian noise, representing measurement errors. This provides realistic data, lumped into the vector $\mathbf{y}_{\text{data}} \in \mathbb{R}^N$, for the parameter estimation process. In this way, the goal is to minimize the objective function given by the discrepancy between the observed noisy data and the model's response, numerically computed using MATLAB's `ode45` solver

$$F(\mathbf{x}) = \frac{1}{\sqrt{N}} \|\mathbf{y}_{\text{data}} - \mathbf{y}_{\text{model}}(\mathbf{x})\|, \quad (43)$$

where $\mathbf{x} = [\omega_n, \zeta, y_0, v_0]$ represents the parameters to be estimated, and $\mathbf{y}_{\text{model}}(\mathbf{x})$ is the response generated by numerically solving the ODE with these parameters for N time-steps. This example serves as a typical scenario in physics and engineering, where models are calibrated against experimental data. The test case demonstrates **CEopt**'s utility in systems where direct analytical modeling is not feasible, emphasizing its strength in handling complex systems described by differential equations.

The Figure 10 captures the system identification process at different iterations. These visualizations highlight the solver's ability to refine its estimates and converge toward the model that best fits the noisy data. Starting with initial guesses far from the true system dynamics, the solver iteratively adjusts the parameters, effectively reducing the discrepancy between the computed model and the observed data. This is evident in the trajectory from initial to final guesses, showing marked improvements in alignment with the real system's behavior.

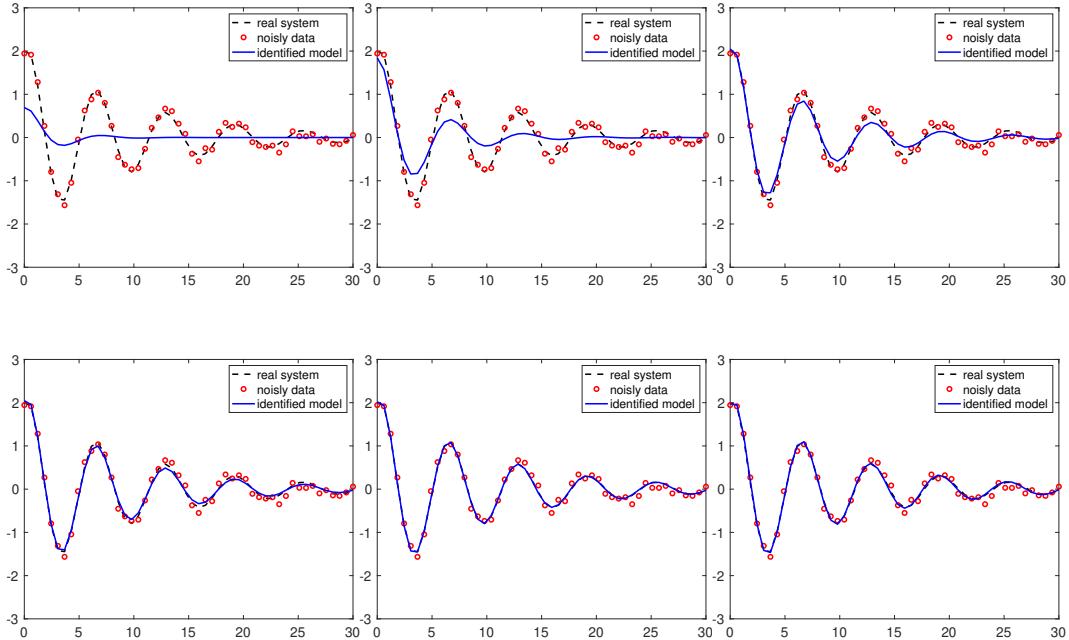


Figure 10: Progressive calibration of a damped harmonic oscillator using **CEopt**. This sequence shows the solver's iterations from initial guesses to refined solutions. Subfigures compare the ground truth (black dashed line), the noisy data (red circles), and the identified model (blue solid line) at various stages.

The MATLAB code below sets up the optimization problem by defining the objective function as the misfit between the numerical solution of the ODE, computed using MATLAB's `ode45` solver, and the noisy data. It uses the **C***E***opt** algorithm to guide the optimization process, aiming to minimize this misfit by adjusting the system parameters.

```

MainCEoptExample4.m
1 clc; clear; close all;
2
3 disp(' ----- ')
4 disp(' MainCEoptExample4.m ')
5 disp(' ----- ')
6
7 % system response observations
8 wn = 1; ksi = 0.1; y0 = 2; v0 = 0.5;
9 t0 = 0.0; t1 = 30.0; Ndata = 50;
10 time = linspace(t0,t1,Ndata)';
11 w = wn*sqrt(1-ksi^2);
12 A = sqrt(y0^2 + ((v0+ksi*wn*y0)/w)^2);
13 phi = atan(y0*w/(v0+ksi*wn*y0));
14 ytrue = A*exp(-ksi*wn*time).*sin(w*time+phi);
15 ydata = ytrue + 0.05*y0*randn(Ndata,1);
16
17 % objective function (minimize the discrepancy)
18 F = @(x) MyMisfitFunc(x,ydata,time);
19
20 % bound for design variables
21 lb = [0.0; 0.0; -3; -3];
22 ub = [2.0; 1.0; 3; 3];
23
24 % define parameters for the CE optimizer
25 CEstr.Nsamp = 50;      % Number of samples
26 CEstr.TolAbs = 1.0e-3; % Absolute tolerance
27 CEstr.TolRel = 1.0e-2; % Relative tolerance
28
29 % CE optimizer
30 tic
31 [Xopt,Fopt,ExitFlag,CEobj] = CEopt(F,[],[],lb,ub,[],CEstr)
32 toc
33
34 % Misfit function
35 function J = MyMisfitFunc(x,ydata,tspan)
36     [Ns,Nvars] = size(x);      % input dimensions
37     J = zeros(Ns,1); % preallocate memory for J
38     wn = x(:,1);        % model parameter 1
39     ksi = x(:,2);        % model parameter 2
40     y0 = x(:,3);        % model parameter 3
41     v0 = x(:,4);        % model parameter 4
42     for n = 1:Ns
43         dydt = @(t,y) [0 1; -wn(n)^2 -2*ksi(n)*wn(n)]*y;
44         [time,ymodel] = ode45(dydt,tspan,[y0(n) v0(n)]);
45         J(n) = norm(ydata-ymodel(:,1))/sqrt(length(ydata));
46     end
47 end

```

This example not only validates the effectiveness of **CEopt** in complex parameter estimation tasks but also demonstrates the solver's potential in applications requiring robust system identification. Given the stochastic nature of **CEopt**, different runs may explore various aspects of the solution space, enhancing the likelihood of approaching the global optimum—a desirable feature in real-world applications where experimental uncertainties and modeling inaccuracies must be managed effectively. Other similar applications can be seen in (Dantas *et al.* 2019a,b; Raqueti *et al.* 2023).

5.5. Example 5: Design of a complex mechanism

This example illustrates the application of the **CEopt** solver to the dimensional synthesis of a four-bar mechanism like the one shown in Figure 13, where one of the coupler points follows a predefined curve represented by coordinates (x_k, y_k) , $k = 1, \dots, N$ (shown as blue dots). This type of task is a good test for CE's capabilities in mechanical design optimization.

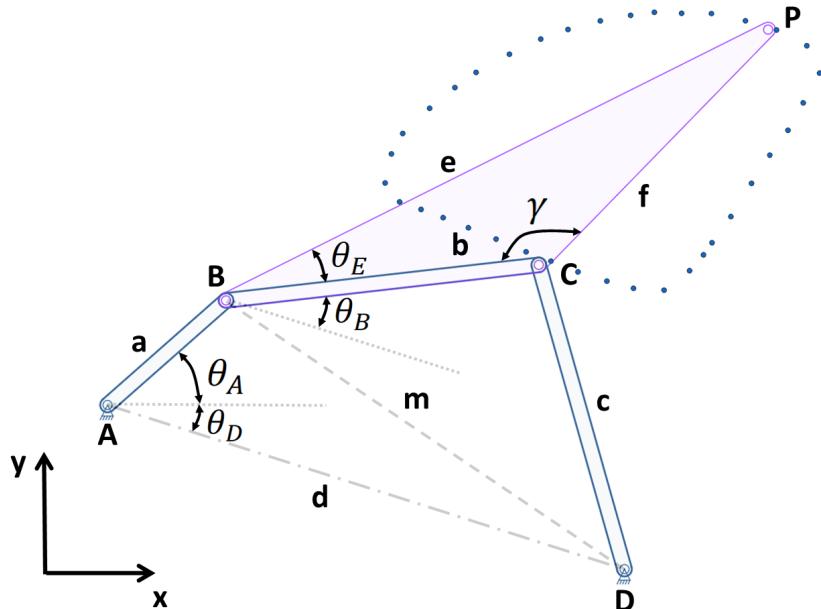


Figure 11: Geometric diagram of a four-bar linkage mechanism. The figure displays the positions of the linkages and the specific trajectory (blue dots) that the coupler point P is designed to follow. It shows the lengths a , b , c , and d , as well as the distances e and f from pivots B and C to point P , respectively. Angles θ_A , θ_B , θ_D , θ_E , and γ are also depicted. The shaded region indicates the range of motion of point P .

