A Framework for Finding Local Saddle Points in Two-Player Zero-Sum Black-Box Games

Anonymous authors
Paper under double-blind review

Abstract

Saddle point optimization is a critical problem employed in numerous real-world applications, including portfolio optimization, generative adversarial networks, and robotics. It has been extensively studied in cases where the objective function is known and differentiable. Existing work in *black-box* settings with unknown objectives that can only be sampled either assumes convexity-concavity in the objective to simplify the problem or operates with noisy gradient estimators. In contrast, we introduce a framework inspired by Bayesian optimization which utilizes Gaussian processes to model the unknown (potentially nonconvex-nonconcave) objective and requires only zeroth-order samples. Our approach frames the saddle point optimization problem as a two-level process which can flexibly leverage existing general-sum Nash game solvers to solve for saddle points of zero-sum games. The upper level of our framework produces a model of the objective function by sampling in promising locations, and the lower level of our framework uses the existing model to frame and solve a general-sum game to identify locations to sample. This lower level procedure can be designed in complementary ways, and we demonstrate the flexibility of our approach by introducing variants which appropriately trade off between factors like runtime, the cost of function evaluations, and the number of available initial samples. We experimentally demonstrate these algorithms on synthetic and realistic datasets in black-box nonconvex-nonconcave settings, showcasing their ability to efficiently locate local saddle points in these contexts.

1 Introduction

We consider the problem of finding saddle points for smooth two-player zero-sum games of the form

$$\textsc{Player 1:}\ \min_x f(x,y) \qquad \textsc{Player 2:}\ \min_y -f(x,y) \qquad x \in \mathbb{R}^{n_x}, y \in \mathbb{R}^{n_y} \tag{1}$$

with an unknown, nonconvex-nonconcave objective f. We assume that we can draw noisy zeroth-order samples of f (i.e., sampling does not provide any derivative information) via a possibly expensive process given coordinates (x, y), where $x \in \mathbb{R}^{n_x}, y \in \mathbb{R}^{n_y}$.

Saddle points are points at which the function f is *simultaneously* a minimum along the x-coordinate and a maximum along the y-coordinate. Such points specialize the well-known Nash equilibrium concept to the setting of two-player, zero-sum games. Saddle point optimization (Tind, 2009) is widely used in real-world applications like economics (Luxenberg et al., 2022), machine learning (Goodfellow et al., 2020), robotics (Agarwal et al., 2023), communications (Moura & Hutchison, 2019), chemistry (Henkelman et al., 2000), and more.

Zero-sum games have been widely studied for known and differentiable objective functions. However, this assumption does not encompass numerous real-world situations with nonconvex-nonconcave objectives which may be unknown and can only be sampled. Such objectives are often referred to as "black-box." For example, in robust portfolio optimization, the goal is to create portfolios resistant to stock market fluctuations (Nyikosa, 2018), which are inherently random and difficult to model but can be sampled in a black-box fashion through trial and error. Similar problems arise in various physical settings, such as robotics (Lizotte et al., 2007) and communication networks (Qureshi & Khan, 2023). Motivated by these real-world examples in nonconvex-nonconcave black-box settings, we present a flexible and extensible framework that seeks to identify a saddle point, (x^*, y^*), such that $f(x^*, y) \leq f(x^*, y^*) \leq f(x, y^*)$, for all x, y in its neighborhood.

Most previous research in this area has focused on solving minimax problems (Bogunovic et al., 2018; Fröhlich et al., 2020; Wang et al., 2022), which take the form $\min_x \max_y f(x, y)$. The difference between minimax and saddle points is subtle: a minimax point achieves the best worst-case outcome for the minimizer (i.e., a Stackelberg equilibrium). In contrast, at a saddle point, the best worst-case and worst best-case outcomes coincide (i.e., a Nash equilibrium). Solutions to minimax problems in general nonconvex-nonconcave settings are not necessarily Nash, and encode an leader-follower hierarchy which is not present for the saddle point concept. In settings like racing, chess, and resource allocation, where rational, adversarial actors make decisions simultaneously, equilibria are best described as saddle points.

Many previous works in saddle point optimization assume convex-concave objectives (v. Neumann, 1928; Korpelevich, 1976; Tseng, 1995; Nemirovski, 2004), for which every minimax point is a saddle and vice versa because the best worst-case and worst best-case always coincide. However, this equivalence does not hold in general nonconvex-nonconcave settings. Notably, some prior works addressing black-box convex-concave settings use zeroth-order samples (Maheshwari et al., 2022). Lastly, finding global saddle points remains an open problem in general settings, so our work specifically focuses on discovering local saddle points, as detailed in Remark 3.5.

In contrast to previous works, we approach this problem in the spirit of bilevel Bayesian optimization: at a high-level, we use Gaussian processes to build a surrogate model for the black-box function $f(x, y)$ by sampling points (x, y) at promising locations, and at a low-level, we identify these sample points by solving *general-sum* games defined on the surrogate model. Specifically, the low-level game selects these samples by seeking local Nash points (Defn. 3.2) of these two-player general-sum games. The high-level optimizer then aims to ensure that in the limit, these samples converge to local saddle points of the black-box problem. We present our **contributions** as follows.

1. To the best of our knowledge, our method for saddle point optimization is the first to experimentally demonstrate an approach that (1) finds saddle points in black-box settings (2) on nonconvex-nonconcave objectives (3) with zeroth-order samples. Prior work either achieves only one or two of these simultaneously or fails to theoretically and experimentally validate the approach in nonconvex-nonconcave settings.
2. We use our framework to propose a set of algorithms for the lower-level game, with each variant catering to a different real-world case based on uncertainty and sampling cost. Thus, our approach allows for versatility in trading off between factors like ease of sampling, exploration and exploitation, and provides a template for future work in this area.
3. We experimentally demonstrate our algorithms' effectiveness on a variety of challenging synthetic and realistic datasets. Due to the limitations of extending Gaussian Processes to higher dimensions, we focus our experiments on low-dimensional settings which are amenable to GPs. We discuss this choice further in our experiments.

2 Related Work and Preliminaries

Saddle Point Optimization: Saddle point problems are widely studied in the game theory (Başar & Olsder, 1998; Cherukuri et al., 2017), optimization (Dauphin et al., 2014; Pascanu et al., 2014), and machine learning communities (Benzi et al., 2005; Jin et al., 2021). We note three previous varieties of algorithms in the area of nonconvex-nonconcave saddle point optimization which guarantee convergence to local saddle points rather than stationary points. Of these, Adolphs et al. (2019) and Gupta et al. (2024) introduce algorithms which solve for saddle points in deterministic settings and neither approach handles unknown objectives. Mazumdar et al. (2019) introduces local symplectic surgery (LSS), a method that, when it converges, provably does so to a local saddle point given access to first-order and second-order derivative measurements by the sampler. However, derivative information is not always available in systems of interest. Moreover, while zeroth-order samplers can estimate noisy first-order (and second-order) derivatives via finite differencing, the extensive sampling requirements for such an approach is prohibitive when sampling is expensive. By contrast, our work proposes an extensible zeroth-order framework for black-box saddle point optimization, and we provide extensive experimental evaluation of several variants tailored to distinct settings.

Gaussian Process (GP): A GP (Rasmussen, 2003), denoted by $\mathrm{GP}(\mu(\cdot), \Sigma(\cdot, \cdot))$, is a set of random variables, such that any finite sub-collection $\{f(x_i)\}_{i=1}^n$ is jointly Gaussian with mean $\mu(x_i) = \mathbb{E}[f(x_i)]$, and covariance $\Sigma = \mathbb{E}[(f(x_i) - \mu(x_i))(f(x_j) - \mu(x_j))], \forall i, j \in \{1, \dots, n\}$.

GPs are mainly used for regression tasks, where they predict an underlying function, $f : \mathbb{R}^n \to \mathbb{R}$, given some previously observed noisy measurements. That is, for any inputs $x_1, \dots, x_n \in \mathcal{X} \subseteq \mathbb{R}^k$, and the corresponding noisy measurements, $r_1, \dots, r_n \in \mathbb{R}$, the vector $r = [r_1, r_2, \dots, r_n]^\top$ is modeled as multivariate Gaussian distribution with mean vector μ (typically assumed to be zero), and covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$. The covariance matrix, Σ, is calculated as follows: $\Sigma_{i,j} = K(x_i, x_j), \forall i, j \in \{1, \dots, n\}$, where $K(\cdot, \cdot)$ is the kernel function. Typically, it is assumed that errors $z_i = r_i - f(x_i)$ are normally, independently, and identically distributed, i.e. $z_i \in \mathcal{N}(0, \sigma_z^2)$. At a test point x_*, we compute the marginal distribution of $f(x_*)$ given r via

$$f\left(x_*\right) \mid r \;\sim\; \mathcal{N}(\underbrace{k_*^\top (\Sigma + \sigma_z^2 I)^{-1} r}_{\mu_t(x_*)}, \underbrace{K\left(x_*, x_*\right) - k_*^\top (\Sigma + \sigma_z^2 I)^{-1} k_*}_{\sigma_t(x_*)}), \tag{2}$$

where $k_* = \left[K\left(x_1, x_*\right), \cdots, K\left(x_n, x_*\right)\right]^\top$. A more detailed description of (2) can be found in Rasmussen (2003). For typical (e.g., squared exponential) kernels, the number of samples required for GP regression increases exponentially in the number of dimensions; thus, GPs are appropriate for only relatively low dimensional spaces. Nevertheless, GPs form a useful tool in addressing many relevant real-world problems. They are well-suited for modeling uncertainty in black-box optimization problems because they predict both the function value and the uncertainty of that prediction, which is crucial for balancing exploration and exploitation in optimization frameworks. Moreover, the smoothness of GP estimates allow for the deployment of standard gradient-based algorithms to estimate solutions to optimization problems.

Bayesian Optimization (BO) with Gaussian Processes: Močkus (1975); Brochu et al. (2010b); Shahriari et al. (2016), is a sequential search method for maximizing an unknown objective function $f : \mathbb{R}^k \to \mathbb{R}$ with as few evaluations as possible. It starts with initializing a prior over f and uses an acquisition function to select the next point x_t given the history of observations, $f(x_1), \dots, f(x_{t-1})$. The unknown objective, f, is sampled at x_t, and its observed value $f(x_t)$ is used to update the current estimate of f. Typically, f is modeled as a GP, and the GP prior is updated with new samples. One common acquisition function, used in the GP-UCB algorithm, is the Upper Confidence Bound $\mathrm{UCB}_t(x, y) = \mu_t\left(x, y\right) + \beta_t \sigma_t\left(x, y\right)$ (defined here to estimate the two-argument zero-sum objectives from (1)). GP-UCB (Srinivas et al., 2010) has been used in a variety of settings including robotics (Deisenroth et al., 2013), chemistry (Westermayr & Marquetand, 2021), user modeling (Brochu et al., 2010a), and reinforcement learning (Cheung et al., 2020). A high β_t parameter in UCB_t implies a more optimistic maximizer (i.e. favoring exploration) in the presence of uncertainty. The UCB acquisition function combines the estimated mean, $\mu_t(x, y)$, and the estimated standard deviation, $\sigma_t(x, y)$, of the unknown objective function f at point x at iteration t. Analogously, we can also define the Lower Confidence Bound ($\mathrm{LCB}_t(x, y) = \mu_t\left(x, y\right) - \beta_t \sigma_t\left(x, y\right)$).

3 Problem Formulation

Problem Setup: We consider the two-player, zero-sum game in (1), and focus on the case in which the objective f is an unknown function defined on the domain $\mathbb{R}^{n_x} \times \mathbb{R}^{n_y}$, and can only be realized through (possibly expensive, noisy) evaluations. That is, we query the objective at a point $(x, y) \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_y}$ and observe a noisy sample $r = f(x, y) + z$, where $z \sim \mathcal{N}\left(0, \sigma_z^2\right)$. Although f itself is unknown, we will assume that it is smooth and can be differentiated twice. Our goal is to find Local Saddle Points (LSPs) of f.

