BAYESIAN SEQUENTIAL OPTIMAL EXPERIMENTAL DESIGN FOR INVERSE PROBLEMS USING DEEP REINFORCEMENT LEARNING

A DISSERTATION SUBMITTED TO THE FACULTY OF
THE UNIVERSITY OF MINNESOTA BY

LOREN JAMES ANDERSON

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY

FADIL SANTOSA

APRIL 2022

Acknowledgements

I would like to thank my thesis adviser Dr. Fadil Santosa for his helpful discussions and insights throughout the past three years advising me on this work. I thank the other committee members Dr. Paul Schrater, Dr. Jarvis Haupt, and Dr. William Leeb for their helpful comments on this document.

I thank the University of Minnesota Twin Cities College of Science & Engineering for providing me a College of Science & Engineering Graduate Fellowship that allowed more time to focus on this work.

In similar vein, I thank the National Science Foundation for providing me a National Science Foundation Graduate Research Fellowship. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 00074041. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Abstract

We perform a comprehensive study on Bayesian sequential optimal experimental design techniques applied to inverse problems. We transform the Bayesian sequential optimal experimental design problem into a reinforcement learning problem to gauge the power of recent deep reinforcement learning algorithms compared to other baseline algorithms. Using KL-divergence as a measure of information gain, we construct objectives to maximize information gain for batch design, greedy design, black-box Bayesian optimization, multi-armed bandit optimization, dynamic programming, approximate dynamic programming, and reinforcement learning. This work showcases novel comparisons between the aforementioned methods and a new application of off-the-shelf reinforcement learning algorithms to Bayesian sequential optimal experimental design for inverse problems in differential equation models.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Experiments are ubiquitous in the world of data, including administering clinical trials [44, 25, 28], determining physical constants, [84, 83, 48, 49], and optimally placing sensors [50, 1, 2, 3, 6, 55]. Seemingly everyone is attempting to exploit data to the fullest and achieve more profit, success, or another valuable objective. Entire teams are dedicated to detect efficiencies through dissecting underlying patterns in the data that are concealed by noise and sheer volume. However, the actual process of experimentation is not always given the same weight as the analysis. Since experiments may be costly (clinical trials), noisy (sensor placements), and time-consuming (determining physical constants), exercising care in the design of such experiments can help achieve the same objectives of reducing cost, saving time, and increasing performance of a system. In this work, we apply experimental design techniques to inverse problems by attempting to recover the unknown parameters of linear and nonlinear mathematical models.

In many practical problems of interest, there exist challenges including nonlinearity, high-dimensionality, long event horizons, noisy and sparse reward signals, and complicated dynamics. Designing experiments solely based on intuition may fail to achieve optimality in these instances, possibly without knowledge of the extent to which intuition fails. We turn to optimal experimental design (OED) methods to construct designs that maximize a predefined objective [5]. Quantifying the experimental conditions and objectives can yield numerical answers to questions relating to the best design choice, the most important design features, the optimal time to conduct experiments, and

the best number of experiments, among many others. Therefore, we follow a rigorous formulation of the OED problem and devise algorithms to produce quantitative solutions to these questions.

First attempts at experimental designs include factorial designs, randomized designs, composite designs, and Latin square designs [36, 29, 18]. These designs are relatively intuitive and simple to describe which makes them enticing design paradigms for baselines and first attempts. However, these design paradigms are constructed independently of the mathematical model. For example, if the experimental goal is parameter inference or prediction, there may be some design choices that help achieve those objectives more quickly, accurately, or precisely using knowledge of the model. Optimal experimental design (OED) leverages the information of the underlying model to guide the choice of design and achieve the best objective. Since the model relates designs to observations through the unknown parameters, knowledge of the model provides more knowledge of the quantities of interest, in contrast to treating the model as a black box. We exploit the knowledge of the underlying mathematical models in this work.

The most fundamental and studied case of OED is for linear models [26]. Here, the experimental observations are related linearly to the designs through unknown parameters. Well-known design paradigms for the linear model include the alphabetical designs of A-optimality, D-optimality, E-optimality, etc. that use the data in the information matrix to achieve varying objectives [15, 18]. For example, A-optimality minimizes the trace of the inverse of the information matrix to minimize the average variance of the unknown parameters. Due to the limiting nature of the linear model, many current efforts instead aim to attack nonlinear OED problems [71, 17, 37, 27, 65, 76]. The objectives for nonlinear OED usually do not have closed-form analytic expressions, and methods such as sampling or approximation are necessary for optimizing the objective [51, 52, 65, 76, 76, 77, 84, 10, 69, 61]. We make extensive use of sampling in Chapters 3 and 4 when the objective does not have a closed-form analytic expression.

Many practical problems of interest will require data collection across multiple experiments to achieve the experimental goal (e.g. parameter inference, prediction). Two fundamental design paradigms that necessitate multiple experiments include batch design and greedy design. Batch design performs multiple experiments concurrently and must choose all designs before any feedback

is received [35, 4]. Batch design possesses the benefit of multi-step lookahead, as it chooses designs to maximize an objective that is calculated multiple experiments into the future. In contrast, greedy design forgoes multi-step lookahead and chooses a design that maximizes the immediate objective of the current experiment [84, 34, 33, 45, 54, 32, 16]. Greedy design has the benefit of feedback, as it can incorporate the feedback of past experiments to influence the choice of design for the current experiment. Since greedy design does not employ multi-step lookahead and batch design does not incorporate feedback, these two design strategies are generally suboptimal. We use these design paradigms as baselines to gauge the effectiveness of more sophisticated algorithms.