Figure 11 depicts the geometric configuration of the four-bar mechanism, where a (crank link), b (coupling link), c (follower link), and d (ground link) are the lengths of the links. The position of the coupler point P is determined by the length e and angle θ_E . The ground link AD has its orientation relative to the x -axis given by angle θ_D . The length of the edge BD is denoted by m . The coordinates of fixed joint A are denoted as (x_A, y_A) . Given the symmetry of the curve, the lengths of links b , c and f are set equal, and from the geometry

we see that

$$e = 2 b \cos \theta_E . \quad (44)$$

To simplify the parameter space, the length of link a is fixed at $a = 1$ without any loss of generality. The coordinates of point P are derived using

$$x_P = x_A + a \cos(\theta_A + \theta_0 + \theta_D) + e \cos(\theta_B + \theta_E + \theta_D) , \quad (45)$$

and

$$y_P = y_A + a \sin(\theta_A + \theta_0 + \theta_D) + e \sin(\theta_B + \theta_E + \theta_D) , \quad (46)$$

where

$$\theta_E = \frac{\pi - \gamma}{2} , \quad (47)$$

$$\theta_B = \cos^{-1} \left(\frac{b^2 + m^2 - c^2}{2 b m} \right) - \beta , \quad (48)$$

$$\beta = \sin^{-1} \left(\frac{a \sin \theta_A}{m} \right) , \quad (49)$$

and

$$m = \sqrt{a^2 + d^2 - 2 a d \cos \theta_A} , \quad (50)$$

being θ_0 an initial angular inclination.

Normalization is applied to the reference and calculated curves to eliminate the dependence on the unknowns x_A and y_A so that

$$\hat{x}_i = \frac{x_k - \bar{x}}{\max(x_k) - \min(x_k)} , \quad (51)$$

$$\hat{y}_i = \frac{y_k - \bar{y}}{\max(y_k) - \min(y_k)} , \quad (52)$$

$$\hat{x}_{P_k} = \frac{x_{P_k} - \bar{x}_P}{\max(x_{P_k}) - \min(x_{P_k})} , \quad (53)$$

and

$$\hat{y}_{P_k} = \frac{y_{P_k} - \bar{y}_P}{\max(y_{P_k}) - \min(y_{P_k})} , \quad (54)$$

for $k = 1, \dots, N$, with the upper bar denoting the ensemble average.

The design problem seeks to minimize an objective function that combines absolute values of the differences of the normalized coordinates

$$F(\mathbf{x}) = \sum_{k=1}^N (| \hat{x}_k - \hat{x}_{P_k} | + | \hat{y}_k - \hat{y}_{P_k} |) , \quad (55)$$

where $\mathbf{x} = [b, d, \gamma, \theta_D, \theta_A]$ represents the design variables. This choice of objective function tries to minimize the influence of measurement errors on the final design, once 1-norm is less sensitive to outliers than 2-norm (Boyd and Vandenberghe 2018).

The Figure 12 provides a visual evaluation of the **CEopt** solver's effectiveness in the dimensional synthesis of a four-bar mechanism. It plots the target trajectory (depicted in blue) against the trajectory (shown in red) achieved through optimization. The close alignment of the two curves demonstrates the capability of **CEopt** to accurately configure the mechanism's parameters to meet the design specifications, thereby validating the optimization's success in achieving a functional and precise mechanical design.

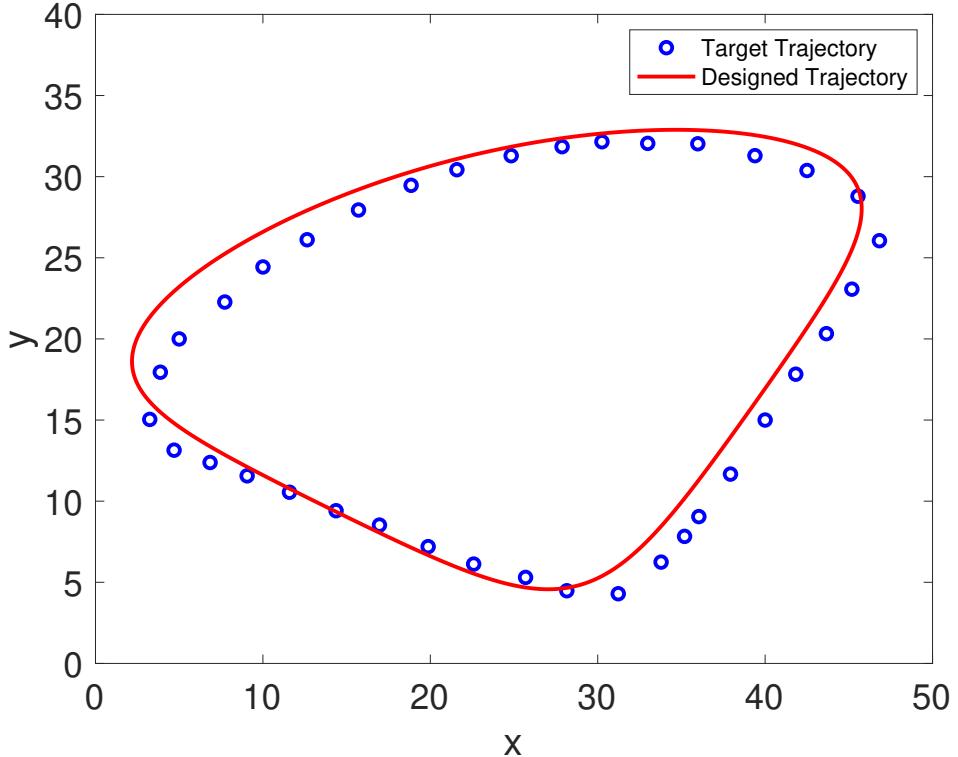


Figure 12: Comparison of target and designed trajectories for coupler point P . This figure shows the desired trajectory (blue dots) that the coupler point P is expected to follow and the trajectory achieved by the optimized four-bar mechanism (red line) using the **CEopt** solver.

For sake of space limitation, the MATLAB implementation for this example is not included in this document. Interested readers can access the complete source code on GitHub, available under the filename `MainCEoptExample5Ext.m`.

This example serves as a robust demonstration of **CEopt**'s application in mechanical systems where design parameters must be optimized to achieve specific dynamic behaviors, showcasing the solver's utility in precision design tasks where analytical solutions are infeasible.

5.6. Example 6: Nonsmooth function with conic constraints

This example introduces the application of the **CEopt** solver to an optimization problem involving a nonsmooth function subject to conic constraints, marking the first instance in this series where constraints are explicitly considered. The objective function and constraints are formulated to test the solver's efficacy in handling lack of differentiability and complex constraint boundaries, characteristics typical of many real-world optimization problems.

```

MainCExptExample6.m
1 clc; clear; close all;
2
3 disp(' ----- ')
4 disp(' MainCExptExample6.m ')
5 disp(' ----- ')
6
7 % objective function and constraints
8 F = @PatternSearchFunc;
9 nonlcon = @ConicConstraints;
10
11 % bound for design variables
12 lb = [-6 -4];
13 ub = [ 2 4];
14 mu0 = [-4 2];
15
16 % cross-entropy optimizer struct
17 CEstr.isVectorized = 1;           % vectorized function
18 CEstr.TolCon        = 1.0e-6;    % relative tolerance
19
20 tic
21 [Xopt,Fopt,ExitFlag,CEstr] = CExpt(F,[],[],lb,ub,nonlcon,CEstr)
22 toc
23
24 % objective function
25 function F = PatternSearchFunc(x)
26     x1 = x(:,1);
27     x2 = x(:,2);
28     F = zeros(size(x1,1),1);
29     for i = 1:size(x,1)
30         if x1(i) < -5
31             F(i) = (x1(i)+5).^2 + abs(x2(i));
32         elseif x1(i) < -3
33             F(i) = -2*sin(x1(i)) + abs(x2(i));
34         elseif x1(i) < 0
35             F(i) = 0.5*x1(i) + 2 + abs(x2(i));
36         elseif x1 >= 0
37             F(i) = .3*sqrt(x1(i)) + 5/2 + abs(x2(i));
38         end
39     end
40 end
41
42 % equality and inequality constraints
43 function [G,H] = ConicConstraints(x)
44     x1 = x(:,1);
45     x2 = x(:,2);
46     G = 2*x1.^2 + x2.^2 - 3;
47     H = (x1+1).^2 - (x2/2).^4;
48 end

```

The objective function for this example is defined as

$$F(x_1, x_2) = \begin{cases} (x_1 + 5)^2 + |x_2| & \text{if } x_1 < -5, \\ -2 \sin(x_1) + |x_2| & \text{if } -5 \leq x_1 < -3, \\ 0.5 x_1 + 2 + |x_2| & \text{if } -3 \leq x_1 < 0, \\ 0.3 \sqrt{x_1} + 2.5 + |x_2| & \text{if } x_1 \geq 0. \end{cases} \quad (56)$$

The constraints include one inequality and one equality conic constraint given by

$$G(x_1, x_2) = 2x_1^2 + x_2^2 - 3 \leq 0, \quad (57)$$

$$H(x_1, x_2) = (x_1 + 1)^2 - \left(\frac{x_2}{2}\right)^4 = 0. \quad (58)$$

These constraints define a feasible region that is both non-convex and challenging due to the presence of an equality constraint that introduces a higher-order nonlinearity.