Our proposed framework will consist of two stages: at the lower level, we will solve a general-sum game defined on a GP surrogate model to identify (local) Nash points, and at the high level we will sample f at those points and refine the GP surrogate model. Next, we formally define Nash points for general-sum games, the conditions they satisfy, and when such a point coincides with a saddle point for a zero-sum game. For a two-player general-sum game, where player 1 is minimizing function f_1, and player 2 is minimizing function f_2, a Nash point, (x^*, y^*), is defined as:

Definition 3.1 (Global Nash Point (GNP) for Two-Player General-Sum Game)**.** (Başar & Olsder, 1998, Defn. 2.1) Point (x^*, y^*) is a global Nash point of objectives f_1, f_2 if, for all $x, y \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_y}$,

$$f_1(x^*, y^*) \leq f_1(x, y^*), f_2(x^*, y^*) \leq f_2(x^*, y). \tag{3}$$

At a GNP, variables x and y cannot change their respective values without achieving a less favorable outcome. In general, there does not always exist a global saddle point, and when it exists, it is not necessarily unique. Finding a GNP is computationally intractable in nonconvex settings (as in global nonconvex optimization), so we seek a Local Nash Point (LNP), where this property need only hold within a small neighborhood. A LNP is characterized by first- and second-order conditions.

Definition 3.2 (LNP for Two-Player General-Sum Game)**.** (Ratliff et al., 2016, Defn. 1) Let $\|\cdot\|$ denote a vector norm. A point, (x^*, y^*), is a local Nash point of cost functions f_1 and f_2 if there exists a $\tau > 0$ such that for any x and y satisfying $\|x - x^*\| \leq \tau$ and $\|y - y^*\| \leq \tau$, we have (3).

Proposition 3.3 (First-order Necessary Condition)**.** *(Ratliff et al., 2016, Prop. 1) For differentiable f_1 and f_2, a local Nash point (x^*, y^*) satisfies $\nabla_x f_1(x^*, y^*) = 0$ and $\nabla_y f_2(x^*, y^*) = 0$.*

Proposition 3.4 (Second-order Sufficient Condition)**.** *(Ratliff et al., 2016, Defn. 3) For twice-differentiable f_1 and f_2, if (x, y) satisfies the conditions in Prop. 3.3, $\nabla^2_{xx} f_1(x, y) \succ 0$, and $\nabla^2_{yy} f_2(x, y) \succ 0$, then it is a strict local Nash point.*

Remark 3.5 (Nash Point is a Saddle Point when $f_1 = -f_2$). If $f = f_1 = -f_2$, then the point (x^*, y^*) is a global saddle point of f when Defn. 3.1 holds and a local saddle point when Defn. 3.2 holds. A local saddle point is characterized by the same first- and second-order conditions defined in Prop. 3.3 and Prop. 3.4, respectively. Henceforth, we use the term saddle point to refer to Nash points in zero-sum games and refer to Defns. 3.1 and 3.2 and Props. 3.3 and 3.4 for Nash and saddle points.

4 Black-box Algorithms for Finding Local Saddle Points

We summarize our Bayesian Optimization (BO)-inspired bilevel framework for identifying local saddle points in the black-box setting. Let μ_t and Σ_t, respectively, define mean and covariance functions that estimate the unknown objective f as a Gaussian process based on a dataset $\mathcal{S}_t = \{(x_i, y_i, r_i)\}_{i=1}^t$ where $r_i \in \mathbb{R}$ is a (potentially noisy) sample of f at point (x_i, y_i). We define a zero-sum game, which we refer to as the *high-level* game,

$$\text{PLAYER 1:} \quad x^* = \arg\min_x \mu_t(x, y) \quad \text{PLAYER 2:} \quad y^* = \arg\min_y -\mu_t(x, y) \tag{4}$$

This game has two purposes: primarily, it seeks to solve for a LSP of the original problem (1). Doing so requires solving the secondary problem of refining the GP estimate by strategically sampling f to form $\mathcal{S}_{t+1} = \mathcal{S}_t \cup \{x_{t+1}, y_{t+1}, r_{t+1}\}$ at iteration $t+1$. To identify promising points, we solve a *low-level* general-sum game

$$\text{PLAYER 1:} \quad \bar{x}^* = \arg\min_x \text{LCB}_t(x, y) \quad \text{PLAYER 2:} \quad \bar{y}^* = \arg\min_y -\text{UCB}_t(x, y) \tag{5}$$

for a local Nash point. As μ_t and Σ_t are smooth functions, we can solve (5) by deploying standard gradient-based algorithms on them to solve the lower-level game for first-order stationary points. In our work, we solve (5) using a technique (Alg. 1) which is based on Newton's method, but the structure of our method allows for any general-sum Nash game solver to be used.

Critically, our method relies on an observation about the relationship between this general-sum LNP and the zero-sum LSP (x^*, y^*) we wish to find. In the limit of infinite samples in the neighborhood of (x^*, y^*), when the GP surrogate converges to f, then the uncertainty σ converges to zero and LCB_t and UCB_t converge to the mean μ_t, which converges to f and leads problems (4) and (5) to coincide. These games optimize μ_t, which is an estimate of f, so note that any solutions will be approximate.

4.1 Defining and Solving The Low-Level Game For Local Nash Points

Extending the familiar "optimization in the face of uncertainty" principle from BO (Snoek et al., 2012) and active learning (Yang et al., 2015), we construct the low-level game in (5) so that each player minimizes a

```
Function LLGAME((x, y, LCB, UCB)):
1:  $\bar{x}_0, \bar{y}_0 = x, y$.
2:  while $M_t^{\text{CB}}(\bar{x}_t, \bar{y}_t) \geq \epsilon$ do
3:      Get next iterate $(\bar{x}_{t+1}, \bar{y}_{t+1})$ using LCB, UCB as shown in (7).
    end
4:  Return $\bar{x}^*, \bar{y}^*$.
End Function
```

Algorithm 1: LLGame

```
Input: $\mathcal{S}_0$, GP prior $(\mu_0, \sigma_0)$, $\epsilon$.
1: Start with initial point $(x_0, y_0) = \arg\min_{(x,y)\in\mathcal{S}_0} M_t^{\mu}(x, y)$.
2: while $M_t^{\mu}(x_t, y_t) \geq \epsilon$ do
3:     $(x_{t+1}, y_{t+1}) = \text{LLGame}(x_t, y_t, \text{LCB}_t, \text{UCB}_t)$.
4:     Sample $f(x_{t+1}, y_{t+1})$ and add to $\mathcal{S}_{t+1}$.
5:     Update $\mu_{t+1}, \sigma_{t+1}, \text{LCB}_{t+1}, \text{UCB}_{t+1}$.
6:     $(x_t, y_t) = (x_{t+1}, y_{t+1})$.
   end
7: Return Saddle point $x^*, y^*$.
```

Algorithm 2: Bayesian Saddle Point (BSP) Algorithm

lower bound on its nominal performance index. As in active learning, this design is intended to encourage "exploration" of promising regions of the optimization landscape early on, before "exploiting" the estimated GP model. These bounds, LCB_t and UCB_t, are constructed at each iteration t of the high-level game.

Finding LNPs is computationally intractable in general; therefore, in practice we seek only points which satisfy the first-order conditions of Prop. 3.3. To find this solution, we introduce a new function, $G_t^{\text{CB}}(x, y) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}^k$, whose roots coincide with these first-order Nash points. The superscript CB and subscript t signify that we are finding the roots with confidence bounds (CB) at iteration t. Specifically, we seek the roots of the following nonlinear system of (algebraic) equations:

$$G_t^{\text{CB}}(x, y) = \begin{bmatrix} \nabla_x \text{LCB}_t(x, y) \\ -\nabla_y \text{UCB}_t(x, y) \end{bmatrix} = 0. \tag{6}$$

The LLGame Algorithm: In Alg. 1, we present our approach to solving the *low-level* game, which we refer to as LLGame. LLGame utilizes the current confidence bounds, $(\text{LCB}_t, \text{UCB}_t)$, to determine the local Nash points by finding roots of G_t^{CB} using Newton's method. The LLGame function does not make any new queries of f; instead, it uses the most up-to-date confidence bounds to identify the local Nash points. LLGame takes an initial point, (x, y), and current confidence bounds, $(\text{LCB}_t, \text{UCB}_t)$, as inputs. Starting from line 2, the algorithm iteratively updates the point, $(\bar{x}_t, \bar{y}_t)$, using (7)—discussed below—and halts when a merit function, $M_t^{\text{CB}}(\bar{x}_t, \bar{y}_t)$, and therefore the gradients $\nabla_{\bar{x}_t}\text{LCB}_t(\bar{x}_t, \bar{y}_t)$ and $\nabla_{\bar{y}_t}\text{UCB}_t(\bar{x}_t, \bar{y}_t)$, are sufficiently small. Ultimately in line 4, the function returns the final point, $(\bar{x}^*, \bar{y}^*)$, once it discovers a stationary point satisfying first-order Nash conditions as per Prop. 3.3.

Defining Convergence (line 2): To gauge the progress towards a root of $G_t^{\text{CB}}(x, y)$, we employ a *merit function*, a scalar-valued function of (x, y), which equals zero at a root and grows unbounded far away from a root. Specifically, we use the squared ℓ_2 norm as the merit function, i.e., $M_t^{\text{CB}}(x, y) = \frac{1}{2}||G_t^{\text{CB}}(x, y)||_2^2$; each LNP of (5) is a global minimizer of M.

Nonlinear Root-finding with Newton's Method (line 3): To find the roots of $G_t^{\text{CB}}(x, y)$, our work employs Newton's method, which is an iterative method that is widely utilized for solving nonlinear systems. The Newton step, $p_t(x, y)$, is obtained by linearly approximating the function, G_t^{CB}, with its Jacobian matrix $J_t(x, y)$ at the current estimate (x, y) and identifying the root of that approximation. The step $p_t(x, y)$ therefore satisfies:

$$J_t(x, y)p_t(x, y) = -G_t^{\text{CB}}(x, y), \quad \text{with} \quad J_t(x, y) = \begin{bmatrix} \nabla^2_{x,x}\text{LCB}_t & \nabla^2_{x,y}\text{LCB}_t \\ -\nabla^2_{y,x}\text{UCB}_t & -\nabla^2_{y,y}\text{UCB}_t \end{bmatrix}. \tag{7}$$

Consequently, we update the current point, (x, y), by taking the step p_t to reach the next point. In our experiments, we employ Newton's method with a Wolfe linesearch, which is known to converge rapidly when initialized near a root, as shown in (Nocedal & Wright, 2006, Ch. 11). Note that the Jacobian J_t requires minimal effort to compute in the lower-dimensional spaces that are classically amenable to black-box optimization. We provide exact implementation details in Appendix B.

Adapting LLGame: We note that other optimizers can be used to solve for (first-order) LNPs of (5). One prevalent example is the gradient ascent-descent method (discussed by Mescheder et al. (2017) and Balduzzi et al. (2018), among others), which uses gradient steps instead of Newton steps as in our method. Our framework is flexible and LLGame can readily be adapted to use these methods to identify local Nash points.