Batch design and greedy design are fundamental to multi-experiment problems, and we desire to employ a paradigm that exploits the strengths of batch and greedy design while preventing the weaknesses; namely, we turn to sequential and OED formulations. A sequential formulation of the experimental design problem permits experiments to be performed consecutively and allows use of past feedback at the current experiment. An OED formulation uses the knowledge of the underlying model to incorporate multi-step lookahead through backing up knowledge of the objective from future experiments to the current experiment. We combine a sequential formulation with an OED formulation to natively incorporate feedback and multi-step lookahead, known as sequential optimal experimental design (sOED) [51, 52, 79, 25, 38, 20]. Algorithms employing sOED incorporate multi-step lookahead through using knowledge of all future experiments and the objective while designing the current experiment. They also incorporate feedback through using the information from past experiments to guide the choice of design at the current experiment. Usually an update rule is used to back up information of the objective from future experiments to the current experiment. Considering that sOED methods incorporate both multi-step lookahead and also feedback, these methods are usually much more computationally demanding. A central theme of this work is the investigation of tradeoff between optimality and computational complexity.

The sOED formulation defines our design paradigm; now we must choose an objective. Our goal is to infer the unknown parameters of a mathematical model, including a linear model (Chapter 2), and linear and nonlinear differential equations (Chapters 3 and 4). We choose to adopt a Bayesian perspective on our goal [27] where instead of providing a point estimate of the parameters

of interest, we provide a measurement of confidence on our unknown parameters through a probability distribution. A Bayesian viewpoint works naturally with sOED, as the current belief state on the parameters of interest is sequentially updated through Bayes' Theorem. With a Bayesian viewpoint, we further specify that our goal is to gain the most information about our unknown parameters after all experimentation is complete. Shannon [78] mathematically defined information on finite spaces. Kullback and Leibler extended the notion of information to more general functions [56] and developed the KL-divergence function that quantifies the information gain from a prior to posterior probability distribution. Lindley [59] proposed to use expected KL-divergence as an objective function to quantify expected information gain [12, 51, 52, 79]. We follow the decision-theoretic approach to experimental design given by Lindley to form an objective that quantifies expected information gain in the unknown parameters through KL-divergence [60].

Bayesian techniques for sOED are scarce [52, 79] due to the computational hurdles: the Bayesian sOED problem is noisy, requires lookahead, has costly dynamics, is possibly nonlinear with high-dimensionality, among other possible challenges. A full description of the Bayesian sOED problem is provided by Shen and Huan [79]. Instead of using previous methods that are intended for use solely on the Bayesian sOED problem, we instead transform the Bayesian sOED problem into a reinforcement learning (RL) problem [82] and use modern deep reinforcement learning (DRL) techniques to maximize the objective. Central to RL is the Markov Decision Process (MDP). In a MDP, there are states, actions to be taken at a state, rewards for achieving objectives, and underlying dynamics to guide the next states and rewards given the current states and actions. RL methods attempt to solve the MDP by forming a policy that provably obtains the most expected sum of rewards from any state, known as an optimal policy. The Bayesian sOED problem can be transformed to a MDP by converting belief states to states, designs to actions, the objective to rewards, and the underlying model to dynamics. Solving the MDP for the optimal policy is equivalent to maximizing the Bayesian sOED objective.

The only work on applying DRL methods to the Bayesian sOED problem for inverse problems that we could find in the literature is that by Shen and Huan [79]. Our work showcases three high-level novelties to the existing literature. The first novelty is the application domain of DRL. Our

application of DRL is to inverse problems for parameter estimation in differential equation models by choosing the times at which to perform experiments and measure observations. The second novelty is the implementation framework of DRL algorithms applied to Bayesian sOED. The algorithm posed by Shen and Huan [79] as stated does not follow the traditional agent-environment interaction standard that was popularized by the OpenAI Gym interface [19]. Due to the notorious difficulty of implementing DRL algorithms, we aim to study DRL algorithms that can be implemented through off-the-shelf libraries such as RL Lib [58] and OpenAI Stable Baselines [47]. Finally, the third novelty is the extensive comparison among Bayesian sOED methods. We compare and contrast the optimality and computational complexity of batch design, greedy design, black-box Bayesian optimization, multi-armed bandit optimization, dynamic programming, approximate dynamic programming, and DRL methods to the Bayesian sOED problem.

1.1 Problem Statement

Current advancements in deep reinforcement learning (DRL) techniques have produced superhuman performance on a bevy of tasks, most notably popular video games [67, 88] and board games [80, 85]. In each situation, the learned strategies were novel and transcended heuristic arguments and human intuition. Since (i) the Bayesian sOED problem can easily be converted to a RL problem, (ii) the literature on applying RL to Bayesian sOED is extremely sparse, and (iii) DRL algorithms are notoriously difficult to implement, we investigate the application of off-the-shelf DRL methods to the Bayesian sOED problem on an assortment of linear and nonlinear mathematical models. The investigation includes a comparison of DRL algorithms with other fundamental Bayesian sOED algorithms including batch design, greedy design, black-box Bayesian optimization, multi-armed bandit optimization, dynamic programming, and approximate dynamic programming. Emphasis is placed on discussing optimality and computational complexity of the algorithms. This work showcases novel comparisons between the aforementioned methods and also new applications of off-the-shelf DRL algorithms to Bayesian sequential optimal experimental design for inverse problems in differential equation models.

1.2 Problem Formulation

The goal of inverse problems is to determine the underlying parameters of a mathematical model. Applying experimental design to inverse problems aims to determine these underlying parameters through designing experiments and taking observations through using the known relationship between designs and observations in the model. We now produce mathematical notations for this process.