The MATLAB code, provided in `MainCEoptExample6.m` and available on GitHub, implements the **CEopt** solver to minimize the objective function while satisfying the constraints.

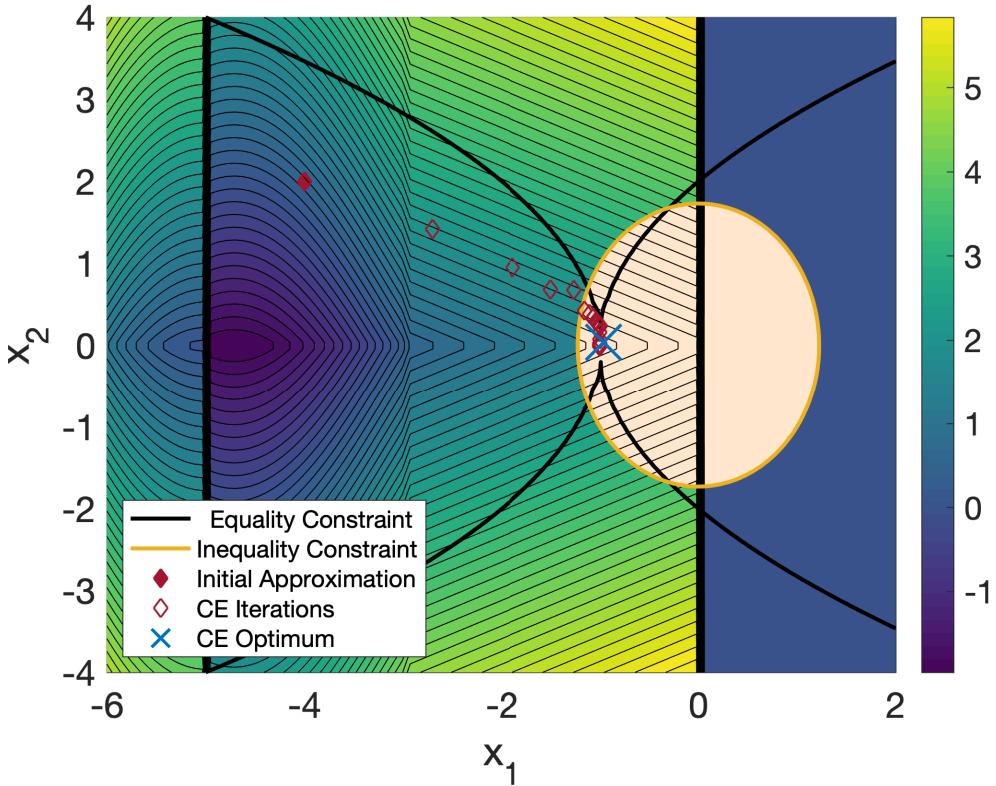


Figure 13: Optimization landscape for the nonsmooth function with conic constraints. The contour map depicts the function's nonsmooth behavior, while the shaded regions represent the feasible area defined by the inequality constraint. The lines and markers indicate the levels of the equality constraint and the optimal solution found by **CEopt**, respectively. This visualization highlights the solver's capability to navigate complex constraint geometries and identify solutions near the boundary of the feasible region.

Figure 13 shows the optimization landscape for this nonsmooth function with clear demarcations of the feasible regions and constraint boundaries. The contour lines provide insights into the function's complexity and its multiple discontinuities. The shaded areas depict the regions where the inequality constraints are satisfied, which are further validated by the equality constraint lines intersecting these regions. The plot visually demonstrates how **Ceopt** efficiently navigates through these constraints to locate an optimal solution that lies at the intersection of these constraints, showcasing its effectiveness in managing complex constraint interactions. This is particularly notable as the optimal solution, marked on the plot, is found near the challenging boundary regions, where the constraints converge.

This example not only tests the solver's capability to handle nonsmooth landscapes and complex constraints but also serves as a benchmark for comparing the **Ceopt**'s performance with MATLAB's traditional genetic algorithms (**ga**), which are frequently employed for similar types of optimization challenges. To facilitate a comprehensive evaluation, try to solve the same problem using the **ga** command in MATLAB and compare the results obtained with those from the provided code. This comparative analysis can highlight the efficiency, accuracy, and computational demands of **Ceopt** relative to **ga**, offering valuable insights into their respective strengths and limitations in handling complex optimization scenarios.

5.7. Example 7: Non-convex structural optimization

This example demonstrates the application of the **Ceopt** solver to optimize a ten-bar truss system, Figure 14, aiming to minimize the structure's weight by adjusting the cross-sectional areas of the bars. The optimization incorporates constraints to ensure the first three natural frequencies exceed specified thresholds to prevent resonance and ensure dynamic stability.

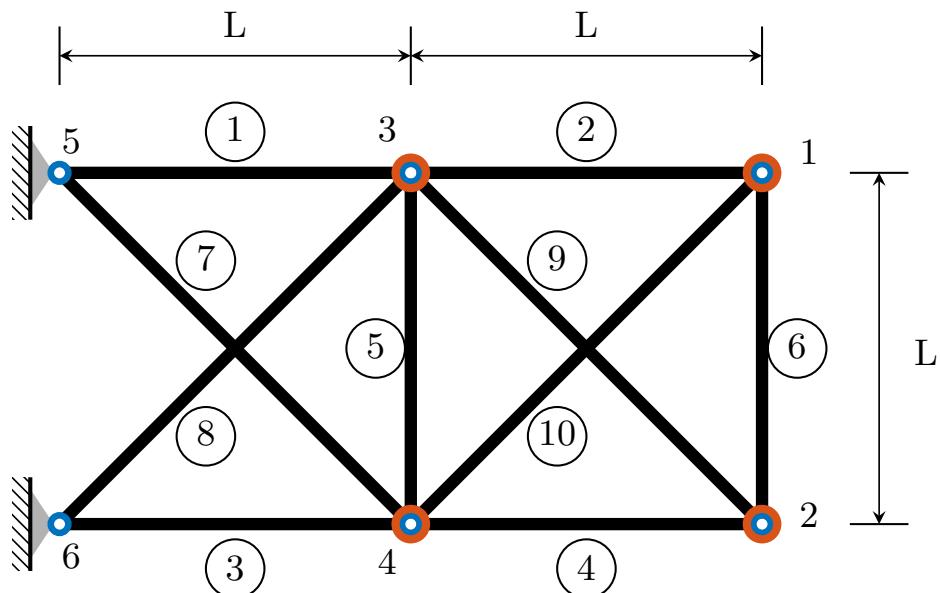


Figure 14: Schematic of the ten-bar truss structure, showing the arrangement and labeling of bars, with dimensions and load conditions specified for the optimization.

The eigenvalue problem for determining natural frequencies is expressed as

$$[\mathbf{K}] \boldsymbol{\phi}_i = \omega_i^2 [\mathbf{M}] \boldsymbol{\phi}_i, \quad (59)$$

where $[\mathbf{K}]$ represents the stiffness matrix, $[\mathbf{M}]$ is the mass matrix, $\boldsymbol{\phi}_i$ the i -th mode shape, and ω_i the i -th natural frequency. The constraints on the frequencies, to ensure they meet the dynamic stability criteria, are formulated as

$$\omega_i \geq \omega_i^\dagger \iff 1 - \frac{\omega_i}{\omega_i^\dagger} \leq 0, \quad (60)$$

with the given threshold values ω_i^\dagger .

The MATLAB code implementing this optimization, `MainCEoptExample7Ext.m`, is available on GitHub. This code sets up the optimization parameters, defines the objective function and constraints, and invokes **CEopt** to solve the problem.