4.2 Solving the High-Level Game: Finding Local Saddle Points with BSP

In the high-level game, we seek to solve zero-sum game (4) to identify the saddle points of μ_t. Upon extracting a solution to (5) in LLGame, we sample the objective f around the low-level Nash point $(\bar{x}^*, \bar{y}^*)$. The result of this sampling is used to update the mean μ_t and uncertainty estimate σ_t for the GP surrogate of f. Following the design of Sec. 4.1, we seek to identify first-order LSPs of (4) with roots of the function G_t^f and global minima of the corresponding merit function M_t^f:

$$G^f(x, y) = \begin{bmatrix} \nabla_x f(x, y) \\ -\nabla_y f(x, y) \end{bmatrix}, \qquad M^f(x, y) = \frac{1}{2}\|G^f(x, y)\|_2^2. \tag{8}$$

However, as f is unknown, we instead define function G_t^μ and corresponding merit function M_t^μ using the GP surrogate model of f by replacing f in (8) with μ_t. Thus, our method identifies saddle points of f by finding the roots of G_t^μ and global minima of M_t^μ.

The Bayesian Saddle Point Algorithm: We now present our overall algorithm BSP (Alg. 2), which searches for the local saddle points. As stated earlier, this distinction between the two games allows us to confirm if a local Nash point is a local saddle point. In Alg. 2, we start by optimizing the hyperparameters of our GP kernel using the initial dataset, $\mathcal{S}_0$, a standard procedure in BO (Snoek et al., 2012). Upon learning the hyperparameters, an initial GP prior, (μ_0, σ_0), is obtained. Then in line 1, we select a starting point, (x_0, y_0), from the initial dataset, $\mathcal{S}_0$, based on the lowest *merit* value. From this point, an iterative search for the local saddle point is conducted in the outer while loop (lines 2-6). The LLGame function is utilized in line 3 to determine the subsequent point (x_{t+1}, y_{t+1}), which is a local Nash point of the general-sum game (5). The point, (x_{t+1}, y_{t+1}), is only a local Nash point for the given LCB_t and UCB_t; we will not be sure if it is a local saddle point of f until we sample f at that point and calculate M_t^μ to validate the conditions in Prop. 3.3.

Consequently, in line 4, the point returned by LLGame is sampled and added to the current dataset. In line 5, new hyperparameters are learned from dataset $\mathcal{S}_{t+1}$, and $\mu_{t+1}, \sigma_{t+1}, \text{LCB}_{t+1}, \text{UCB}_{t+1}$ are updated accordingly. [1] After sampling at the point (x_{t+1}, y_{t+1}), we have decreased the variance at that point, and thus the mean, $\mu_t(x_{t+1}, y_{t+1})$, and its gradient $\nabla\mu_t$ are better representations of f and its gradient ∇f. After each update to the GP surrogate, we check if the merit function value is sufficiently small, i.e. $M_t^\mu(x_t, y_t) \le \epsilon$, where $\epsilon > 0$ is a user-specified tolerance. Ultimately in line 7, the local first-order saddle point (x^*, y^*) is returned after the completion of the outer loop.

4.3 Convergence

In Lemma 4.1, we demonstrate that Alg. 1 will terminate and converge to a point which satisfies Prop. 3.3 under standard technical assumptions for Newton steps to be descent directions on the merit function. Our experiments in Sec. 5 include cases that both satisfy and violate these assumptions, showcasing the performance of our algorithm under various circumstances. The precise statement of Lemma 4.1 closely follows Theorem 11.6 from Nocedal & Wright (2006), which is a standard result on the convergence of Newton methods for identifying roots of nonlinear equations.

Lemma 4.1 (Convergence to Stationary Point Satisfying Prop. 3.3 in LLGame)**.** *Let $J(x, y)$ be Lipschitz continuous in a neighborhood $\mathcal{D} \subset \mathbb{R}^{n_x \times n_y}$ surrounding the sublevel set $\mathcal{L} = \left\{x, y : M_t^{\text{CB}}(x, y) \le M_t^{\text{CB}}(x_0, y_0)\right\}$. Assume that both $\|J(x, y)\|_2$ and $\|G_t^{\text{CB}}(x, y)\|_2$ have upper bounds in $\mathcal{D}$. Let step lengths α_k satisfy the Wolfe conditions (Nocedal & Wright, 2006, Sec. 3.1). If $\|J(x, y)^{-1}\|_2$ has an upper bound, then Alg. 1 will converge and return a root of $G_t^{\text{CB}}(x, y)$ which satisifes Prop. 3.3.*

Proof. These assumptions match those of Theorem 11.6 of Nocedal & Wright (2006), which can be applied to find that the Zoutendijk condition holds, that is

$$\sum_{k \ge 0} \cos^2(\theta_k) \, \|J^\intercal(x_k, y_k) G_t^{\text{CB}}(x_k, y_k)\|_2^2 < \infty, \tag{9}$$

---

[1] This is a standard procedure in black-box optimization, explained in further detail in Appendix B.8.

where θ_k is the angle between the Newton direction and the gradient of the merit function, ∇M^{CB}, cf. Eq. 11.40 in Nocedal & Wright (2006). We are assured that $\cos\theta_k \geq \delta$ for some $\delta > 0$ if we take Newton steps at every iteration k. Specifically, recall that $\|J(x_k, y_k)\|_2$ and $\|J^{-1}(x_k, y_k)\|_2$ are bounded above by finite and nonzero $B_1, B_2 \in \mathbb{R}_+$, respectively. From these assumptions, we can provide an upper bound on the condition number $\kappa(J) = {}^{\sigma_{\max}(J)}/_{\sigma_{\min}(J)}$, which is the ratio of the largest singular value of J, $\sigma_{\max}(J)$, and smallest singular value of J, $\sigma_{\min}(J)$. Since inverting a matrix inverts its singular values, we know that $\|J\|_2 = \sigma_{\max}(J) \leq B_1$ and that $\|J^{-1}\|_2 = \sigma_{\max}(J^{-1}) = {}^{1}/_{\sigma_{\min}(J)} \leq B_2$, which further implies that $\sigma_{\min}(J) \geq {}^{1}/_{B_2}$ and produces the desired bound on the condition number, $\kappa(J) \geq B_1 B_2$. Using Eq. 11.42 of Nocedal & Wright (2006), we find that $\cos\theta_k \geq {}^{1}/_{\kappa(J)} = \delta$. Lastly, it follows from (9) that $\lim_{k\to\infty} J^{\intercal}(x_k, y_k) G_t^{\mathrm{CB}}(x_k, y_k) = 0$, implying that the iterates approach a stationary point of M^{CB} at which $G_t^{CB} = 0$. Thus, the Newton iterates converge to a root of G_t^{CB} which satisifies the first-order necessary Nash conditions (Prop. 3.3). ☐

In Alg. 2, we sample f at the LNP returned by Alg. 1 to reduce the variance, σ_t, in the confidence bounds. This improves the accuracy of the mean function μ_t and its gradient $\nabla\mu_t$ as estimators for f and its gradient ∇f around the sampled point. Consequently, the merit function, M_t^{μ}, closely approximates M^f and thus effectively validates convergence to the local (first-order) saddle point. As estimates improve with iterations, the loop in Alg. 2 is expected to terminate at a local saddle point.

Next, we provide an intuitive explanation of why sampling at local Nash points of confidence bounds in (5) will lead to local saddle points of f in (1). As we sample local Nash points, the variance in the confidence bounds at those points will decrease, and LCB/UCB will get closer to each other, as shown by Lemma A.1 and Remark A.2. As we keep sampling, the variance will eventually become the noise variance, σ_z. As such, $\nabla\sigma \to 0$ and therefore $\nabla\mathrm{LCB}, \nabla\mathrm{UCB} \to \nabla\mu \to \nabla f$, and therefore finding local Nash points of the confidence bounds will eventually lead to first-order saddle points of μ and consequently of f. To confirm that we find a saddle, we verify the second-order condition (Prop. 3.4) for the final saddle point returned by Alg. 2. If this point does not satisfy the second-order conditions, we reinitialize our algorithm from a new initial point. We find that, in practice, our algorithms find LSPs on the first initialization more frequently than baseline methods.

4.4 Variants of BSP

BSP Expensive: Our BSP method in Alg. 2 aims to minimize queries of the function f by taking multiple Newton steps per query of f. However, the algorithm may become unstable if the confidence bounds $\mathrm{UCB}_{t+1}, \mathrm{LCB}_{t+1}$ do not accurately approximate f. Additionally, it is possible that querying f can often be inexpensive, for example, in the case of Reinforcement Learning in simulated environments (Sutton & Barto, 2018). In this case, we query f during each iteration of Alg. 2 after taking a single Newton step, in contrast to the multiple steps taken in Alg. 1 (lines 2-3). This approach is referred to as BSP-*expensive* since we make more queries of f, while our original algorithm in Alg. 2 is referred to as BSP-*efficient*. We demonstrate in our results that this variant can more effectively and efficiently solve complex scenarios than baseline methods under these conditions.

Exploration and Exploitation: In Alg. 2, we encourage more exploration by using LCB for minimization and UCB for maximization. Since the value of unexplored regions has high variance and thus a lower LCB value, the minimization procedure will explore those regions first (vice-versa for UCB and maximization). As such, we refer to our original proposed method, as BSP-*explore*. Alternatively, we can use LCB for maximization and UCB for minimization, thus promoting more exploitation by our algorithm. We will refer to this variant as BSP-*exploit*. In real-world scenarios, this approach might be suitable when optimizing a well-understood process, fine-tuning known models, or when domain knowledge allows for confidently focusing on exploitation.

5 Experiments

In this section, we evaluate the BSP algorithms presented in Alg. 2 and Sec. 4.4 across various test environments. We consider four versions: 1) BSP-*efficient-explore* (Ef-Xplore), 2) BSP-*efficient-exploit* (Ef-Xploit), 3) BSP-*expensive-explore* (Exp-Xplore), and 4) BSP-*expensive-exploit* (Exp-Xploit). Our experiments, designed around challenging baseline problems widely recognized in prior work, demonstrate that each algorithm excels in specific settings.

While GPs are most effective in lower-dimensional spaces, we evaluate our methods on problems with up to 10 dimensions. We acknowledge that our experiments may not fully capture the complexity of high-dimensional real-world applications; however, our primary goal is to establish a foundation for solving these problems. We view this work as a first step and aim to extend our approach to more complex real-world settings in future research.

1. Decaying Polynomial: In our first experiment, we examine the performance of our algorithms for a nonconvex-nonconcave objective taken from (Mazumdar et al., 2020; Gupta et al., 2024):

$$f_{\exp}(x,y) = \exp\left(-0.01\left(x^2+y^2\right)\right)\left(\left(0.3x^2+y\right)^2+\left(0.5y^2+x\right)^2\right). \tag{10}$$

This example is particularly difficult for three reasons: first, multiple LSPs exist. Second, the origin is a spurious saddle point which satisfies first-order conditions but not second-order conditions. Third, the function gradients decay to zero further from the origin, meaning that an iterative algorithm strays too far may "stall," taking smaller and smaller step sizes, but never converging to a fixed point.

2. High-Dimension Polynomial: We consider a sixth-order polynomial from Bertsimas et al. (2010):

$$\begin{aligned} f_{\text{bertsimas}}(x,y) = &-2x^6+12.2x^5-21.2x^4-6.2x+6.4x^3+4.7x^2-y^6+11y^5 \\ &-43.3y^4+10y+74.8y^3-56.9y^2+4.1xy+0.1y^2x^2-0.4y^2x-0.4x^2y. \end{aligned} \tag{11}$$

The decision space is within $[x_{\min}=-0.95, x_{\max}=3.2]\times[y_{\min}=-0.45, y_{\max}=4.4]$. The objective, $f_{\text{poly}}(x,y)$, is nonconvex-nonconcave and has multiple LSPs (Defn. 3.2). We form a high-dimension $(2n)$-D polynomial by letting, for $\vec{x}\in\mathbb{R}^n, \vec{y}\in\mathbb{R}^n$,

$$f_{\text{poly}}(\vec{x},\vec{y}) = \sum_{i=1}^{n} f_{\text{bertsimas}}(x_i,y_i). \tag{12}$$

We set $n=5$ to evaluate how well our proposed algorithms identify LSPs with more dimensions. A 10-dimensional space is sufficient for many relevant applications, though we note that it may constitute a relatively low dimensionality for others.