We assume a finite number of experiments indexed by $k \in \{0,1,2,\ldots,N-1\}$ for a total of N experiments. Each experiment is endowed with a design space $\mathscr{D}_k \subseteq \mathbb{R}^{M_d}$ that dictates the allowed designs at experiment k. The parameters of the model $\boldsymbol{\theta}$ may be multidimensional. Denoting the parameter space as Θ, we assume that $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^{M_{\boldsymbol{\theta}}}$. Given parameters $\boldsymbol{\theta}$ and a design d, our model is used to predict an output which we measure. However, there is possible measurement and modeling error. Therefore, the measurement $y \in \mathscr{Y} \subseteq \mathbb{R}^{M_y}$ is given by the relationship

$$y = G(\boldsymbol{\theta}, d) + \varepsilon, \tag{1.1}$$

where $G(\boldsymbol{\theta}, d)$ is the forward map, representing the model, and ε is measurement error. Conducting N experiments yields designs $\{d_k\}_{k=0}^{N-1}$ and corresponding measurements $\{y_k\}_{k=0}^{N-1}$. The inverse problem is to solve for $\boldsymbol{\theta}$ given the design and observation data. In most of the existing literature, it is assumed that the designs are fixed quantities. The subject of experimental design is to choose the experiments that maximize a given goal.

For inverse problems, our goal is to infer the unknown parameters. As opposed to providing point estimates of our unknown parameters, we instead aim to provide a measurement of confidence over the unknown parameters. As such, we adopt a Bayesian viewpoint on our experimental design problem and express the confidence over the unknown parameters with a probability distribution. With this viewpoint, we further specify our goal to be the most information gain on $\boldsymbol{\theta}$ after completion of all experiments. There are many interpretations of information content, but here we will follow the approach by Shannon [78], Kullback and Leibler [56], and Lindley [59, 60].

To perform Bayesian experimental design, we first need to discuss sequential design, as the

confidence distribution will be updated sequentially by Bayes' Rule. In sequential experimental design, we view each experiment separately. We start with experiment d_0, collect data y_0, and then conduct experiment d_1 and measure y_1, and so on. The sole control of the experimenter over the outcome of the experiment is the experimental design d_k. Various conditions may be imposed on the design space relating to the problem at hand. For example, we sometimes restrict $d_{i+1} > d_i$ when the experimental design is time at which a measurement is sampled. The information set $I_k = \{d_0, y_0, d_1, y_1, \ldots, d_{k-1}, y_{k-1}\}$ contains all designs and past observations and is recursively defined as

$$I_0 = \emptyset, \qquad I_{k+1} = I_k \cup \{d_k, y_k\}.$$

In Bayesian analysis, we need to compute a posterior belief on $\boldsymbol{\theta}$ given the current design and observation. We incorporate Bayes' Theorem as follows:

$$p(\boldsymbol{\theta}|d_k, y_k, I_k) = \frac{p(y_k|\boldsymbol{\theta}, d_k, I_k)p(\boldsymbol{\theta}|I_k)}{p(y_k|d_k, I_k)} \tag{1.2}$$

The prior belief on the current parameters, $p(\boldsymbol{\theta}|I_k)$, quantifies the certainty of the actual parameters across the parameter space Θ. We simplified $p(\boldsymbol{\theta}|d_k, I_k)$ to $p(\boldsymbol{\theta}|I_k)$ in the above equation since knowing the design of the current experiment should not influence the prior. The likelihood term $p(y_k|\boldsymbol{\theta}, d_k, I_k)$ quantifies the likelihood of the observations given a specified value of the parameters, the design, and the past observations and designs. The evidence term $p(y_k|d_k, I_k)$ quantifies the likelihood of the observation across the current parameter priors given the current design and past observations. Finally, the posterior term $p(\boldsymbol{\theta}|d_k, y_k, I_k)$ quantifies the certainty of the parameters after choosing a design and receiving the observation. In this manner, we can sequentially update the current belief on the parameters by noting that $p(\boldsymbol{\theta}|d_k, y_k, I_k) = p(\boldsymbol{\theta}|I_{k+1})$ and applying the above formula again as needed.

Shannon [78] defined information for finite sets. Kullback and Leibler extended information for more general functions [56] and constructed a function coined Kullback-Leibler divergence (KL-divergence) that quantifies the information gain from prior distribution $p(\cdot|I_k)$ to posterior dis-

tribution $p(\cdot|I_{k+1})$, namely $D_{KL}(p(\cdot|I_{k+1})||p(\cdot|I_k))$. In terms of experimental design, we have

$$D_{KL}(p(\cdot|\mathbf{y},\mathbf{d})||p(\cdot)) = f(\mathbf{d},\mathbf{y}) = \int_\Theta p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d}) \ln\left[\frac{p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})}{p(\boldsymbol{\theta})}\right] d\boldsymbol{\theta}, \tag{1.3}$$

where $\mathbf{d}$ and $\mathbf{y}$ are vectors of designs and corresponding observations from separate experiments. Lindley suggests to use KL-divergence as a utility function in experimental design to quantify expected information gain as an objective [59]. Lindley's decision-theoretic framework for experimental design considers a following general objective:

$$U(\mathbf{d}) = \int_{\mathscr{Y}} \int_\Theta u(\mathbf{d},\mathbf{y},\boldsymbol{\theta}) p(\boldsymbol{\theta},\mathbf{y}|\mathbf{d})\, d\boldsymbol{\theta}\, d\mathbf{y}. \tag{1.4}$$