Figure 15 provides a compelling visualization of the optimization outcomes for the ten-bar truss system, highlighting the transformation from the initial configuration to the optimized structure. This visualization demonstrates the optimization's impact, showing a significant reduction in material usage without compromising structural integrity. The optimized configuration achieves improved structural efficiency by redistributing material where it contributes most to load-bearing capacity, which is evident from the streamlined appearance of the truss. This not only ensures that the structure meets all imposed constraints, including mechanical stability and frequency requirements, but also underscores the potential of **CEopt** to enhance performance in complex engineering designs.

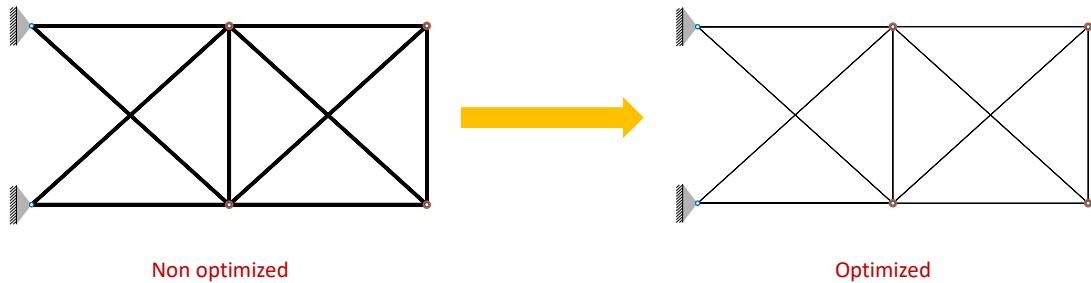


Figure 15: Visualization of optimization results for the ten-bar truss system. This figure contrasts the initial and optimized configurations, emphasizing the reduction in material use while enhancing structural efficiency and meeting dynamic stability requirements.

5.8. Example 8: Fractional-order controller optimal tuning

This example focuses on the optimal tuning of a fractional-order controller for an inverted pendulum, a canonical control system challenge illustrated in Figure 16 left. The inverted pendulum system, characterized by a pole balanced on a moving base, is inherently unstable and nonlinear, presenting a classic test for assessing control strategies. Fractional-order control is employed to enhance precision and adaptability, leveraging the nuanced capabilities of fractional calculus to address complexities traditional controllers may struggle with.

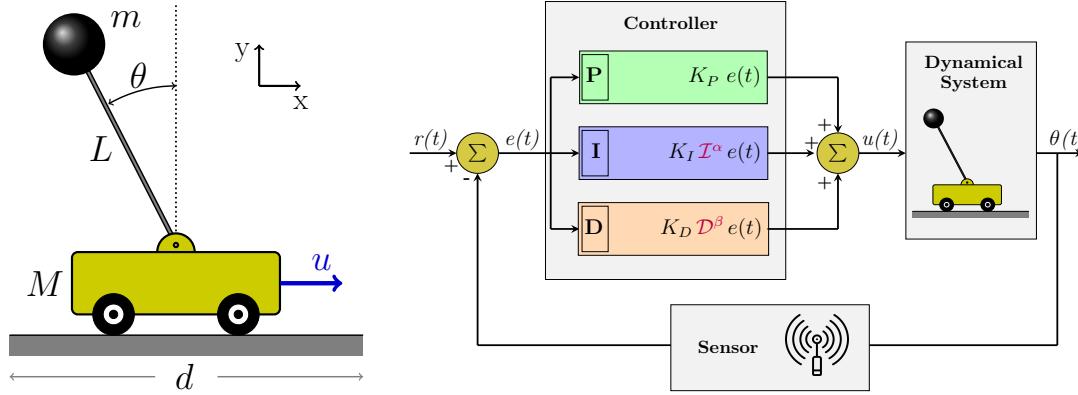


Figure 16: Control system diagram showcasing the components of the fractional-order controller for the inverted pendulum over a moving cart. The diagram illustrates the feedback loop, including the fractional integrator and differentiator, demonstrating how the control action $u(t)$ is computed based on the error signal $e(t)$.

The dynamic behavior of the system is governed by the following differential equations, which describe the motion of the base and the angular movement of the pendulum

$$m L \cos \theta(t) \ddot{x}(t) + (J + m L^2) \ddot{\theta}(t) - m g L \sin \theta(t) = 0, \quad (61)$$

$$(m + M) \ddot{x}(t) + m L \cos \theta(t) \ddot{\theta}(t) - m L \sin \theta(t) \dot{\theta}^2(t) = u(t), \quad (62)$$

where $x(t)$ represents the cart horizontal displacement; $\theta(t)$ is the inverted pendulum angular displacement; the upper dot is an abbreviation for temporal derivative; m is the mass of the pendulum; M is the mass of the cart; L is the length of the pendulum; J is the moment of inertia; g is the acceleration due to gravity, and $u(t)$ is the control force applied to the cart.

The control law employs a fractional-order approach, described by

$$u(t) = K_P e(t) + K_I T^\alpha e(t) + K_D D^\beta e(t), \quad (63)$$

where $e(t) = r(t) - \theta(t)$ is the control error, and T^α and D^β represent the fractional integral and derivative operators, respectively. The controller parameters K_P , K_I , and K_D , as well as the fractional-orders α and β , are tuned to optimize system response.

Fractional calculus introduces flexibility to the controller by accounting for past states of the system (Ortigueira 2011; Ortigueira and Machado 2015; D’Elia, Gulian, Olson, and Karpiadakis 2021), a feature that is vital for handling dynamic complexities like those of an inverted pendulum. The fractional operators are implemented using the FOMCON toolbox (Tepljakov, Petlenkov, and Belikov 2011a,b), using the Riemann-Liouville definition

$$T^\alpha e(t) = \frac{1}{\Gamma(\alpha)} \int_a^t (t - \tau)^{\alpha-1} e(\tau) d\tau, \quad (64)$$

$$D^\beta e(t) = \frac{d^n}{dt^n} \left(\frac{1}{\Gamma(n - \beta)} \int_a^t (t - \tau)^{n-\beta-1} e(\tau) d\tau \right), \quad (65)$$

where Γ denotes the Gamma function.

The control cost function to be minimized for $\mathbf{x} = [K_P, K_I, K_D, \alpha, \beta]$, which emphasizes both performance and energy efficiency, integrates the square of the error and the square of the control signal over time

$$F(\mathbf{x}) = \int_0^t e(\tau)^2 d\tau + \int_0^t \left[\frac{u(\tau)}{100} \right]^2 d\tau, \quad (66)$$

subject to the constraints that cart displacement must remain within specified bounds

$$-d/2 \leq x(t) \leq d/2 \iff |x(t)| - d/2 \leq 0. \quad (67)$$

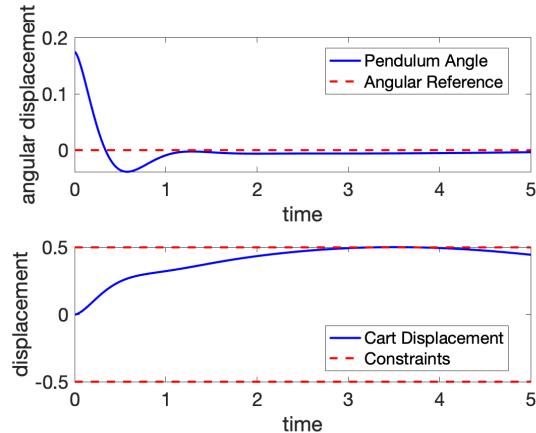


Figure 17: Controlled Dynamics of the Cart-Pendulum System. This figure illustrates the controlled trajectories of the cart position $x(t)$ and the pendulum angle $\theta(t)$ over time, showcasing the effectiveness of the optimized fractional-order controller in stabilizing the inverted pendulum over a moving cart.

Figure 17 displays the dynamic behavior of both the cart position $x(t)$ and the pendulum angle $\theta(t)$ under the influence of an optimized fractional-order controller. The trajectories demonstrate how the controller effectively stabilizes the pendulum in an upright position while managing the horizontal motion of the cart. These results validate the controller's performance in real-time, emphasizing its ability to maintain stability and respond appropriately to the system's inherent instabilities and nonlinear characteristics. The successful regulation of both position and angle highlights the practical applicability and efficiency of **CEopt** to tune the optimized control strategy in a challenging dynamic environment.

The MATLAB code `MainCEoptExample8Ext.m` provided on GitHub facilitates the parameter tuning process using the **CEopt** solver, demonstrating the effectiveness of fractional-order control in complex scenarios. For further details on the applications of fractional-order control, see Basilio, Telles Ribeiro, Cunha Jr, and Oliveira (2019, 2021); Basilio, Oliveira, Telles Ribeiro, and Cunha Jr (2022).