3. ARIMA Tracking Model Predictive Controller (MPC): Finally, we test our algorithms on a more realistic zero-sum game involving an ARIMA process that synthesizes a discrete time 1D time series of length F, denoted by $s_F\in\mathbb{R}^F$, for a model predictive controller to track. This setup mirrors real-world systems like that of Stent et al. (2024), in which an autonomous system corrects for distracted human driving. We represent the ARIMA process for initial state $s_0\in\mathbb{R}$ and model parameters $\alpha\in\mathbb{R}, \beta\in\mathbb{R}$: $s_F=\text{ARIMA}(s_0,\alpha,\beta)$. The MPC takes the ARIMA time series, s_F, as input and returns a controller cost $f_{\text{MPC}}\in\mathbb{R}$ incurred while tracking the given time series. The MPC solves an optimization problem with quadratic costs and linear constraints, encapsulating vehicle dynamics and control limits. The optimization problem is represented as $f_{\text{MPC}}=\text{MPC}(A,B,s_0,s_F)$, returning the final overall cost of tracking the ARIMA-generated time series s_F, initial state, s_0 and model dynamics A,B. Further details can be found in Appendix B.6.

Zero-Sum Game Formulation: We formulate the interaction between the ARIMA forecaster and the MPC as a zero-sum game. The antagonist selects the ARIMA parameters α,β to generate difficult-to-track time series forecasts, s_F, resulting in a higher MPC cost f_{MPC}. In many scenarios, we want to find model dynamics that are robust and can effectively handle various tracking signals. As such, the protagonist chooses the MPC model dynamics parameters A,B to accurately track s_F and minimize the controller cost. This competitive scenario is formulated as follows:

$$f_{\text{MPC}} = \text{MPC}(\underbrace{A,B}_{\text{protagonist}}, \hat{s}_0, s_F=\text{ARIMA}(s_0, \underbrace{\alpha,\beta}_{\text{antagonist}})). \tag{13}$$

This game is motivated by real-world scenarios requiring robust controllers for adversarial and out-of-distribution inputs, and it has multiple LSPs (Defn. 3.2).

Experimental Setup: We compare our algorithms in two settings. In the first, we initialize our algorithms with a large number of sample points, modeling a scenario where the objective is well understood. In the second, we initialize algorithms with a small number of sample points, modeling a scenario where obtaining samples is expensive. In all our experiments, we use a squared exponential kernel, where the kernel value of two data points x_i and x_j is given by:

$$k\left(x_i, x_j\right) = \sigma_f^2 \exp\left[-\frac{1}{2}\frac{\left(x_i - x_j\right)^T\left(x_i - x_j\right)}{\sigma_l^2}\right], \tag{14}$$

where σ_f (signal variance) and σ_l (signal length scale) are hyperparameters. We learn these hyperparameters via maximum likelihood to initialize our algorithms. In all experiments, we assume that we observe noisy measurements of the underlying function. To ensure that algorithms are initialized at points with a non-zero gradient in the decaying polynomial example (top row of Fig. 1), we ensure these are selected from the non-flat regions of the function. All experiments are performed with 20 seeds for each algorithm. For the scenario with a limited number of initially sampled points, we sampled 50 points for the decaying and high-dimension polynomial objectives, and 10 points for the ARIMA-MPC objective. We present the exact experimental setup for each test case in Appendix B.

5.1 Experimental Results

Fig. 1 summarizes the performance of the Ef-Xplore and Exp-Xploit variants of our algorithms for the first two scenarios mentioned above; for results related to Ef-Xploit and Exp-Xplore, we refer the reader to Fig. 4 in Appendix C, where we report similar results to Ef-Xplore and Exp-Xploit, respectively. The middle column of Fig. 1 considers test cases with a large number of initially sampled points, while the right column considers test cases with a limited number of initially sampled points. We sample 1000 initial points for the decaying polynomial and 500 initial points for the high-dimensional polynomial and for ARIMA-MPC (for which we report results in Fig. 2).

We assess the performance of our algorithms using the merit function from Eq. (8), M^f, which is calculated based on the true gradients of the underlying function, rather than the confidence bounds employed in the actual algorithm. As previously mentioned, M^f will attain a global minimum (of 0) when the first-order conditions in Prop. 3.3 are satisfied. This evaluation metric offers a direct measure of the algorithms' effectiveness in identifying local saddle points of the underlying function. We also compute success rates for each algorithm in finding a saddle point (Table 2). *We only include a run as a success if the BSP solution satisfies second-order sufficient conditions Prop. 3.4 according to the ground truth derivatives* ∇f *and* $\nabla^2 f$. Lastly, we also compute rates of convergence, measured in Newton steps, for our algorithms. Overall, our results show that our first-order algorithm finds saddle points at a higher rate than all baselines, even outperforming the success rate of a state-of-the-art second-order method!

Baselines: We consider multiple baseline algorithms: naive random sampling (Random), gradient descent-ascent with finite differencing (GDA with FD), and local symplectic surgery (LSS), a state-of-the-art baseline from Mazumdar et al. (2019), and four algorithm variants from Maheshwari et al. (2022). The first three methods assume access to zeroth-, first-, and second-order derivatives, respectively, and the last is a zeroth-order method. In the random sampling baseline, we uniformly sample a fixed number of points (x, y) from the hyperbox $\{x, y : x_{\min} \leq x \leq x_{\max},\ y_{\min} \leq y \leq y_{\max}\}$, and retain the point with the lowest real merit function value (M^f). In the gradient descent-ascent baseline, we employ finite differencing to estimate each player's gradient and use a Wolfe linesearch to select step sizes. This approach allows for a more directed search compared to random sampling. Approximating the gradient with finite differencing provides a fair comparison between our method and a first-order approach using zeroth-order samples. For the decaying and high-dimension polynomial examples, we compare with LSS. LSS requires access to function gradients and Hessians; rather than provide finite differenced estimates (which can be extremely noisy and require excessive function evaluations), we provide oracle access to the true function derivatives and corrupt sampels with standard Gaussian noise as explained in Appendix B.

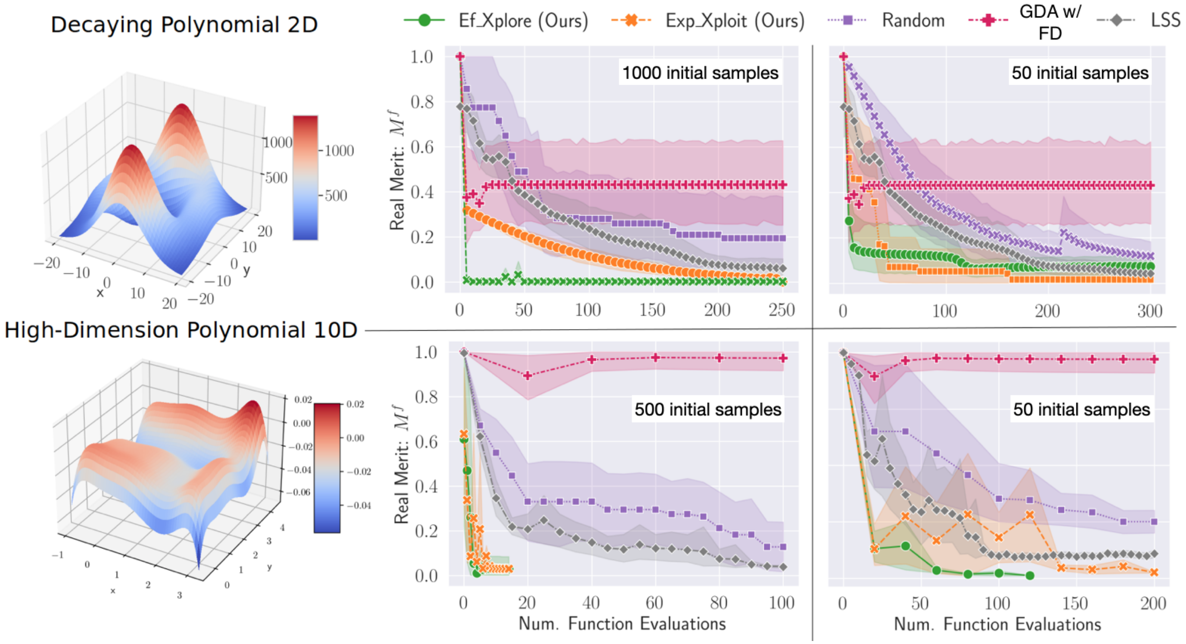

Figure 1: **Comparisons of selected algorithm variants with baselines:** We compare variants of our proposed algorithms with baseline methods across two domains (rows), the decaying and high-dimension polynomials, landscapes of which are shown in the first column. The middle column considers test cases with a large number of initially sampled points, while the right column examines test cases with a limited number of initially sampled points. In each case, we report the value of (real) merit function M^f vs. the number of underlying function evaluations. *Key takeaway:* Generally, Ef-Xplore converges faster with a large number of initial samples by taking multiple Newton steps at each step in order to exploit the accurate prior while Exp-Xploit exhibits quicker convergence with limited samples by taking single Newton steps to avoid unfavorable regions amid uncertainty. Finally, we find that Ef-Xplore and Exp-Xploit converge faster than all three baseline methods, indicating the benefit of the GP surrogate in improving convergence compared to baselines which are often unable to converge.

We provide an additional set of four baselines (OGDA-RR, OGDA-WR, SGDA-RR, SGDA-WR) from (Maheshwari et al., 2022) for the decaying and high-dimension polynomial experiments. These algorithms are designed for and validated on zeroth-order convex-concave problems but can be applied in general settings.

Analysis of a Large Number of Initially Sampled Points: In this setting, we observed that the *exploit* variants of our proposed algorithms, Ef-Xploit (blue) and Exp-Xploit (orange), demonstrated the fastest convergence. This outcome is expected since the accuracy of the confidence bounds was higher, reducing the need for exploration. Overall, Ef-Xploit (blue) achieved the fastest convergence in these experiments due to its ability to take multiple accurate Newton steps. In the decaying polynomial example from Fig. 1, Ef-Xplore converges quickly as well as taking single Newton steps helps the algorithm converge in this particularly complicated landscape. In contrast, the *explore* algorithms had slower convergence, as they prioritize exploration. This general pattern continues to hold in Fig. 2, for the ARIMA-MPC scenario.

Analysis of Limited Number of Initially Sampled Points: In this setting, we observed that the *explore* variants of our proposed algorithms, Ef-Xplore (green) and Exp-Xplore (red), achieved the best performance. Notably, the algorithm variant Exp-Xplore (red) demonstrated fast convergence, which can be attributed to its exploration approach and avoiding multiple incorrect Newton steps in the face of uncertainty. The *exploit* variants, Ef-Xploit (blue) and Exp-Xploit (orange) exhibited slower convergence, as they relied too heavily on prior information and consequently took incorrect steps. Specifically, the Ef-Xploit (blue) variant failed to converge for some seeds since it took incorrect Newton steps and was unable to explore. Finally, the *expensive* variants, in general, are more stable in this setting, as they only take single Newton steps and are less likely to reach unfavorable regions. Exp-Xploit (orange), for example, converges to a real merit vaue $M^f = 0$ for both experiments. We note that for the decaying polynomial example, we still see the efficient variants converge faster, and this result reflects the complexity of the objective landscape.