Here, $u(\mathbf{d},\mathbf{y},\boldsymbol{\theta})$ is a utility function that specifies the worth of receiving observation $\mathbf{y}$ given design $\mathbf{d}$ and parameters $\boldsymbol{\theta}$ in relation to the overall experimental goal. Substituting KL-divergence for the utility function yields the following:

$$\begin{aligned}
U(\mathbf{d}) &= \int_{\mathscr{Y}} u(\mathbf{d},\mathbf{y},\boldsymbol{\theta}) p(\boldsymbol{\theta},\mathbf{y}|\mathbf{d})\, d\boldsymbol{\theta}\, d\mathbf{y}, \\
&= \int_{\mathscr{Y}} \int_\Theta u(\mathbf{d},\mathbf{y},\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d}) d\boldsymbol{\theta} p(\mathbf{y}|\mathbf{d}) d\mathbf{y}, \\
&= \int_{\mathscr{Y}} u(\mathbf{d},\mathbf{y}) p(\mathbf{y}|\mathbf{d}) d\mathbf{y}, \\
&= \int_{\mathscr{Y}} \int_\Theta p(\tilde{\boldsymbol{\theta}}|\mathbf{y},\mathbf{d}) \ln\left[\frac{p(\tilde{\boldsymbol{\theta}}|\mathbf{y},\mathbf{d})}{p(\tilde{\boldsymbol{\theta}})}\right] d\tilde{\boldsymbol{\theta}} p(\mathbf{y}|\mathbf{d}) d\mathbf{y}, \\
&= \int_{\mathscr{Y}} \int_\Theta p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d}) \ln\left[\frac{p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})}{p(\boldsymbol{\theta})}\right] d\boldsymbol{\theta} p(\mathbf{y}|\mathbf{d}) d\mathbf{y}, \\
&= \mathbb{E}_{\mathbf{y}|\mathbf{d}}[D_{KL}(p(\cdot|\mathbf{y},\mathbf{d})||p(\cdot))].
\end{aligned} \tag{1.5}$$

Therefore, the objective is expected KL-divergence when the utility function is KL-divergence. In other words, the objective is expected information gain when our utility function is information gain.

The above completes the description of our Bayesian sOED problem. The above equation is the objective for batch design when taking the maximum over all possible $\mathbf{d} \in \mathscr{D}^N$. In Chapter 2, the objective has a closed-form analytic solution, while in Chapters 3 and 4, we have to sample

the double integrals to compute the objective. Other design paradigms will transform the above objective. In Section 2.3, we show equivalence between solving a Markov Decision Process (MDP) for an optimal policy and maximizing an objective, allowing us to solve the objective with deep reinforcement learning (DRL) techniques. The above objective provides a mathematically rigorous and highly developed starting point to all other design paradigms that we consider.

1.3 Outline

The above sections provide the foundation for this work. We have explained our motivation for pursuing this project along with a statement of the problem in Section 1.1. The problem formulation in Section 1.2 provides the necessary notation and concepts to understand the upcoming algorithms. We choose to study batch design, greedy design, multi-armed bandits, black-box Bayesian optimization, dynamic programming, approximate dynamic programming, and deep reinforcement learning. The choice of algorithms was motivated by their fundamental nature and use as comparisons in other Bayesian sOED problems [79, 52]. The implementations of these algorithms are fairly primitive, as we are more concerned about comparing the algorithms rather than producing state-of-the-art results. Further, the comparison of a diverse and primitive set of algorithms provides a useful baseline for future work.

Chapter 2 considers the Bayesian sOED problem applied to a linear model with two unknown parameters. We allow $N = 10$ experiments with designs from $\mathscr{D} = \{0.1, 0.2, 0.3, \ldots, 1.0\}$. The objective has a closed-form analytic solution, which enables quick computation and allows us to maximize the objective through exhausting all possible design sequences. We compare batch design, greedy design, dynamic programming, approximate dynamic programming, and reinforcement learning algorithms. Since batch design is optimal in this case, we are able to determine the optimality of the other algorithms by comparison. Finally, we summarize by discussing the strengths and weaknesses of each algorithm.

Chapter 3 considers the Bayesian sOED problem applied to two distinct ordinary differential equations. In this chapter, we assume that the state of each ODE can be measured as well as the velocity. We compare the two distinct ODE because one of the ODEs is linear in the unknown

parameters and could be solved using techniques from Chapter 2 since velocity data is available. There are $N = 2$ experiments allowed with designs from the discrete set $\mathscr{D} = \{0, 1, 2, \ldots, 9\}$. We apply batch design, greedy design, black-box Bayesian optimization, multi-armed bandits, approximate dynamic programming, and deep reinforcement learning. The objective function does not yield a closed-form analytic expression, and we must resort to extensive Monte Carlo sampling techniques for computation. Due to the difficult computation, we increase emphasis on discussing the computational complexity of each algorithm.

Chapter 4 analyzes the same problem and algorithms as Chapter 3 while changing the observations: we only allow state observations as a more practical assumption. We compare results across ODE1 and ODE2 as well as results across Chapters 3 and 4. The algorithms are described in detail in Chapter 3, so we refrain from duplicating the same material in Chapter 4 and primarily present results. As we compare Chapter 3 and Chapter 4, we note that the allowed data in Chapter 4 is essentially halved per experiment, and we analyze the corresponding decrease in information gain.

We provide a final comparison across all chapters, showcase the novelty of our work, and discuss future directions in Chapter 5. We highlight how these methods can be applied to parameter inference for the well-known Lotka-Volterra differential equations [62].