5.9. Example 9: Sparse Identification of Nonlinear Dynamics – SINDy

The final example discusses the application of the Sparse Identification of Nonlinear Dynamics (SINDy) algorithm (Brunton, Proctor, and Kutz 2016; Brunton and Kutz 2022) using the **Ceopt** solver. SINDy is an innovative machine learning method for discovering governing differential equations from time-series data of a system's state variables. This approach is particularly potent in scenarios where the underlying dynamics are complex but sparsely driven by a few active terms in a potentially large system of possible dynamics.

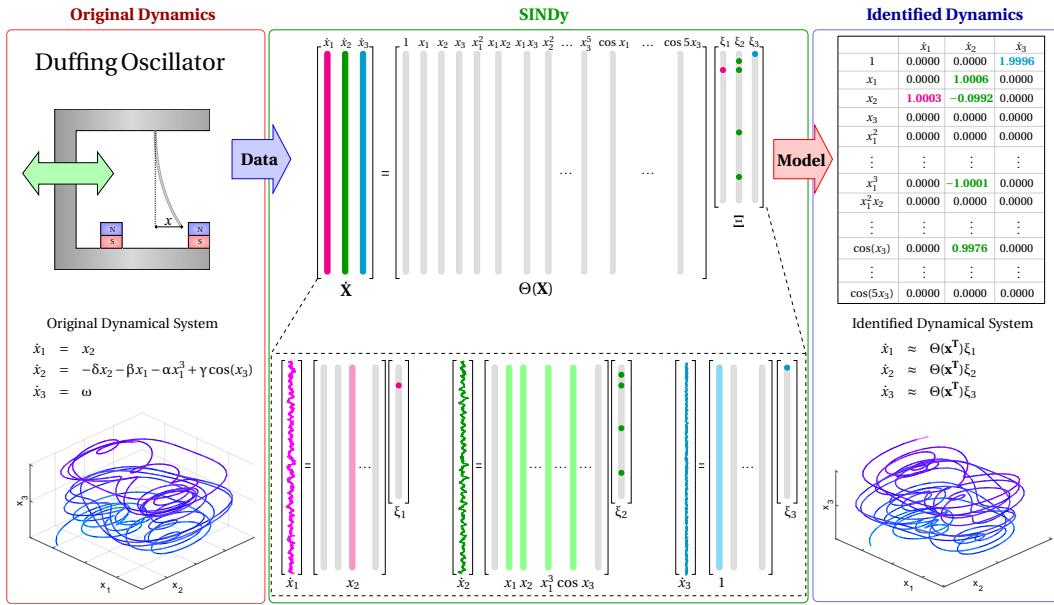


Figure 18: Application of SINDy algorithm to discover the underlying differential equations associated with a given time-series. In the left part one has the original dynamical system equations and a plot of the system's trajectory in the phase space. The middle section shows the library of functions $\Theta(\mathbf{X})$, alongside the coefficients matrix Ξ representing the selected terms that best describe the system dynamics. The right section displays the identified dynamical equations resulting from the SINDy process.

The central challenge in SINDy is to construct an evolutionary equation that accurately captures the dynamics of the system from noisy data. The original method, which is schematically illustrated in Figure 18, involves:

- **Data Collection:** Gather time-series measurements of the system's state variables. In this example, synthetic data is generated using the forced Duffing oscillator, a chaotic system known for its intricate nonlinear dynamics.
- **Library Construction:** From the time-series data, a comprehensive library of potential functions (polynomials, trigonometric functions, logarithms, etc.) is constructed. These functions are candidate terms that may appear in the true governing equations.
- **Regression Problem:** The next step is to solve a regression problem where the coefficients of these functions are determined such that they best fit the observed data. The fit is quantified by how well the derivatives (computed from the data) can be approximated by linear combinations of the library functions.

In this sense, SINDy employs misfit function that balances fidelity to the data with sparsity in the model representation. It combines the mean squared error of the model fitting with a penalty proportional to the number of non-zero coefficients, being formulated as

$$\mathcal{J}(\Xi) = \|\dot{\mathbf{X}} - \Theta(\mathbf{X})\Xi\|_2 + \lambda \|\Xi\|_0, \quad (68)$$

where \mathbf{X} and $\dot{\mathbf{X}}$ are matrices which contain the state variables and their derivatives, respectively, among the columns; $\Theta(\mathbf{X})$ is a dictionary of functions used to reconstruct the vector field² of the underlying dynamical system, with each column representing an independent function; Ξ is the matrix of coefficients for $\Theta(\mathbf{X})$; λ is a regularization parameter that controls the trade-off between model complexity (sparsity) and fit to the data; $\|\cdot\|_0$ denotes the “zero-norm”, which counts the number of non-zero entries, promoting sparsity.

The primary challenge here arises from the use of the “zero-norm” for regularization, promoting sparsity in the coefficients during the regression process. The L^0 count, indicating the number of non-zero coefficients, transforms the optimization problem into a non-convex NP-hard one, notoriously difficult to solve due to the potential for multiple local minima.

Obviously, the CE method may be employed to tackle this non-convex optimization problem, despite not be the standard choice. The usual procedure to handle this problem in machine learning literature is to use convex relaxation techniques, like Sequential Threshold Least-Squares (STLS) (Brunton *et al.* 2016) or Sparse Relaxed Regularized Regression (SR3) (Zheng, Askham, Brunton, Kutz, and Aravkin 2019). Here we decided to use CE method to test its capacity in such a difficult problem. Therefore, **CEopt** solver evolves a population of solutions towards the optimum by iteratively learning from the best performers, making it suitable for finding global optima in complex landscapes posed by L^0 regularization.

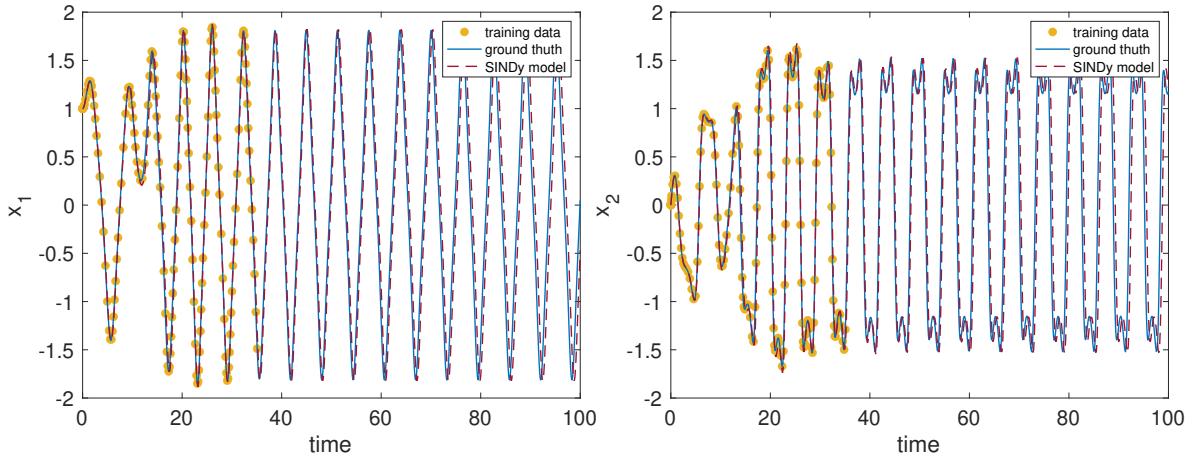


Figure 19: Results of applying the CE-SINDy algorithm to the forced Duffing oscillator. From left to right: state variable x_1 , and state variable x_2 . The plots compare the ground truth, training data, and the SINDy data-driven model predictions.

²Here we assume the dynamics is defined by an autonomous system of differential equations $\dot{\mathbf{x}}(t) = \mathcal{F}(\mathbf{x}(t))$, where $\mathbf{x}(t)$ is a state vector that lumps the state coordinates and the function \mathcal{F} , known as the system vector field, gives the evolution law of the dynamical system.

In this example, synthetic data was generated using the forced Duffing oscillator, which is known for its chaotic and nonlinear dynamics. The parameters used were $\alpha = 1.0$, $\beta = -1.0$, $\delta = 0.3$, $\gamma = 0.65$, and $\omega = 1$. Initial conditions were set to $[x_1(0), x_2(0), x_3(0)] = [1, 0, 0]$, and the temporal interval of analysis was from $t_0 = 0$ to $t_1 = 100$. The ground truth trajectories of the system were obtained using MATLAB's `ode45` function, and noisy training data was generated by sampling the ground truth data and adding Gaussian noise.

The MATLAB code `MainCeoptExample9.m` implements SINDy using **Ceopt** as optimization solver. It constructs the library of functions from the noisy data, sets up the optimization problem with L^0 regularization, and uses **Ceopt** to find the sparsest set of coefficients that can model the data accurately. The code is available on GitHub, for replication and exploration.