Comparisons with Baselines: Random sampling and GDA with FD never converge, though random sampling reduces the merit function value. The noise introduced by finite differencing renders GDA with FD ineffective in this problem setting. Despite reaching a low merit function value, we note that LSS fails to converge in many of these scenarios. In the decaying polynomial example, we see LSS iterate outward far from the origin where ∇f becomes very small and no saddle points exist. Note that this behavior is consistent with Mazumdar et al. (2019), which claims only that if LSS converges, it finds a saddle. Our experiments on the decaying and high-dimension polynomials indicate that LSS fails to converge to a saddle point far more often than BSP (Table 2). Furthermore, we find that OGDA-RR, OGDA-WR, SGDA-RR, and SGDA-WR (Maheshwari et al., 2022) always fail to find saddles in the two polynomial settings.

Runtime and Success Rate: We compare the runtimes of each algorithm variant in terms of the total number of Newton steps taken. In Table 1, we present the total number of Newton steps taken by each algorithm variant over 10 seeds for scenarios with a limited number of initially sampled points. Neglecting the time to query the underlying function, the *efficient* variants require 400-500% more time compared to the *expensive* variants. This outcome is expected, as *expensive* variants only take one Newton step between each sample of the underlying function evaluation. Furthermore, as anticipated, the *explore* variants require more Newton steps due to their exploration approach, but the difference is not substantial.

Domain	Type of steps	Efficient	Expensive
High-Dimension Polynomial	Explore	2858	1464
	Exploit	2613	**1381**
Decaying Polynomial	Explore	2690	**1893**
	Exploit	3000	3000
ARIMA-MPC	Explore	1053	353
	Exploit	815	**229**

Table 1: **Runtime Comparison:** In this table, we show the number of Newton steps taken by each algorithm variant for the limited number of initially sampled points scenarios over 10 seeds.

Domain	Ef-Xplore	Exp-Xplore	Ef-Xploit	Exp-Xploit	LSS	SGDA-RR	SGDA-WR	OGDA-RR	OGDA-WR	Random	GDA w/ FD
Decaying Polynomial	**60**%	**60**%	30%	50%	15%	0%	0%	0%	0%	0%	0%
High-Dimension Polynomial	95%	**100**%	65%	80%	70%	0%	0%	0%	0%	0%	0%
ARIMA-MPC	90%	**95**%	75%	85%	-	-	-	-	-	-	-

Table 2: **Success Rate:** In this table, we show the percent of successful seeds out of 20 seeds for each algorithm variant for the limited number of initially sampled points scenarios. We show results on our baseline algorithms for two of the experiments. We adapt LSS to the black box setting by providing an oracle to sample derivative information. To avoid including spurious saddle points as successes, we report the success rate according to the true second-order conditions, defined in Prop. 3.4, at the first solution BSP finds (i.e., without reinitialization). The methods proposed by Maheshwari et al. (2022) always fail to find saddle points.

In Table 2, we display the full results for the percentage of successful seeds out of 20 seeds for each algorithm variant in scenarios with a limited number of initially sampled points. The results reveal that the *explore* variants exhibit more reliable convergence to saddle points compared to the *exploit* variants. This outcome is expected, as *explore* variants emphasize exploration and, therefore, are more likely to converge. Additionally, the *expensive* variants demonstrate greater stability in this setting, as they only take single Newton steps and are less prone to reaching unfavorable regions. The Ef-Xploit variant exhibits the lowest success rate in convergence; it did not achieve a 100% success rate in cases with a limited number of initial samples, as it relied on exploitation and took incorrect Newton steps due to high uncertainty. We note that LSS often fails to converge to a LSP due to algorithmic assumptions (which only guarantee that when LSS converges, it finds a saddle) or domain-specific factors (iterating towards low-gradient regions of the objective landscape). We find that Exp-Xploit, Exp-Xplore, and Ef-Xplore find saddle points strictly more often than LSS, and that Ef-Xploit performs similarly or better depending on the experiment. Neither of our other baselines

(Random and GDA with FD) find solutions for these problems. Lastly, none of the algorithms from Maheshwari et al. (2022) (OGDA-RR, OGDA-WR, SGDA-RR, SGDA-WR) ever find a LSP, confirming that methods for convex-concave settings are unable to find solutions for these problems of interest.

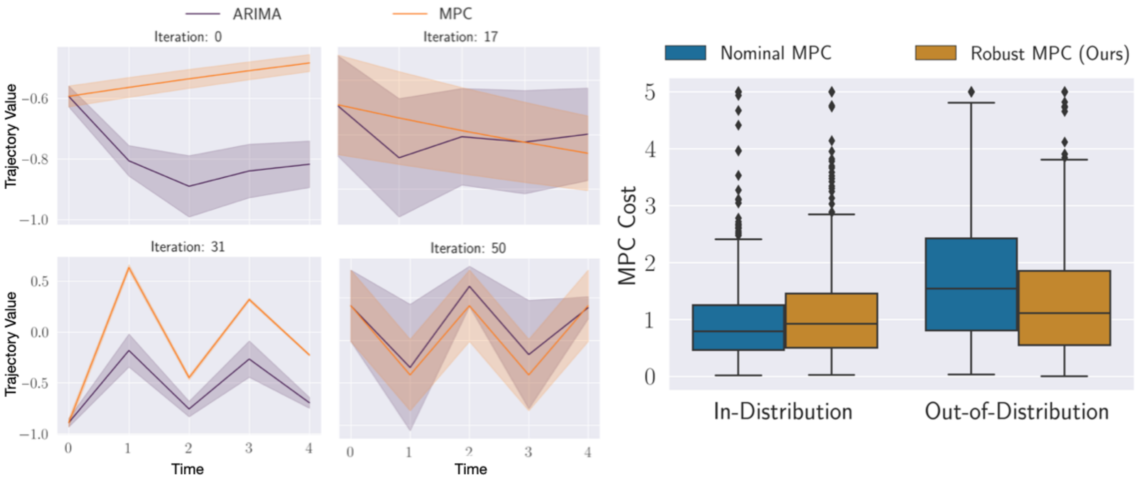

Figure 2: **Saddle Point Optimization with BSP leads to Robust MPC on out-of-distribution (OoD) data:** On the left, we display the MPC tracking of the timeseries generated by the ARIMA model at various iterations for the Ef-Xplore variant. The ARIMA target trajectory is depicted in purple, while the corresponding MPC tracking is illustrated in orange. Initially, MPC performs poorly (iteration 0), but gradually improves its tracking (iteration 17). Consequently, the ARIMA makes tracking more challenging for the MPC (iteration 31), until they both reach equilibrium (iteration 50). On the right, we compare the final robust MPC parameters (orange), obtained through our algorithm, to the nominal MPC parameters (blue) on in-distribution data (left column) and OoD data (right column). *Key takeaway*: significantly, the robust MPC successfully identifies robust MPC parameters and achieves **27.6%** lower mean MPC cost on OoD data compared to nominal MPC without reducing performance on in-distribution data.

Key takeaways from our experimental results: Our experimental results highlight several important insights about our algorithms variants. 1) The *efficient* variants require fewer underlying function evaluations and will work best when the prior is accurate. 2) The *expensive* variants offer faster runtimes and will provide more stable convergence when the prior is inaccurate. 3) The *explore* variants provide a higher success rate when the number of initial samples is limited. 4) The *exploit* variants exhibit faster convergence in the setting with a large number of initial samples. These findings suggest that the choice of an algorithm variant should depend on the specific characteristics of the problem at hand, such as the number of available initial samples, runtime requirements, and domain knowledge of the underlying objective. Moreover, we find that our methods converge faster than simple baselines based on random sampling and finite differencing, and that they converge faster (and more reliably) than the state-of-the-art (second-order) LSS algorithm when it is adapted to the black-box setting. This occurs even though our algorithm uses first-order methods to find critical points and then verifies the result with second-order saddle point conditions.

Performance of BSP Variants in ARIMA-MPC Example: In Fig. 2, we evaluate the performance of the MPC parameters found by our algorithms. Specifically, we compare the performance of the MPC parameters obtained at the end of the Ef-Xplore variant on both the in-distribution and out-of-distribution (OoD) ARIMA forecasting timeseries datasets. When MPC operates at a LSP, we can expect it to be robust to perturbations in ARIMA parameters and therefore to OoD time series. Indeed, controller parameters found by our algorithm toughly match in-distribution performance and achieve 27.6% lower mean MPC cost on OoD data, thus showcasing our algorithm's ability to locate saddle points which correspond to robust performance on OoD data without reducing performance on in-distribution data. We provide the exact details of this experiment in Appendix B, and present full convergence results on the ARIMA-MPC example in Fig. 4. These results mirror those of previous experiments, and we find that the exploit variants converge faster with many initial samples while the efficient variants do so with limited initial samples. Due to the poor performance of the baselines RANDOM, LSS, OGDA-RR, OGDA-WR, SGDA-RR, and SGDA-WR on the simpler polynomial examples, we do not add them for the ARIMA-MPC experimental setting.

Summary: Our experimental results demonstrate that our proposed BSP algorithms converge faster, sample more efficiently, and produce more robust solutions than existing methods in a variety of black-box saddle point optimization problems.

6 Conclusion

We present a BO-inspired framework for identifying local saddle points for an unknown objective function, f, with zeroth-order samples. We frame the problem of finding local saddle points for an unknown objective function as a two-level procedure. A *low-level* algorithm constructs a general-sum game from a Gaussian process which approximates the unknown function f, and solves for the local Nash points of this game by finding the roots of a system of nonlinear algebraic equations. A *high-level* algorithm queries the points returned by the *low-level* algorithm to refine the GP estimate and monitor convergence toward local saddle points of the original problem. We validate the effectiveness of our algorithm through extensive Monte Carlo testing on multiple examples. Our results outperform several approaches, including a state-of-the-art approach (LSS) our zeroth-order method outperforms despite providing it with a gradient oracle.

Limitations and Future Work: While our proposed framework shows promising results, we plan to address certain limitations in future work. First, we plan to extend our strong experimental efforts with more thorough theoretical guarantees using promising techniques introduced by recent works (Han et al., 2024; Gemp et al., 2024) which propose objective functions using an equivalent definition of Nash equilibria and for which local optima approximate Nash equilibria. Moreover, Han et al. (2024) introduce a GP-based game solver based on trust-region methods and an associated no-regret proof which may be relevant for developing future theoretical results. Second, our method currently finds saddle points by solving for first-order critical points, checking that they satisfy second-order conditions, and reinitializing the algorithm with the additional data if not. In future work, we intend to explore ways to directly incorporate second-order sufficient conditions (Prop. 3.4) to enhance the performance within the general-sum low-level game of our framework. Third, we recognize that GPs can become prohibitive in higher dimensions, limiting this method's application to some real-world problems. As such, we plan to identify methods to adapt the framework to increasing dimensionality. Fourth, to further demonstrate our framework's adaptability, we will test our algorithm with other acquisition functions, such as knowledge gradient (Ryzhov et al., 2012) and entropy search (Hennig & Schuler, 2012). Lastly, we plan to apply our method to additional real-world scenarios, including stock market portfolio optimization, online marketplace auctions, and robust dynamic control by extending our MPC example.

References

Leonard Adolphs, Hadi Daneshmand, Aurelien Lucchi, and Thomas Hofmann. Local saddle point optimization: A curvature exploitation approach. In Kamalika Chaudhuri and Masashi Sugiyama (eds.), *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pp. 486–495. PMLR, 16–18 Apr 2019. URL `https://proceedings.mlr.press/v89/adolphs19a.html`.

Shubhankar Agarwal, David Fridovich-Keil, and Sandeep P Chinchali. Robust forecasting for robotic control: A game-theoretic approach. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, 2019.

David Balduzzi, Sebastien Racaniere, James Martens, Jakob Foerster, Karl Tuyls, and Thore Graepel. The mechanics of n-player differentiable games. In *International Conference on Machine Learning*, pp. 354–363. PMLR, 2018.

Tamer Başar and Geert Jan Olsder. *Dynamic Noncooperative Game Theory, 2nd Edition.* Society for Industrial and Applied Mathematics, 1998. doi: 10.1137/1.9781611971132. URL `https://epubs.siam.org/doi/abs/10.1137/1.9781611971132`.