Chapter 2

OED in Linear Regression

Our first application of Bayesian sequential optimal experimental design (Bayesian sOED) for inverse problems is to linear regression in two variables. We assume the parameters $\boldsymbol{\theta} = (\theta_1, \theta_2)$ are drawn from a normal distribution with zero mean and covariance equal to the identity matrix: $\mathcal{N}(\boldsymbol{\mu} = \mathbf{0}, \Sigma = \mathbb{I}_2)$. The designs are chosen from the discrete set $\mathscr{D} = \{0.1, 0.2, 0.3, \ldots, 1.0\}$. We allow $N = 10$ experiments. The forward model is

$$G_k(\boldsymbol{\theta}, d_k) = \theta_1 + \theta_2 \cdot d_k. \tag{2.1}$$

We anticipate that for most choices of d_k and $\boldsymbol{\theta}$ that $G_k(\boldsymbol{\theta}, d_k) \subseteq (-5, 5)$. Therefore, we set $\varepsilon \sim \mathcal{N}(0, 1)$, with relation

$$y_k = \theta_1 + \theta_2 \cdot d_k + \varepsilon = \begin{pmatrix} 1 & d_k \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} + \varepsilon. \tag{2.2}$$

With these normal prior distributions on $\boldsymbol{\theta}$ and ε, the posteriors also are also normally distributed. Performing a Bayesian update with prior distribution $\mathcal{N}(\boldsymbol{\mu}_{prior}, \Sigma_{prior})$, designs $\mathbf{d}$, and observations $\mathbf{y}$ with n experiments yields posterior $\mathcal{N}(\boldsymbol{\mu} = A^{-1}b, \Sigma = A^{-1})$ where

$$A = \Sigma_{prior}^{-1} + \sum_{k=0}^{n-1} \begin{pmatrix} 1 & d_k \\ d_k & d_k^2 \end{pmatrix} \quad \text{and} \quad b = \Sigma_{prior}^{-1} \boldsymbol{\mu}_{prior} + \sum_{k=0}^{n-1} \begin{pmatrix} y_k \\ y_k d_k \end{pmatrix}. \tag{2.3}$$

Complete details of this derivation are located in the appendix Section 7.1 in a more general form. Surprisingly, the covariance matrix is independent of y_k, which will vastly simplify the computation of the objective. The KL-divergence between a prior $\mathcal{N}(\boldsymbol{\mu}_{prior}, \Sigma_{prior})$ and posterior $\mathcal{N}(\boldsymbol{\mu}_{post}, \Sigma_{post})$ is computed in appendix Section 7.2 as follows:

$$D_{KL}(\mathcal{N}(\boldsymbol{\mu}_{post}, \Sigma_{post}) || \mathcal{N}(\boldsymbol{\mu}_{prior}, \Sigma_{prior})) = \frac{1}{2}[\log(\det(\Sigma_{prior})) - \log(\det(\Sigma_{post}))]. \tag{2.4}$$

Therefore, the KL-divergence is independent of y_k and ε as well. Noting that $\boldsymbol{\Sigma}_{prior}$ is constant, we can now construct the objective function solely dependent on a design sequence of length n; i.e., $|\mathbf{d}| = n$. Using (1.5), we see the following:

$$\begin{aligned} U(\mathbf{d}) &= \mathbb{E}_{\mathbf{y}|\mathbf{d}}[D_{KL}(p(\cdot|\mathbf{y},\mathbf{d})||p(\cdot))] && (2.5)\\ &= \mathbb{E}_{\mathbf{y}|\mathbf{d}}\left[\frac{1}{2}[\log(\det(\Sigma_{prior})) - \log(\det(\Sigma_{prior}^{-1} + \sum_{k=0}^{n-1}\begin{pmatrix} 1 & d_k \\ d_k & d_k^2 \end{pmatrix}))]\right] && (2.6)\\ &= \frac{1}{2}[\log(\det(\Sigma_{prior})) - \log(\det(\Sigma_{prior}^{-1} + \sum_{k=0}^{n-1}\begin{pmatrix} 1 & d_k \\ d_k & d_k^2 \end{pmatrix}))]. && (2.7) \end{aligned}$$

Therefore, in the case of OED for linear regression, the objective has a closed-form analytic expression. This computation was also performed by Chaloner and Verdinelli [27]. An analytic expression for the objective trivializes the computation; in other chapters when the KL-divergence is dependent on the observation, methods such as Monte Carlo sampling are required.

In what follows, we apply batch design, greedy design, approximate dynamic programming, dynamic programming, and reinforcement learning to solve the Bayesian sOED problem. Each algorithm attempts to optimize the objective in different ways; some incorporate feedback, while others allow for lookahead. This current chapter is primarily meant to provide a discussion of the algorithms, as the existence of an analytic expression trivializes the computation and allows for enumeration of all possible $\mathbf{d}$ with exact values for the objective. Throughout, we compare and contrast the various algorithms and optimality criteria.

2.1 Batch Design

The goal of batch design is to maximize our objective while performing a set of experiments concurrently [35, 4]. In this manner, batch design lacks feedback from experiments and incorporates only multi-step lookahead. With $|\mathbf{d}| = N = 10$, we have the following optimization problem:

$$\mathbf{d}^* = \underset{\mathbf{d} \in \mathscr{D}^{10}}{\operatorname{argmax}} U(\mathbf{d}). \tag{2.8}$$

Since we can compute $U(\mathbf{d})$ quickly, our batch algorithm is to list the possible $\mathbf{d}$, compute their corresponding objective scores, and then determine the best such $\mathbf{d}$, namely $\mathbf{d}^*$. Since successive Bayesian updates are invariant to the order of the update, we only need to consider sorted $\mathbf{d}$; i.e. $d_0 < d_1 < d_2 < \ldots < d_9$. Using sorted $\mathbf{d}$ reduces the number of possible $\mathbf{d}$ from 10^{10} to $92378 \approx 10^5$. Since the objective is only dependent on $\mathbf{d}$, we do not need to simulate any experiments to yield observations and only need to perform a single batch update with 10 arbitrary observations. Since updates are deterministic, we must only compute the KL-divergence from a single batch update for each $\mathbf{d}$, as (2.6) and (2.7) show. We are analyzing the objective function of each possible $\mathbf{d}$, so this section serves as our Exploratory Data Analysis (EDA). The batch design pseudocode is displayed in Algorithm 1.