Table 2: Identified coefficients of Duffing oscillator obtained via CE-SINDy, employing a dictionary of functions with linear and cubic polynomials, and cosine, as well as $\lambda = 0.25$.

| Dictionary functions | Coefficients | | |
|-------------------------|--------------|----------|----------|
| | $x_1(t)$ | $x_2(t)$ | $x_3(t)$ |
| 1 | -0.0160 | 0.1226 | 0.9947 |
| x_1 | -0.0097 | 0.9841 | -0.0122 |
| x_2 | 0.9910 | -0.2687 | -0.0385 |
| x_3 | -0.0063 | -0.0025 | -0.0039 |
| x_1^3 | -0.0090 | -0.9912 | -0.0048 |
| x_2^3 | -0.0376 | -0.0138 | -0.0089 |
| x_3^3 | -0.0000 | -0.0000 | -0.0000 |
| $\cos x_1$ | -0.0064 | -0.1018 | -0.0196 |
| $\cos x_2$ | -0.0197 | -0.0868 | -0.0105 |
| $\cos x_3$ | -0.0393 | 0.6229 | -0.0075 |

Table 3: Identified coefficients of Duffing oscillator obtained via CE-SINDy after thresholding, employing a dictionary of functions with linear and cubic polynomials, and cosine, as well as $\lambda = 0.25$.

| Dictionary functions | Coefficients | | |
|-------------------------|--------------|----------|----------|
| | $x_1(t)$ | $x_2(t)$ | $x_3(t)$ |
| 1 | 0 | 0 | 0.9947 |
| x_1 | 0 | 0.9841 | 0 |
| x_2 | 0.9910 | -0.2687 | 0 |
| x_3 | 0 | 0 | 0 |
| x_1^3 | 0 | -0.9912 | 0 |
| x_2^3 | 0 | 0 | 0 |
| x_3^3 | 0 | 0 | 0 |
| $\cos x_1$ | 0 | 0 | 0 |
| $\cos x_2$ | 0 | 0 | 0 |
| $\cos x_3$ | 0 | 0.6229 | 0 |

The identified coefficients before and after applying a threshold $\lambda = 0.25$ are presented in Tables 2 and 3. The coefficients obtained before thresholding indicate the presence of several non-zero terms, reflecting the complexity of the system's dynamics. However, after applying a threshold, many coefficients are set to zero, retaining only the most significant terms that contribute to the system's behavior. This sparsity is crucial for interpretability and reducing overfitting, demonstrating SINDy can be used combined with the **CEopt** solver to handle identification problems for complex nonlinear systems.

The results presented in Figure 19 show the comparison between the training data, ground truth, and the predictions made by the thresholded SINDy model for the state variables x_1 and x_2 . The SINDy model accurately captures the dynamics of the system, even with the presence of noise in the training data.

Although the CE method is not the standard solver for SINDy, this example highlights its robustness and capability to achieve good results in a challenging optimization problem. We do not recommend using CE in place of STLS or SR3 for SINDy. However, this example shows that CE can handle the problem when good bounds for the parameters and a good guess for the mean are prescribed.

6. Final remarks

This paper introduces **CEopt**, a MATLAB-based implementation of the Cross-Entropy (CE) method for solving non-convex optimization problems. Through detailed theoretical foundations, historical development, and practical examples, we have demonstrated the method's robustness and versatility in addressing complex optimization challenges.

The CE method stands out for its ability to transform deterministic optimization problems into stochastic ones, leveraging adaptive importance sampling to iteratively hone in on the global optimum. This probabilistic framework simplifies navigating complex, multidimensional landscapes and ensures a focused and efficient search process. The method's simplicity, minimal set of control parameters, and robust convergence properties make it accessible and effective across various applications.

Our implementation, **CEopt**, extends the CE method's capabilities by incorporating advanced features such as robust handling of both equality and inequality constraints through an augmented Lagrangian approach. This enhancement broadens the method's applicability, particularly in engineering and control systems where such constraints are critical. Additionally, **CEopt** offers user-friendly interfaces and detailed diagnostic outputs, facilitating ease of use and comprehensive analysis of the optimization process.

The numerical experiments conducted illustrate **CEopt**'s effectiveness across various domains, including mechanical equilibrium, fractional-order control, and machine learning. These case studies underscore the method's capacity to provide high-quality solutions in scenarios characterized by non-convex, multidimensional, and noisy data landscapes.

While **CEopt** provides a robust and user-friendly implementation of the CE method, it is crucial for users to carefully specify the bounds for the design variables using the **lb** (lower bound) and **ub** (upper bound) parameters. Without well-defined bounds, the solver may struggle to find high-quality solutions, especially in high-dimensional spaces where the search space can become overwhelmingly large and complex. For more challenging problems, fine-tuning the tolerances and smoothing parameters within the **CEstr** structure is often necessary.

This fine-tuning process can be iterative and requires a degree of trial and error to balance the convergence speed and solution accuracy. Users must be prepared to adjust these settings based on the specific characteristics of their optimization problems to fully leverage the potential of **CEopt**.

In conclusion, **CEopt** represents a significant addition to the optimization toolkit, bridging the gap between theory and practical application. Its development enhances researchers' and practitioners' ability to engage with and solve complex non-convex optimization problems, providing a reliable, transparent, and user-friendly tool. Future work will focus on further enhancing the algorithm's efficiency, expanding its applicability, and integrating it with other optimization frameworks to address the evolving needs of the optimization community.

We invite the numerical optimization community to explore **CEopt** and contribute to its ongoing development, ensuring it remains a cutting-edge tool for addressing the increasingly complex optimization challenges of our time. For more information, updates, and educational content, please visit <https://ceopt.org>. This website and the solver will be continuously updated to include improvements, bug corrections, and new educational content as the use of the solver expands. We encourage the community to explore, use, and contribute to **CEopt**, enhancing its capability to tackle optimization problems across various domains.

Acknowledgments

The first author acknowledge the support given by by the Brazilian agencies Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) under Finance Code 001, Conselho Nacional de Desenvolvimento Científico e Tecnológico under the grant 305476 /2022-0, and the Carlos Chagas Filho Research Foundation of Rio de Janeiro State (FAPERJ) under grants 210.167/2019, 211.037/2019, and 201.294/2021. The authors thank Mr. Diego Matos (UERJ) for the elaboration of Figure 18 and Prof. Welington Oliveira (MINES Paris-Tech) for an insightful discussion about augmented Lagrangian methods.

References

- Asmussen S, Kroese DP, Rubinstein RY (2005). “Heavy tails, importance sampling and cross–entropy.” *Stochastic Models*, **21**(1), 57–76. [doi:10.1081/STM-200046472](https://doi.org/10.1081/STM-200046472).
- Balay S, et al. (2023). “PETSc/TAO Users Manual.” *Technical Report ANL-21/39 - Revision 3.20*, Argonne National Laboratory. [doi:10.2172/1968587](https://doi.org/10.2172/1968587).
- Basílio JCC, Oliveira TR, Telles Ribeiro JG, Cunha Jr A (2022). “Evaluation of Fractional-Order Sliding Mode Control Applied to an Energy Harvesting System.” In *16th International Workshop on Variable Structure Systems (VSS 2022)*. Rio de Janeiro, Brazil. [doi:10.1109/VSS57184.2022.9902105](https://doi.org/10.1109/VSS57184.2022.9902105).
- Basílio JCC, Telles Ribeiro JG, Cunha Jr A, Oliveira TR (2019). “On the classical and fractional control of a nonlinear inverted cart-pendulum system: a comparative analysis.” In *15th International Conference on Vibration Engineering and Technology of Machinery (VETOMAC 2019)*. Curitiba, Brazil. [doi:10.1007/978-3-030-60694-7_26](https://doi.org/10.1007/978-3-030-60694-7_26).