Michele Benzi, Gene H. Golub, and Jörg Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 2005. doi: 10.1017/S0962492904000212.

Dimitris Bertsimas, Omid Nohadani, and Kwong Meng Teo. Robust optimization for unconstrained simulation-based problems. *Operations Research*, 58(1):161–178, 2010. doi: 10.1287/opre.1090.0715. URL `https://doi.org/10.1287/opre.1090.0715`.

Ilija Bogunovic, Jonathan Scarlett, Stefanie Jegelka, and Volkan Cevher. Adversarially robust optimization with gaussian processes. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *ArXiv*, abs/1012.2599, 2010a.

Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, 2010b. URL `https://arxiv.org/abs/1012.2599`.

Ashish Cherukuri, Bahman Gharesifard, and Jorge Cortes. Saddle-point dynamics: conditions for asymptotic stability of saddle points. *SIAM Journal on Control and Optimization*, 55(1):486–511, 2017.

Wang Chi Cheung, David Simchi-Levi, and Ruihao Zhu. Reinforcement learning for non-stationary markov decision processes: The blessing of (more) optimism. In *International Conference on Machine Learning*, pp. 1843–1854. PMLR, 2020.

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in neural information processing systems*, 27, 2014.

Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2): 408–423, 2013.

John D'Errico. nearestspd. `https://www.mathworks.com/matlabcentral/fileexchange/42885-nearestspd`. Accessed: 2025-04-07.

Lukas Fröhlich, Edgar Klenske, Julia Vinogradska, Christian Daniel, and Melanie Zeilinger. Noisy-input entropy search for efficient robust bayesian optimization. In Silvia Chiappa and Roberto Calandra (eds.), *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 2262–2272. PMLR, 26–28 Aug 2020. URL `https://proceedings.mlr.press/v108/frohlich20a.html`.

Ian Gemp, Luke Marris, and Georgios Piliouras. Approximating nash equilibria in normal-form games via stochastic optimization. *arXiv preprint arXiv:2310.06689*, 2024.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11): 139–144, 2020.

Kushagra Gupta, Xinjie Liu, Ufuk Topcu, and David Fridovich-Keil. Second-order algorithms for finding local nash equilibria in zero-sum games. *arXiv preprint arXiv:2406.03565*, 2024.

Minbiao Han, Fengxue Zhang, and Yuxin Chen. No-regret learning of nash equilibrium for black-box games via gaussian processes. 2024.

Graeme Henkelman, Blas P. Uberuaga, and Hannes Jónsson. A climbing image nudged elastic band method for finding saddle points and minimum energy paths. *The Journal of Chemical Physics*, 113(22):9901–9904, 12 2000.

Philipp Hennig and Christian J. Schuler. Entropy search for information-efficient global optimization. *J. Mach. Learn. Res.*, 13(null):1809–1837, jun 2012. ISSN 1532-4435.

Chi Jin, Praneeth Netrapalli, Rong Ge, Sham M Kakade, and Michael I Jordan. On nonconvex optimization for machine learning: Gradients, stochasticity, and saddle points. *Journal of the ACM (JACM)*, 68(2): 1–29, 2021.

G. M. Korpelevich. The extragradient method for finding saddle points and other problems. In *Ekonomika i Matematicheskie Metody*, volume 12, pp. 747–756, 1976.

Daniel Lizotte, Tao Wang, Michael Bowling, and Dale Schuurmans. Automatic gait optimization with gaussian process regression. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pp. 944–949, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

Eric Luxenberg, Philipp Schiele, and Stephen Boyd. Robust bond portfolio construction via convex-concave saddle point optimization. *arXiv preprint arXiv:2212.02570*, 2022.

Chinmay Maheshwari, Chih-Yuan Chiu, Eric Mazumdar, Shankar Sastry, and Lillian Ratliff. Zeroth-order methods for convex-concave min-max problems: Applications to decision-dependent risk minimization. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera (eds.), *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pp. 6702–6734. PMLR, 28–30 Mar 2022.

Eric Mazumdar, Lillian J Ratliff, and S Shankar Sastry. On gradient-based learning in continuous games. *SIAM Journal on Mathematics of Data Science*, 2(1):103–131, 2020.

Eric V. Mazumdar, Michael I. Jordan, and S. Shankar Sastry. On finding local Nash equilibria (and only local Nash equilibria) in zero-sum games. *ArXiv*, abs/1901.00838, 2019.

Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. The numerics of gans. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

J. Močkus. On bayesian methods for seeking the extremum. In G. I. Marchuk (ed.), *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*, pp. 400–404, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.

José Moura and David Hutchison. Game theory for multi-access edge computing: Survey, use cases, and future trends. *IEEE Communications Surveys & Tutorials*, 21(1):260–288, 2019. doi: 10.1109/COMST.2018.2863030.

Arkadi Nemirovski. Prox-method with rate of convergence o(1/t) for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1):229–251, 2004.

J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. ISBN 9780387400655. URL `https://books.google.co.in/books?id=VbHYoSyelFcC`.

FM Nyikosa. *Adaptive Bayesian optimization for dynamic problems*. PhD thesis, University of Oxford, 2018.

Razvan Pascanu, Yann N. Dauphin, Surya Ganguli, and Yoshua Bengio. On the saddle point problem for non-convex optimization, 2014. URL `https://arxiv.org/abs/1405.4604`.

Muhammad I. Qureshi and Usman A. Khan. Distributed saddle point problems for strongly concave-convex functions. *IEEE Transactions on Signal and Information Processing over Networks*, 9:679–690, 2023. doi: 10.1109/TSIPN.2023.3317807.

Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pp. 63–71. Springer, 2003.

Lillian J Ratliff, Samuel A Burden, and S Shankar Sastry. On the characterization of local nash equilibria in continuous games. *IEEE transactions on automatic control*, 61(8):2301–2307, 2016.

J. Revels, M. Lubin, and T. Papamarkou. Forward-mode automatic differentiation in Julia. *arXiv:1607.07892 [cs.MS]*, 2016. URL `https://arxiv.org/abs/1607.07892`.

Ilya O. Ryzhov, Warren B. Powell, and Peter I. Frazier. The knowledge gradient algorithm for a general class of online learning problems. *Operations Research*, 60(1):180–195, 2012.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016. doi: 10.1109/JPROC.2015.2494218.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 1015–1022, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Simon Stent, Andrew P. Best, Shabnam Hakimi, Guy Rosman, Emily S. Sumner, and Jonathan DeCastro. Adversarial simulation for developing and testing assistive driving technology, Jul 2024.

Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. second. *A Bradford Book*, 2018.

Jørgen Tind. *Saddle Point Theory and Optimality Conditions*, pp. 3355–3357. Springer US, Boston, MA, 2009. ISBN 978-0-387-74759-0. doi: 10.1007/978-0-387-74759-0_574. URL `https://doi.org/10.1007/978-0-387-74759-0_574`.

Paul Tseng. On linear convergence of iterative methods for the variational inequality problem. *Journal of Computational and Applied Mathematics*, 60:237–252, 1995.

J. v. Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928. doi: 10.1007/BF01448847. URL `https://doi.org/10.1007/BF01448847`.

Zhongruo Wang, Krishnakumar Balasubramanian, Shiqian Ma, and Meisam Razaviyayn. Zeroth-order algorithms for nonconvex–strongly-concave minimax problems with improved complexities. *Journal of Global Optimization*, 2022. doi: 10.1007/s10898-022-01160-0. URL `https://doi.org/10.1007/s10898-022-01160-0`.

Julia Westermayr and Philipp Marquetand. Machine learning for electronically excited states of molecules. *Chemical Reviews*, 121(16):9873–9926, 08 2021. doi: 10.1021/acs.chemrev.0c00749. URL `https://doi.org/10.1021/acs.chemrev.0c00749`.

Yi Yang, Zhigang Ma, Feiping Nie, Xiaojun Chang, and Alexander G Hauptmann. Multi-class active learning by uncertainty sampling with diversity maximization. *International Journal of Computer Vision*, 113: 113–127, 2015.

Appendix for *A Framework for Finding Local Saddle Points in Two-Player Zero-Sum Black-Box Games*

A Technical Insights **17**
A.1 UCB and LCB will approach one another at sampled points 17
A.2 Variance of the Sampled Point 19

B Experimental Details **19**
B.1 Newton's method implementation details 19
B.2 Gradient Descent-Ascent with Finite Differencing implementation details 20
B.3 Local Symplectic Surgery implementation details 20
B.4 Decaying polynomial implementation details 20
B.5 High-dimension polynomial implementation details 20
B.6 ARIMA tracking Model Predictive Controller (MPC) 20
B.7 Robust MPC experimental details 21
B.8 Updating Hyperparameters 21

C Ablation Studies **21**
C.1 An Example of Unsuccessful Runs using the Gradient Descent-Ascent with Finite Differencing Baseline 23

A Technical Insights

In this section, we delve into the convergence properties of our proposed algorithms. As mentioned in Sec. 4.3, when we sample local Nash points obtained by Alg. 1, the variance in the confidence bounds at that local Nash point decreases, and LCB/UCB converge towards each other eventually at the local Nash point. In Lemma A.1, we demonstrate that with zero observation noise, the confidence bounds at a sampled point are equal, i.e., LCB = UCB. Subsequently, in Remark A.2, we explore the more general case involving non-zero observation noise, positing that as we repeatedly sample in close proximity to the same point, the variance of the sampled point becomes predominantly dependent on the observation noise, resulting in LCB ≈ UCB. Lastly, in Appendix A.2, we provide experimental evidence to corroborate the decrease in the variance of the sampled point during our algorithm's execution.

Consequently, these results attest that as we sample the local Nash point, the variance in the GP at that point will decrease, and eventually becomes a observation noise variance. As such, the gradient of the variance, $\nabla\sigma \to 0$, and therefore $\nabla\text{LCB}, \nabla\text{UCB} \to \nabla\mu$. As stated in Sec. 4.3, since $\nabla\mu$ will become a reliable estimator of the true gradients of the unknown function, ∇f, finding local Nash points of the confidence bounds will eventually lead to local saddle points of μ and consequently of f. This convergence property is a key feature of our proposed algorithms, ensuring that the method converges to a solution that represents a local saddle point of the underlying unknown function.

A.1 UCB and LCB will approach one another at sampled points

For the ease of the proofs, we will focus on the case when $r = f(x) + z$, where $z \sim \mathcal{N}\left(0, \sigma_z^2\right)$. The proof can easily be generalized for $r = f(x, y) + z$.