Algorithm 1: Batch Design in Linear Regression

Input : All $\mathbf{d}$ of length 10
Initialize empty *list* of (objective, $\mathbf{d}$) pairs
for *each* $\mathbf{d}$ **do**
    Perform batch update (2.3)
    Compute objective using KL-divergence (2.4)
    Append (objective, $\mathbf{d}$) pair to *list*
end
Sort *list* by decreasing objective
Output: *list* of sorted (objective, $\mathbf{d}$) pairs by decreasing objective

2.1.1 Results

Table 2.1 displays sample sorted $\mathbf{d}$ along with their objective scores and associated rankings from 0 (best) to 92377 (worst) by decreasing objective. The best $\mathbf{d}$ appear to consist of designs on the boundary of the design space $\mathscr{D}$. The optimal $\mathbf{d}^*$ is non-symmetric and consists of four 0.1 designs and six 1.0 designs. Also, repeatedly choosing the smallest design (0.1) yields the worst objective.

Rank	$\mathbf{d}$	$U(\mathbf{d})$
0	(0.1, 0.1, 0.1, 0.1, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0)	1.798
1	(0.1, 0.1, 0.1, 0.1, 0.1, 1.0, 1.0, 1.0, 1.0, 1.0)	1.796
2	(0.1, 0.1, 0.1, 0.1, 0.9, 1.0, 1.0, 1.0, 1.0, 1.0)	1.787
3	(0.1, 0.1, 0.1, 0.2, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0)	1.785
4	(0.1, 0.1, 0.1, 0.1, 0.2, 1.0, 1.0, 1.0, 1.0, 1.0)	1.785
5	(0.1, 0.1, 0.1, 0.1, 0.1, 0.9, 1.0, 1.0, 1.0, 1.0)	1.782
10	(0.1, 0.1, 0.1, 0.3, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0)	1.774
100	(0.1, 0.1, 0.1, 0.1, 0.2, 0.3, 1.0, 1.0, 1.0, 1.0)	1.746
200	(0.1, 0.1, 0.1, 0.1, 0.7, 0.8, 0.9, 1.0, 1.0, 1.0)	1.735
500	(0.1, 0.1, 0.4, 0.4, 0.9, 1.0, 1.0, 1.0, 1.0, 1.0)	1.718
1000	(0.1, 0.1, 0.1, 0.4, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0)	1.702
2000	(0.1, 0.1, 0.1, 0.7, 0.7, 0.9, 0.9, 0.9, 0.9, 1.0)	1.685
5000	(0.1, 0.1, 0.1, 0.2, 0.5, 0.8, 0.8, 0.8, 0.9, 1.0)	1.656
10000	(0.2, 0.2, 0.4, 0.4, 0.4, 0.4, 0.9, 1.0, 1.0, 1.0)	1.629
20000	(0.1, 0.2, 0.3, 0.3, 0.6, 0.7, 0.7, 0.7, 1.0, 1.0)	1.595
50000	(0.1, 0.4, 0.4, 0.4, 0.4, 0.6, 0.6, 0.7, 0.9, 1.0)	1.525
92377	(0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1)	1.203

Table 2.1: Sample of sorted $\mathbf{d}$ and respective rankings and objective scores.

The distribution of rewards is displayed in Figure 2.1. The values are nearly symmetric about the mean with a slightly longer left tail. Note that this distribution is based on sorted $\mathbf{d}$ and does not reflect a set of random $\mathbf{d}$. We now detail the summary statistics of the distribution in Table 2.2.

Mean	Median	Min	Max
1.530	1.533	1.203	1.798

Table 2.2: Summary statistics of sorted $\mathbf{d}$ where $|\mathbf{d}| = 10$.

The mean and median are virtually identical, as evidenced by the near-symmetry of the distribution. The range of possible objective scores is slightly less than 0.6. Intrigued by the best and worst $\mathbf{d}$ only containing designs of 0.1 and 1.0, we investigate all sorted $\mathbf{d}$ where $|\mathbf{d}| = 10$ containing only

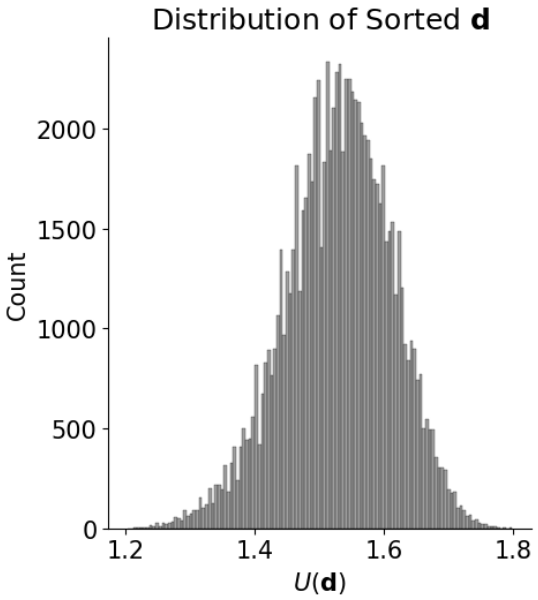

Figure 2.1: Histogram of $U(\mathbf{d})$ for sorted $\mathbf{d}$ where $|\mathbf{d}| = 10$.

designs in $\mathscr{D} = \{0.1, 1.0\}$ in Table 2.3.