- Basílio JCC, Telles Ribeiro JG, Cunha Jr A, Oliveira TR (2021). “An optimal fractional LQR-based control approach applied to a cart-pendulum system.” In *Second International Nonlinear Dynamics Conference (NODYCON 2021)*. Virtual Mode, Italy. doi:10.1007/978-3-030-81166-2_17.
- Bazaraa MS, Sherali HD, Shetty CM (2006). *Nonlinear Programming: Theory and Algorithms*. Wiley-Interscience.
- Bekker J, Aldrich C (2011). “The cross-entropy method in multi-objective optimisation: An assessment.” *European Journal of Operational Research*, **211**(1), 112–121. doi:10.1016/j.ejor.2010.10.028.
- Bendtsen C (2022). *pso: Particle Swarm Optimization*. R package version 1.0.4.
- Benham T, Duan Q, Kroese DP, Liquet B (2017). “CEoptim: Cross-entropy R package for optimization.” *Journal of Statistical Software*, **76**(8), 1–29. doi:10.18637/jss.v076.i08.
- Bertsekas D (2016). *Nonlinear Programming*. 3rd edition. Athena Scientific.
- Bertsekas DP (2009). *Convex Optimization Theory*. Athena Scientific.
- Bertsekas DP (2015). *Convex Optimization Algorithms*. Athena Scientific.
- Bonnans J, Gilbert JC, Lemarechal C, Sagastizábal CA (2009). *Numerical Optimization: Theoretical and Practical Aspects*. 2nd edition. Springer.
- Botev ZI, Kroese DP, Rubinstein RY, L’Ecuyer P (2013). “Chapter 3 - The cross-entropy method for optimization.” In C Rao, V Govindaraju (eds.), *Handbook of Statistics*, volume 31 of *Handbook of Statistics*, pp. 35–59. Elsevier. doi:10.1016/B978-0-444-53859-8.00003-5.
- Boyd S, Vandenberghe L (2004). *Convex Optimization*. Cambridge University Press.
- Boyd S, Vandenberghe L (2018). *Introduction to Applied Linear Algebra – Vectors, Matrices, and Least Squares*. Cambridge University Press.
- Brunton S, Kutz JN (2022). *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. 2nd edition. Cambridge University Press.
- Brunton SL, Proctor JL, Kutz JN (2016). “Discovering governing equations from data by sparse identification of nonlinear dynamical systems.” *Proceedings of the National Academy of Sciences*, **113**, 3932–3937. doi:10.1073/pnas.1517384113.
- Chan J, Papaioannou I, Straub D (2023). “Bayesian improved cross entropy method for network reliability assessment.” *Structural Safety*, **103**, 102344. doi:10.1016/j.strusafe.2023.102344.
- Chepuri K, Homem-de Mello T (2005). “Solving the vehicle routing problem with stochastic demands using the cross-entropy method.” *Annals of Operations Research*, **134**(1), 153–181. doi:10.1007/s10479-005-5729-7.

- Conn AR, Gould NIM, Toint PL (1991). “A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds.” *SIAM Journal on Numerical Analysis*, **28**(2), 545–572.
- Conn AR, Gould NIM, Toint PL (1997). “A globally convergent augmented Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds.” *Mathematics of Computation*, **66**(217), 261–288.
- Cunha Jr A (2021). “Enhancing the performance of a bistable energy harvesting device via the cross-entropy method.” *Nonlinear Dynamics*, **103**, 137–155. [doi:10.1007/s11071-020-06109-0](https://doi.org/10.1007/s11071-020-06109-0).
- Cunha Jr A, Barton DAW, Ritto TG (2023). “Uncertainty quantification in mechanistic epidemic models via cross-entropy approximate Bayesian computation.” *Nonlinear Dynamics*, **111**, 9649–9679. [doi:10.1007/s11071-023-08327-8](https://doi.org/10.1007/s11071-023-08327-8).
- Cunha Jr A, Soize C, Sampaio R (2015). “Computational modeling of the nonlinear stochastic dynamics of horizontal drillstrings.” *Computational Mechanics*, **56**, 849–878. [doi:10.1007/s00466-015-1206-6](https://doi.org/10.1007/s00466-015-1206-6).
- Dantas E, Cunha Jr A, Silva TAN (2019a). “A numerical procedure based on cross-entropy method for stiffness identification.” In *5th International Conference on Structural Engineering Dynamics (ICEDyn 2019)*. Viana do Castelo, Portugal.
- Dantas E, Cunha Jr A, Soeiro FJCP, Cayres BC, Weber HI (2019b). “An inverse problem via cross-entropy method for calibration of a drill string torsional dynamic model.” In *25th ABCM International Congress of Mechanical Engineering (COBEM 2019)*. Uberlândia, Brazil.
- De Boer P, Kroese DP, Mannor S, Rubinstein RY (2005). “A tutorial on the cross-entropy method.” *Annals of Operations Research*, **134**, 19–67. [doi:https://doi.org/10.1007/s10479-005-5724-z](https://doi.org/10.1007/s10479-005-5724-z).
- de Mello TH (2007). “A study on the cross-entropy method for rare-event probability estimation.” *INFORMS Journal on Computing*, **19**(3), 381–394. [doi:10.1287/ijoc.1060.0176](https://doi.org/10.1287/ijoc.1060.0176).
- D’Elia M, Gulian M, Olson H, Karniadakis GE (2021). “Towards a Unified theory of Fractional and Nonlocal Vector Calculus.” *Fractional Calculus and Applied Analysis*, **24**(5), 1301–1355. [doi:10.1515/fca-2021-0057](https://doi.org/10.1515/fca-2021-0057).
- Durán Fernández O, Figueroa Mora K, Maldonado P (2023). “Optimizing an investment portfolio using cross-entropy algorithms.” In *2023 XLIX Latin American Computer Conference (CLEI)*, pp. 1–5.
- Farahmand-Tabar S, Ashtari P, Babaei M (2023). *Dynamic intelligence of self-organized map in the frequency-based optimum design of structures*, pp. 1–37. Springer Nature Singapore, Singapore. [doi:10.1007/978-981-19-8851-6_45-1](https://doi.org/10.1007/978-981-19-8851-6_45-1).
- Fortin FA, De Rainville FM, Gardner MA, Parizeau M, Gagné C (2012). “DEAP: Evolutionary algorithms made easy.” *Journal of Machine Learning Research*, **13**, 2171–2175.

- Fourer R (1996). *AMPL : A modeling language for mathematical programming*. 2. ed. edition. San Francisco, Calif. : Scientific Pr.
- Galassi M, et al. (2018). *GNU Scientific Library Reference Manual*. 3rd edition. Free Software Foundation.
- Gonçalves MS, Lopez RH, Miguel LFF (2015). “Search group algorithm: A new metaheuristic method for the optimization of truss structures.” *Computers & Structures*, **153**, 165–184. doi:10.1016/j.compstruc.2015.03.003.
- Grant M, Boyd S (2008). “Graph implementations for nonsmooth convex programs.” In V Blondel, S Boyd, H Kimura (eds.), *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pp. 95–110. Springer-Verlag Limited. http://stanford.edu/~boyd/graph_dcp.html.
- Grant M, Boyd S (2014). “CVX: Matlab Software for Disciplined Convex Programming, version 2.1.” <http://cvxr.com/cvx>.
- Haber RE, Beruvides G, Quiza R, Hernandez A (2017). “A simple multi-objective optimization based on the cross-entropy method.” *IEEE Access*, **5**, 22272–22281. doi:10.1109/ACCESS.2017.2764047.
- Hastie T, Tibshirani R, Friedman J (2016). *The Elements of Statistical Learning*. Springer.
- Hindmarsh AC, Brown P, Grant K, Lee S, Serban R, Shumaker DE, Woodward CS (2005). “SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers.” *ACM Transactions on Mathematical Software*, **31**(3), 363–396. doi:10.1145/1089014.1089020.
- Homem-de Mello T, Rubinstein R (2002). “Estimation of rare event probabilities using cross-entropy.” In *Proceedings of the Winter Simulation Conference*, volume 1, pp. 310–319 vol.1.
- Hornik K, Meyer D, Schwendinger F, Theussl S (2023). *ROI: R Optimization Infrastructure*. R package version 1.0-1, URL <https://CRAN.R-project.org/package=ROI>.
- Issa MVS, Cunha Jr A, Soeiro FJCP, Pereira A (2018). “Structural optimization using the cross-entropy method.” In *XXXVIII Congresso Nacional de Matemática Aplicada e Computacional (CNMAC 2018)*. Campinas, Brazil.
- Issa MVS, Cunha Jr A, Soeiro FJCP, Pereira A (2023). “Optimizing truss structures with natural frequency constraints using the cross-entropy method.” In *27th International Congress of Mechanical Engineering (COBEM 2023)*. Florianópolis, Brazil.
- Johnson SG (2007). “The NLOpt nonlinear-optimization package.” <https://github.com/stevengj/nlopt>.
- Kaipio J, Somersalo E (2004). *Statistical and Computational Inverse Problems*. Springer.
- Kroese DP, Botev ZI, Taimre T, Vaisman R (2019). *Data Science and Machine Learning: Mathematical and Statistical Methods*. Chapman and Hall/CRC.
- Kroese DP, Hui KP (2007). *Applications of the Cross-Entropy Method in Reliability*, pp. 37–82. Springer Berlin Heidelberg. doi:10.1007/978-3-540-37372-8_3.