We denote the set of observed points as $\mathbf{X} = \{x_1, x_2, \ldots, x_n\}$ and $r = \{r_1, r_2, \ldots, r_n\}$. Consider that the point, x_*, has already been sampled, as such $x_* = x_i$ for some $i \in \{1 \ldots N\}$. Now, recall the predictive variance of the point x_* is:

$$\sigma(x_*|\mathbf{X}, r, x_*) = K\left(x_*, x_*\right) - k_*^\top (\Sigma(\mathbf{X}, \mathbf{X}) + \sigma_z^2 I)^{-1} k_*, \tag{15}$$

where $k_* = \left[K(x_1, x_*), K(x_2, x_*), \ldots, K(x_n, x_*)\right]^\top$, and $\Sigma(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{n\times n}$ given by

$$\begin{bmatrix} K\left(x_1, x_1\right) & K\left(x_1, x_2\right) & \cdots & K\left(x_1, x_n\right) \\ \vdots & \vdots & \ddots & \vdots \\ K\left(x_n, x_1\right) & K\left(x_n, x_2\right) & \cdots & K\left(x_n, x_n\right) \end{bmatrix}. \tag{16}$$

Lemma A.1 (Equality of UCB_t and LCB_t at sampled points under zero observation noise)**.** *Consider the upper confidence bound,* UCB_t*, and the lower confidence bound,* LCB_t*, for a given time step t. In the case of zero observation noise, i.e., $z_t = 0$, the confidence bounds become equal for any sampled point x such that* $\mathrm{UCB}_t(x) = \mathrm{LCB}_t(x)$*.*

Proof. Since, $z_t = 0$, the predictive variance at the point x_* is:

$$\sigma(x_*|\mathbf{X}, r, x_*) = K\left(x_*, x_*\right) - k_*^\top \Sigma^{-1} k_*. \tag{17}$$

Let the point x_* be some point $x_i \in \mathbf{X}$, i.e., $x_* = x_i$ for some $1 \leq i \leq n$. Then, the corresponding kernel vector is given by the i-th column of the covariance matrix $\Sigma(\mathbf{X}, \mathbf{X})$, so $k_* = k_*^\top = \Sigma_{:,i}$ and the kernel value is $K(x_*, x_*) = K(x_i, x_i)$. The variance at the new point x_* becomes:

$$\sigma(x_*|\mathbf{X}, r, x_*) = K(x_i, x_i) - \Sigma_{:,i}^\top \Sigma^{-1} \Sigma_{:,i}. \tag{18}$$

Next, we have:

$$\Sigma^{-1}\Sigma_{:,i} = \mathbf{e}_i, \tag{19}$$

where $\mathbf{e}_i$ is the i-th standard basis vector. This can be seen from the property of the inverse matrix, i.e., $\Sigma(\mathbf{X}, \mathbf{X})^{-1}\Sigma(\mathbf{X}, \mathbf{X}) = I$, where I is the identity matrix.

Thus, the variance at the new point x_* simplifies to:

$$\sigma(x_*|\mathbf{X}, r, x_*) = K(x_i, x_i) - \Sigma_{:,i}^\top \mathbf{e}_i = K(x_i, x_i) - \Sigma_{i,i}. \tag{20}$$

Since x_* is a previously sampled point, the kernel function $K(x_i, x_i)$ and $\Sigma_{i,i}$ are equal to 1. Thus, the variance at x_* is:

$$\sigma(x_*|\mathbf{X}, r, x_*) = 1 - 1 = 0. \tag{21}$$

This shows that the variance at a previously sampled point x_* is zero in the no observation noise case, for any kernel function.

UCB and LCB at the point x_* are given by:

$$\mathrm{UCB}_t(x_*) = \mu_t\left(x_*\right) + \beta_t \sigma_t\left(x_*\right), \quad \mathrm{LCB}_t(x_*) = \mu_t\left(x_*\right) - \beta_t \sigma_t\left(x_*\right). \tag{22}$$

Since we just showed the sampled point, x_*, the variance $\sigma_t\left(x_*\right) = 0$. Then:

$$\mathrm{UCB}_t(x_*) = \mathrm{LCB}_t(x_*) = \mu_t\left(x_*\right) \tag{23}$$

□

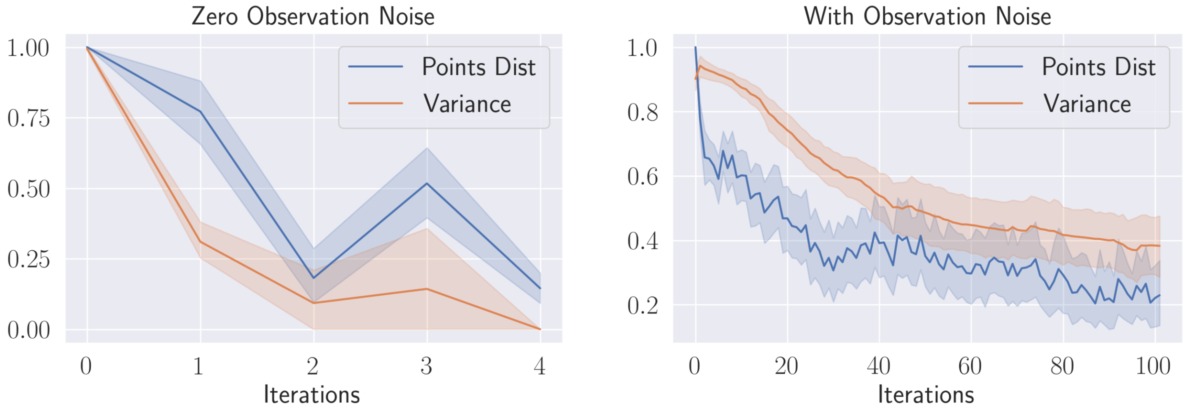

Figure 3: **The variance at the sampled points decreases over time.**

Remark A.2 (Approximate equality of UCB and LCB at sampled points under observation noise). Consider the upper confidence bound, UCB_t, and the lower confidence bound, LCB_t, for a given time step t. In the presence of observation noise, i.e., $z_t \sim \mathcal{N}\left(0, \sigma_z^2\right)$, when the same point is sampled repeatedly, or nearby points are sampled, the predictive variance $\sigma(x)$ approaches the observation noise σ_z^2. This is based on the fact after repeated sampling, the only uncertainty left about the sampling point will be due to the observation noise. As such, as the predictive variance becomes smaller due to repeated sampling, the confidence bounds at the sampled point x will have $\mathrm{UCB}_t(x) \approx \mathrm{LCB}_t(x)$.

A.2 Variance of the Sampled Point

In Fig. 3, we demonstrate the reduction in the variance of the sampled points as we approach the saddle point during the execution of our algorithms. We compare cases with and without observation noise for a convex-concave objective function of the form $ax^2 + bxy - cy^2$, where coefficients $a, b, c > 0$, scenario using 10 seeds. We note that LSPs will be locally convex-concave in their neighborhoods and so these results will apply locally in those scenarios. The blue line represents the distance between two consecutive points for each function evaluation, while the orange line indicates the variance in the GP prior at the sampled points. The primary observation is that as we take smaller Newton steps and sample points close to each other, the variance at those points decreases, thus increasing the accuracy of the mean function μ_t and its gradient $\nabla \mu_t$ as estimators for the objective function f and its gradient ∇f around the sampled points.

B Experimental Details

We provide the exact implementation details of all the experiments. Starting with compute, all experiments were conducted on a desktop computer equipped with an AMD Ryzen 9 5900X CPU and 32 GB RAM. No GPUs were required for these experiments.

B.1 Newton's method implementation details

1. Invertibility of Jacobian: To solve for the Newton step $p(x_t, y_t)$ in (7), the Jacobian matrix $J(x_t, y_t)$ must be non-singular. Therefore, at each new iterate, we need to verify the invertibility of $J(x_t, y_t)$. A common way to ensure invertibility is by adding a constant factor λI to the diagonal. In this work, we choose the damping factor λ by using the L2 norm of the difference between the Jacobian matrix and its nearest symmetric positive semidefinite approximation. We obtain the nearest symmetric positive semidefinite matrix to the Jacobian using the method using (D'Errico).

2. Wolfe Linesearch: Newton's method alone (with a unit step length) does not guarantee convergence to the root unless the starting point is sufficiently close to the solution. The problem of nonlinear root-finding

(Nocedal & Wright, 2006, Ch.11) can be reframed in terms of minimizing a merit function which is typically the squared norm of the first-order optimality conditions (Nocedal & Wright, 2006, Eq. 11.35). To enhance robustness, we employ a Wolfe linesearch to ensure sufficient decrease of the merit function M. Thus, our use of linesearch aligns with standard practice, as discussed and explained in Nocedal & Wright (2006, Ch.3).

B.2 Gradient Descent-Ascent with Finite Differencing implementation details

We utilize JAX (Bradbury et al., 2018) to compute gradients via finite differencing of function samples.

B.3 Local Symplectic Surgery implementation details

For LSS, we utilize the same regularization and parameters described by (Mazumdar et al., 2019, Sec. 5.1). We use ForwardDiff.jl (Revels et al., 2016) for computing gradients for LSS.

B.4 Decaying polynomial implementation details

The objective function of the decaying polynomial problem is depicted in the top left plot of Fig. 1. The decision variables were $x, y \in \mathbb{R}^2$. To ensure that algorithms are initialized at points with a non-zero gradient in the decaying polynomial example (top row), we ensure these are selected from the non-flat regions of the function (a distance between 9 and 18 from the origin). The Hessian regularization constant was $\lambda = 0.01$. The strong Wolfe parameters, according to (Nocedal & Wright, 2006, Ch.3), were $c_1 = 0.01$ and $c_2 = 0.7$. We added observation noise $z_t \sim \mathcal{N}\left(0, \sigma_z^2 = 1\right)$ to each underlying objective function sample. We utilize JAX (Bradbury et al., 2018) to compute gradients for this example.

B.5 High-dimension polynomial implementation details

The objective function of the high-dimension polynomial problem is depicted in the bottom left plot of Fig. 1. The decision variables were $x, y \in \mathbb{R}^5$, resulting in a combined decision variable of 10 dimensions. The Hessian regularization constant was $\lambda = 0.01$. The strong Wolfe parameters, according to (Nocedal & Wright, 2006, Ch.3), were $c_1 = 0.01$ and $c_2 = 0.7$. We added observation noise $z_t \sim \mathcal{N}\left(0, \sigma_z^2 = 0.003\right)$ to each underlying objective function sample. We utilize JAX (Bradbury et al., 2018) to compute gradients for this example.

B.6 ARIMA tracking Model Predictive Controller (MPC)

In this experiment, an ARIMA process synthesizes a discrete time $1D$ time series of length F, denoted by $s_F \in \mathbb{R}^F$, for an MPC controller to track. Specifically, the ARIMA process generates the time series as: $s_{t+1} = \mu + \alpha s_t + \beta w_{t-1} + w_t$, where $\mu \in \mathbb{R}$ is the mean, $w = \mathcal{N}(0, \sigma) \in \mathbb{R}$ is noise, and $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}$ are model parameters. Consequently, we represent the ARIMA process for initial state s_0 and model parameters α, β: $s_F = \text{ARIMA}(s_0, \alpha, \beta)$.

The MPC controller takes the ARIMA time series, s_F, as input and returns a controller cost $f_{\text{MPC}} \in \mathbb{R}$ to track the given time series. The MPC controller solves the following optimization problem:

$$\min_{\hat{u}} \text{MPC}(A, B, \hat{s}_0, s_F) = \sum_{t=0}^{F} \left(\hat{s}_t - s_t\right)^\top Q \left(\hat{s}_t - s_t\right) + \hat{u}_t^\top R \hat{u}_t. \tag{24a}$$

$$\text{subject to:} \quad \hat{s}_t = A\hat{s}_{t-1} + B\hat{u}_{t-1} \tag{24b}$$

$$u_{\min} \leq u_t \leq u_{\max}. \tag{24c}$$

The optimization problem has quadratic costs and linear constraints. The quadratic costs in (24a), measure how well the controller tracks the timeseries s_F, and how much controller effort was used. The linear constraints in (24b) encapsulate the system dynamics, and $A, B \in \mathbb{R}$ are controller parameters that describe the dynamics. The MPC has additional control constraints in (24c), which describe the control limits of the controller. As such, we represent the optimization problem of the MPC controller as follows: $f_{\text{MPC}} = \text{MPC}(A, B, \hat{s}_0, s_F)$, which returns the final overall cost of tracking the ARIMA-generated time series s_F, and initial state, $\hat{s}_0$.

In our experiment, the decision variables were $x, y \in \mathbb{R}^2$, which resulted in a combined decision variable of four dimensions. We set the Hessian regularization constant to $\lambda = 0.001$. Following the recommendations in (Nocedal & Wright, 2006, Ch.3), we chose strong Wolfe parameters $c_1 = 0.01$ and $c_2 = 0.8$. Both the parameters of the ARIMA process (α, β) and the MPC parameters (A, B) were constrained to lie within the range $[-1, 1]$. By incorporating these constraints and parameters, we ensured a consistent framework for the optimization problem while providing sufficient flexibility for the ARIMA process and the MPC to interact in the zero-sum game. We utilized CVXPYLAYERS (Agrawal et al., 2019) for computing gradients.