$\mathbf{d}$	$U(\mathbf{d})$
(0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1)	1.203
(0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 1.0)	1.482
(0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 1.0, 1.0)	1.630
(0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 1.0, 1.0, 1.0)	1.718
(0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 1.0, 1.0, 1.0, 1.0)	1.770
(0.1, 0.1, 0.1, 0.1, 0.1, 1.0, 1.0, 1.0, 1.0, 1.0)	1.796
(0.1, 0.1, 0.1, 0.1, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0)	1.798
(0.1, 0.1, 0.1, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0)	1.778
(0.1, 0.1, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0)	1.733
(0.1, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0)	1.653
(1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0)	1.522

Table 2.3: Objective scores of $\mathbf{d}$ where $|\mathbf{d}| = 10$ containing only designs 0.1 and 1.0.

There is a single mode of $U(\mathbf{d})$. Namely, choosing four 0.1 designs and six 1.0 designs is optimal, and the objective decreases monotonically when any number of 0.1 designs are replaced with 1.0 or vice versa. Table 2.3 also suggests that having designs of 1.0 is beneficial in general, as a $\mathbf{d}$ consisting entirely of 1.0 designs is only slightly worse than the mean.

For the sake of completeness, we display the $\mathbf{d}^*$ of lengths 1 to 10 inclusive in Table 2.4. No-

$\lvert\mathbf{d}\rvert$	$\mathbf{d}^*$	$U(\mathbf{d})$
1	(1.0)	0.549
2	(1.0, 1.0)	0.805
3	(0.1, 1.0, 1.0)	1.016
4	(0.1, 1.0, 1.0, 1.0)	1.173
5	(0.1, 0.1, 1.0, 1.0, 1.0)	1.315
6	(0.1, 0.1, 1.0, 1.0, 1.0, 1.0)	1.431
7	(0.1, 0.1, 0.1, 1.0, 1.0, 1.0, 1.0)	1.540
8	(0.1, 0.1, 0.1, 1.0, 1.0, 1.0, 1.0, 1.0)	1.632
9	(0.1, 0.1, 0.1, 0.1, 1.0, 1.0, 1.0, 1.0, 1.0)	1.721
10	(0.1, 0.1, 0.1, 0.1, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0)	1.798

Table 2.4: All $\mathbf{d}^*$ of lengths 1 to 10 inclusive.

tably, the first two $\mathbf{d}^*$ only contain the design 1.0. Even though the system is underdetermined with one design, the objective is still positive since information is gained about the more probable regions of Θ in which $\boldsymbol{\theta}$ exist. Afterwards, the $\mathbf{d}^*$ contain both 0.1 and 1.0 designs. The objective steadily increases as $|\mathbf{d}|$ increases, but the rate of increase is negative; we have not yet investigated if the objective is bounded for variable length $\mathbf{d}$.

2.1.2 Summary

These results suggest that the interaction between elements of the design space is non-trivial and transcends basic heuristic arguments. For instance, the best $\mathbf{d}$ was asymmetric, containing four 0.1 designs and six 1.0 designs. Also, choosing the seemingly best design of 1.0 for each experiment through a myopic point of view would yield a mediocre $\mathbf{d}$ at best. It appears that only choosing elements at the boundaries of the design space is a good heuristic to receive the most reward. This remained true when running a batch design study on variable length $\mathbf{d}$ from 1 to 10 inclusive. It remains to be seen if this trend holds true for larger $|\mathbf{d}|$ or for multi-dimensional designs.

Batch design sufficed for our case of the linear OED problem, as we were able to exhaustively search $\mathscr{D}^{10}$ in under one minute of computation with single-core processing. This was considerably aided by closed-form equations for the objective. Furthermore, we did not need to sample values of the objective since Bayesian updates were deterministic and the objective was ultimately independent of $\mathbf{y}$ and ε.

If any step of the study becomes more difficult, then batch design can fail miserably. For example, Bayesian updates or the objective may need to be computed through a more intensive method (e.g. Monte Carlo). Each $\mathbf{d}$ may need to be tested multiple times to smooth out stochasticity in the observations if they influence the reward function. The algorithm as posed grows exponentially in $|d|$ and needs to be adapted for the case when $|d| > 1$ or for continuous state spaces. To resolve these issues in future algorithms, we will need to turn to function approximation and iterative methods.

2.2 Greedy Design

Batch design considered the Bayesian OED problem; in the greedy design paradigm [84, 34, 33, 45, 54, 32, 16], we generalize to a Bayesian sequential OED (sOED) problem in which the information $I_k = \{d_0, y_0, \ldots, d_{k-1}, y_{k-1}\}$ from past experiments 0 through $k-1$ may be used to determine design d_k for experiment k. Batch design is a special case of Bayesian sOED since it neglects the information I_k of past experiments. In greedy design, the design choice at each stage is myopic: we successively choose the design that maximizes the objective after a single experiment using the latest posterior belief state as the new prior. For experiment k, we have the following objective:

$$d_k^* = \underset{d_k \in \mathscr{D}}{\operatorname{argmax}}\, U(d_k), \qquad \boldsymbol{\theta} \sim \mathscr{N}(\boldsymbol{\mu}_k, \Sigma_k). \tag{2.9}$$

The explicit optimization problem is as follows:

$$d_k^* = \underset{d_k \in \mathscr{D}}{\operatorname{argmax}}\, \frac{1}{2}\left[\log(\det(\Sigma_k)) - \log\left(\det\left(\Sigma_k^{-1} + \begin{pmatrix} 1 & d_k \\ d_k & d_k^2 \end{pmatrix}\right)\right)\right]. \tag{2.10}$$

Once a design d_k has been chosen and observation y_k received, the posterior $\mathscr{N}(\boldsymbol{\mu}_k, \Sigma_k)$ is computed through Bayes' Theorem and the entire process repeats with a new objective for d_{k+1}.