- Kroese DP, Porotsky S, Rubinstein RY (2006). “The cross-entropy method for continuous multi-extremal optimization.” *Methodology and Computing in Applied Probability*, **8**(3), 383–407. doi:[10.1007/s11009-006-9753-0](https://doi.org/10.1007/s11009-006-9753-0).
- Kroese DP, Rubinstein RY, Cohen I, Porotsky S, Taimre T (2013). *Cross-Entropy Method*, pp. 326–333. Springer. doi:[10.1007/978-1-4419-1153-7_131](https://doi.org/10.1007/978-1-4419-1153-7_131).
- Kroese DP, Rubinstein RY, Taimre T (2007). “Application of the cross-entropy method to clustering and vector quantization.” *Journal of Global Optimization*, **37**(1), 137–157. doi:[10.1007/s10898-006-9041-0](https://doi.org/10.1007/s10898-006-9041-0).
- Kroese DP, Taimre T, Botev ZI (2011). *Handbook of Monte Carlo Methods*. Wiley.
- Lubin M, Dowson O, Dias Garcia J, Huchette J, Legat B, Vielma JP (2023). “JuMP 1.0: Recent improvements to a modeling language for mathematical optimization.” *Mathematical Programming Computation*. doi:[10.1007/s12532-023-00239-3](https://doi.org/10.1007/s12532-023-00239-3).
- Maher M, Liu R, Ngoduy D (2013). “Signal optimisation using the cross entropy method.” *Transportation Research Part C: Emerging Technologies*, **27**, 76–88. doi:[10.1016/j.trc.2011.05.018](https://doi.org/10.1016/j.trc.2011.05.018).
- Mannor S, Peleg D, Rubinstein R (2005). “The cross entropy method for classification.” In *Proceedings of the 22nd International Conference on Machine Learning*, ICML ’05, p. 561–568. Association for Computing Machinery.
- Marelli S, Sudret B (2014). “UQLab: A framework for uncertainty quantification in MATLAB.” In *The 2nd International Conference on Vulnerability and Risk Analysis and Management (ICVRAM 2014)*. University of Liverpool, United Kingdom.
- Margolin L (2005). “On the convergence of the cross-entropy method.” *Annals of Operations Research*, **134**(1), 201–214. doi:[10.1007/s10479-005-5731-0](https://doi.org/10.1007/s10479-005-5731-0).
- MathWorks Inc (2024a). *MATLAB Global Optimization Toolbox version 24.1 (R2024a)*. URL <https://www.mathworks.com>.
- MathWorks Inc (2024b). “MATLAB Optimization Toolbox version: 24.1 (R2024a).” URL <https://www.mathworks.com>.
- Miranda LJV (2018). “PySwarms, a research-toolkit for particle swarm optimization in Python.” *Journal of Open Source Software*, **3**. doi:[10.21105/joss.00433](https://doi.org/10.21105/joss.00433).
- Moustapha M, Lataniotis C, Wiederkehr P, Wagner PR, Wicaksono D, Marelli S, Sudret B (2022). “UQLib user manual.” *Technical report*, Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland. Report UQLab-V2.0-201.
- Nocedal J, Wright S (2006). *Numerical Optimization*. 2nd edition. Springer.
- Oliveira TR, Krstic M (2022). *Extremum Seeking Through Delays and PDEs*. SIAM. doi:[10.1137/1.9781611977356](https://doi.org/10.1137/1.9781611977356).
- Ortigueira MD (2011). *Fractional Calculus for Scientists and Engineers*. Springer. doi:[10.1007/978-94-007-0747-4](https://doi.org/10.1007/978-94-007-0747-4).

- Ortigueira MD, Machado JAT (2015). “What is a fractional derivative?” *Journal of Computational Physics*, **293**, 4–13. doi:[10.1016/j.jcp.2014.07.019](https://doi.org/10.1016/j.jcp.2014.07.019).
- Quaranta G, Lacarbonara W, Masri SF (2020). “A review on computational intelligence for identification of nonlinear dynamical systems.” *Nonlinear Dynamics*, **99**, 1709–1761. doi:[10.1007/s11071-019-05430-7](https://doi.org/10.1007/s11071-019-05430-7).
- Raqueti RS, Teloli RO, da Silva S, Busseta P, Cunha Jr A (2023). “On the use of stochastic Bouc-Wen model for simulating viscoelastic internal variables from a finite element approximation of steady-rolling tire.” *Journal of Vibration and Control*, **29**, 4799–4813. doi:[10.1177/107754632211250](https://doi.org/10.1177/107754632211250).
- Rubinstein RY (1997). “Optimization of computer simulation models with rare events.” *European Journal of Operations Research*, **99**, 89–112. doi:[10.1016/S0377-2217\(96\)00385-2](https://doi.org/10.1016/S0377-2217(96)00385-2).
- Rubinstein RY (1999). “The cross-entropy method for combinatorial and continuous optimization.” *Methodology and Computing in Applied Probability*, **2**, 127–190. doi:[10.1023/A:1010091220143](https://doi.org/10.1023/A:1010091220143).
- Rubinstein RY (2005). “A stochastic minimum cross-entropy method for combinatorial optimization and rare-event estimation.” *Methodology and Computing in Applied Probability*, **7**(1), 5–50. doi:[10.1007/s11009-005-6653-7](https://doi.org/10.1007/s11009-005-6653-7).
- Rubinstein RY, Kroese DP (2004). *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Information Science and Statistics. Springer-Verlag.
- Rubinstein RY, Kroese DP (2007). *Simulation and the Monte Carlo Method*. 2nd edition. Wiley.
- Rubinstein RY, Kroese DP (2016). *Simulation and the Monte Carlo Method*. 3rd edition. Wiley.
- Scrucca L (2013). “GA: A package for genetic algorithms in R.” *Journal of Statistical Software*, **53**(4), 1–37. doi:[10.18637/jss.v053.i04](https://doi.org/10.18637/jss.v053.i04).
- Shampine LF, Reichelt MW (1997). “The MATLAB ODE suite.” *SIAM Journal on Scientific Computing*, **18**(1), 1–22. doi:[10.1137/S1064827594276424](https://doi.org/10.1137/S1064827594276424).
- Soize C (2017). *Uncertainty Quantification: An Accelerated Course with Advanced Applications in Computational Engineering*. Springer.
- Tarantola A (2005). *Inverse Problem Theory and Methods for Model Parameter Estimation*. SIAM.
- Tenorio L (2017). *An Introduction to Data Analysis and Uncertainty Quantification for Inverse Problems*. SIAM.
- Tepljakov A, Petlenkov E, Belikov J (2011a). “FOMCON: a MATLAB toolbox for fractional-order system identification and control.” *International Journal of Microelectronics and Computer Science*, **2**(2), 51–62.

- Tepljakov A, Petlenkov E, Belikov J (2011b). “FOMCON: Fractional-order modeling and control toolbox for MATLAB.” In *Proceedings of the 18th International Conference Mixed Design of Integrated Circuits and Systems - MIXDES 2011*, volume , pp. 684–689.
- Theußl S, Schwendinger F, Hornik K (2020). “ROI: An extensible R optimization infrastructure.” *Journal of Statistical Software*, **94**(15), 1–64. doi:[10.18637/jss.v094.i15](https://doi.org/10.18637/jss.v094.i15).
- Vijina P, Jayasree M (2020). “A survey on recent approaches in image reconstruction.” In *2020 International Conference on Power, Instrumentation, Control and Computing (PICC)*, pp. 1–5.
- Virtanen P, et al. (2020). “SciPy 1.0: Fundamental algorithms for scientific computing in Python.” *Nature Methods*, **17**, 261–272. doi:[10.1038/s41592-019-0686-2](https://doi.org/10.1038/s41592-019-0686-2).
- Wolszczak P, Lonkwic P, Cunha Jr A, Litak G, Molski S (2019). “Robust optimization and uncertainty quantification in the nonlinear mechanics of an elevator brake system.” *Meccanica*, **54**, 1057–1069. doi:[10.1007/s11012-019-00992-7](https://doi.org/10.1007/s11012-019-00992-7).
- Xu H, Fan J, Ma H, Wang Q (2024). “Semiparametric musculoskeletal model for reinforcement learning-based trajectory tracking.” *IEEE Transactions on Instrumentation and Measurement*, **73**, 1–16. doi:[10.1109/TIM.2024.3370760](https://doi.org/10.1109/TIM.2024.3370760).
- Zhao Y, Chen J, Liu L, Cheng X, Xie K, Hu J, Wang Q (2024). “An efficient cross-entropy method addressing high-dimensional dependencies for composite systems reliability evaluation.” *International Journal of Electrical Power & Energy Systems*, **157**, 109857. doi: <https://doi.org/10.1016/j.ijepes.2024.109857>.
- Zheng P, Askham T, Brunton SL, Kutz JN, Aravkin AY (2019). “A Unified Framework for Sparse Relaxed Regularized Regression: SR3.” *IEEE Access*, **7**, 1404–1423. doi:[10.1109/ACCESS.2018.2886528](https://doi.org/10.1109/ACCESS.2018.2886528).

Affiliation:

Americo Cunha Jr
 Department of Applied Mathematics
 Rio de Janeiro State University – UERJ
 Rua São Francisco Xavier, 524, Rio de Janeiro, RJ - Brasil - 20550-900
 E-mail: americoco.cunha@uerj.br