B.7 Robust MPC experimental details

In this section, we provide the details of the Robust MPC experiments. In this experiment, we demonstrated that the MPC parameters found at the end of our zero-sum game between the ARIMA antagonist player and the MPC protagonist player will be more robust to out-of-distribution data. Our algorithm will converge to a local saddle point of this zero-sum game, where both players will be in equilibrium. The ARIMA antagonist player cannot find more adversarial parameters for MPC, while MPC cannot get better at tracking ARIMA generated forecasts.

Specifically, for the experiment, we compared a nominal MPC with a robust MPC (found using our method) on in-distribution data and out-of-distribution data. To generate in-distribution data, we sampled 500 ARIMA time series forecasts, s_F, with $\alpha, \beta \in [-1, 1]$. We chose out-of-distribution ARIMA parameters similar to the final ARIMA antagonist player parameters. Although the parameters were constrained to be within $[-1, 1]$ during the actual algorithm, we selected out-of-distribution parameters $\alpha = -0.1$ and $\beta = -1.2$ and sampled 500 ARIMA time series forecasts for these parameters. This choice enabled us to evaluate the robustness of the MPC against data that deviates from its original training distribution. Finally, we picked nominal MPC parameters, A, B, by fitting the MPC parameters to in-distribution data using supervised learning, i.e., the best A, B to minimize the MPC tracking cost for the in-distribution data. We chose robust MPC parameters as the final MPC protagonist player at the convergence of the zero-sum game. By comparing the performance of the nominal and robust MPCs on both in-distribution and out-of-distribution data, we aimed to demonstrate the effectiveness of our method in finding robust MPC parameters that can handle deviations from the original training data distribution better than the nominal MPC.

In Fig. 2, we compared the MPC tracking costs of both the MPCs on in-distribution data and out-of-distribution data. As expected, the nominal MPC performs better on in-distribution data since it is trained on this data. However, the robust MPC significantly outperforms the nominal MPC on out-of-distribution data by achieving a 27.6% lower mean MPC cost. Additionally, the poor performance of the nominal MPC indicates that the final ARIMA antagonist player parameters are indeed challenging. These results suggest that our algorithm has successfully identified robust MPC parameters and adversarial ARIMA parameters.

B.8 Updating Hyperparameters

Steps for updating μ_{t+1}, σ_{t+1}, UCB_{t+1}, and LCB_{t+1}:

1. Learn hyperparameters of the kernel function using maximum likelihood estimation, as explained in Section 2.3 of Rasmussen (2003).
2. Using the learned hyperparameter and updated kernel function, construct the μ_t, σ_t using the current dataset, as explained in Eq 2.25 and 2.26 in Rasmussen (2003).
3. Construct new UCB_t and LCB_t. This is straightforward since:
$$\text{UCB}_t = \mu_t + \beta * \sigma_t, \text{LCB}_t = \mu_t - \beta * \sigma_t.$$
4. Collect new samples and repeat the process.

C Ablation Studies

We generate full convergence rate results for all of our algorithm variants across the decaying polynomial, high-dimension polynomial, and ARIMA-MPC examples. For a full discussion on our variants, we refer the reader to Sec. 5.

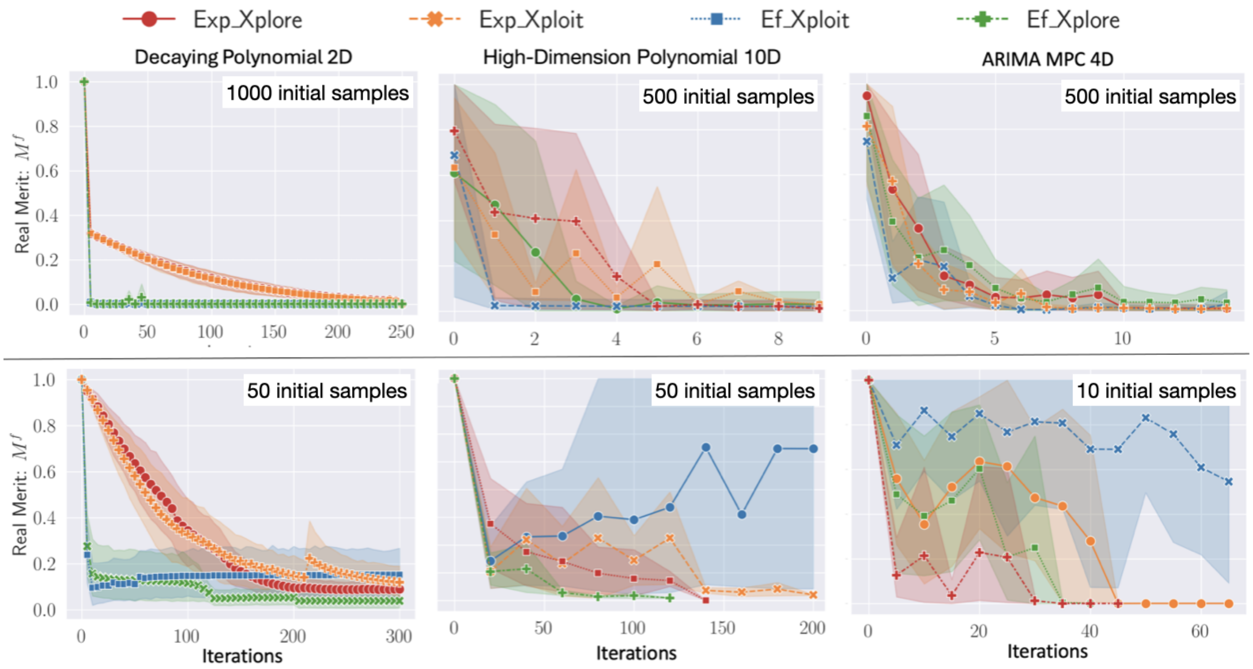

Figure 4: **Comparisons of our proposed algorithm variants:** We compare all four variants of our proposed algorithms across the three experiments (each column) described in Sec. 5. The horizontal axis denotes the number of underlying function evaluations, while the vertical axis represents the value of the real merit function, M^f. The top row considers test cases with a large number of initially sampled points, while the bottom row examines test cases with a limited number of initially sampled points. The *key takeaway* is that generally, *exploit* variants converge faster with a large number of initial samples due to effective utilization of accurate priors, while *explore* variants exhibit quicker convergence with limited samples by prioritizing exploration amid uncertainty. *Efficient* variants converge faster with many initial samples by taking multiple accurate Newton steps, while *expensive* variants show stable though often slower convergence with limited samples, taking single Newton steps to avoid unfavorable regions.

The results in Fig. 4 reveal that the *explore* variants exhibit more reliable convergence to saddle points compared to the *exploit* variants. This outcome is expected, as *explore* variants emphasize exploration and, therefore, are more likely to converge. Additionally, the *expensive* variants demonstrate greater stability in this setting, as they only take single Newton steps and are less prone to reaching unfavorable regions. The Ef-Xploit variant exhibits the lowest success rate in convergence, as it relies on exploitation and may take incorrect Newton steps.

C.1 An Example of Unsuccessful Runs using the Gradient Descent-Ascent with Finite Differencing Baseline

In Fig. 5, we provide a random sample of four runs of the gradient descent-ascent with finite differencing on the high-dimension polynomial example. This baseline never succeeds on this testbed, and these examples indicate a few of the reasons, including noisy gradient estimates that prevent converging to a point which is not a saddle, as well as leaving the feasible region and never recovering.

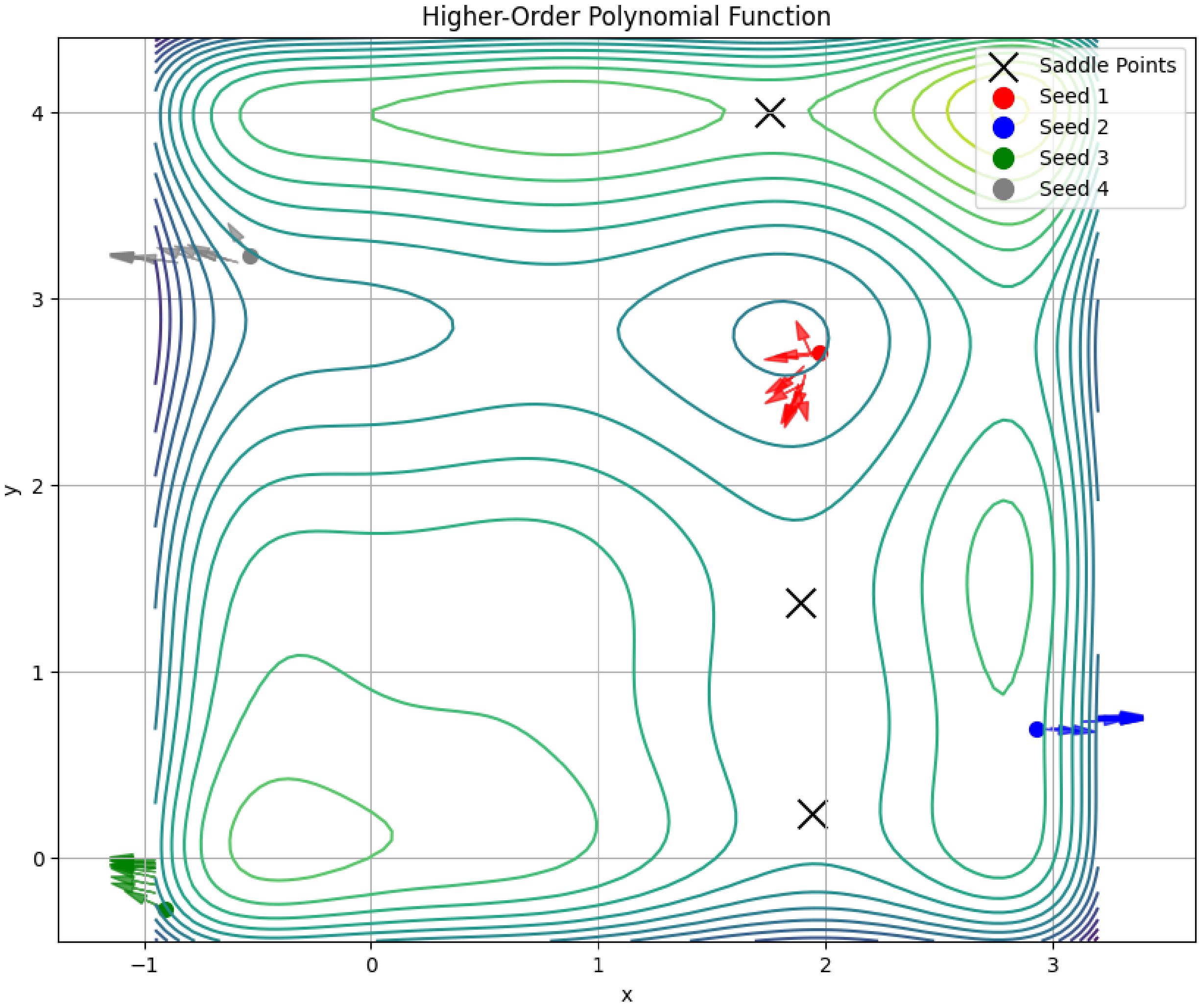

Figure 5: A random sample of four runs of the GDA with FD on the high-dimension polynomial. Three of these runs fail because they leave the feasible region and follow the gradient (of the merit function) away from it. Another fails to find a saddle point due to noise introduce by finite differencing, resulting in convergence to a point which is not a Nash point. Each arrow indicates the direction of the step.