Greedy design has an advantage over batch design in that it uses the feedback of the information set $I_k = \{d_0, y_0, \ldots, d_{k-1}, y_{k-1}\}$ to determine the design d_k. For example, a Bayesian update can be used to find the posterior belief state after the experiment, and the posterior belief state can provide samples of the prior for the next experiment. Since Bayesian updates are deterministic and the

objective only depends on the design choice in the linear case, we do not see this advantage here. Another benefit of a greedy approach is that updates can continue until a predetermined threshold of utility is reached. A clear disadvantage is that one-step lookahead is myopic and may produce a suboptimal $\mathbf{d}$ that can never be remedied through future design choices. The pseudocode is shown in Algorithm 2.

Algorithm 2: Greedy Design in Linear Regression

Input : Initial belief state $\mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0)$
Design space $\mathscr{D}$,
Objective threshold or $|\mathbf{d}|$ threshold

```
Initialize list of best designs
while threshold not reached do
    Initialize best d to 0
    Initialize best objective to 0
    for each d in 𝒟 do
        Compute objective (2.10)
        if objective > best objective then
            Set best d to d
            Set best objective to objective
        end
    end
    Append best d to list of best designs
    Compute posterior belief state using best d and current belief state (2.3)
end
```

Output: Greedy $\mathbf{d}^*$

2.2.1 Results

The output of greedy design is $\mathbf{d}^* = (1.0, 1.0, 0.1, 1.0, 0.1, 1.0, 0.1, 1.0, 0.1, 1.0)$. Coincidentally, this is the same $\mathbf{d}^*$ found with batch design. Note that the first k designs of the above $\mathbf{d}^*$ form design subsequences $\mathbf{d}_k^* = (d_0, d_1, \ldots, d_{k-1})$. Comparing these subsequences to optimal batch designs of variable length in Table 2.4, we see that $\mathbf{d}_k^*$ is equivalent to the optimal batch design $\mathbf{d}^*$ for $|\mathbf{d}| \in \{1, 2, \ldots, 10\}$. Therefore, greedy design is identical to batch design for design sequences up to length 10 for our problem. We are unsure if this fact holds true for arbitrary $|\mathbf{d}|$.

Figure 2.2 shows the greedy objective for each experiment assuming that past designs were chosen greedily. We see that the design space boundaries (top and bottom row) are usually the best

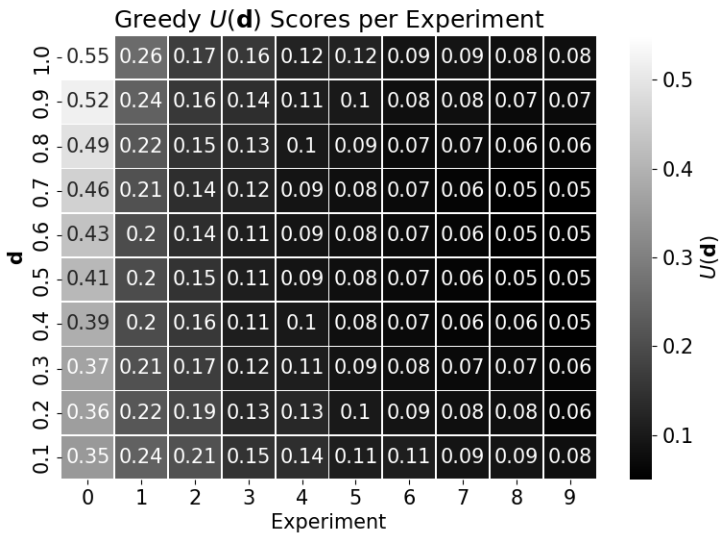

Figure 2.2: Heatmap of greedy design. Square (d,k) shows $U(d_k)$ assuming $(d_0,d_1,\ldots,d_{k-1}) = \mathbf{d}^*_{k-1}$.

choices. Also, the middle rows are generally the worst. Designs 0.1 and 1.0 are the best two design choices for the last five experiments. These facts provide more evidence that the designs 0.1 and 1.0 are useful simply because they are on the boundary.

2.2.2 Summary

The $\mathbf{d}^*$ returned by greedy design is identical to that of batch design for $|\mathbf{d}|$ up to length 10. This suggests that a myopic point of view may be optimal for this case of the linear problem, although we cannot presently provide a proof or even a heuristic. Nonetheless, it may be the case that the myopic point of view is at worst nearly optimal for this problem. If the optimal design sequence is only composed of designs 0.1 and 1.0, which has overwhelming evidence from both the batch and now greedy cases, this idea has promise. Another point of note is that designs in the middle of the design space were continually the worst choices, with reward usually growing steadily from the middle rows to the boundary rows in Figure 2.2. It is unclear if it is just a feature of one-dimensional design or the linear problem in general.

If the myopic point of view is optimal, computational complexity is considerably reduced. Greedy design requires only a single pass through each experiment. Each design must be tested

for each experiment; in our case of deterministic updates and an objective depending only on design sequence, we only need to run a number of experiments equal to the number of designs times the number of experiments. If the myopic point of view is a good approximation to the optimal design, this method can significantly reduce computational cost. A highlight of greedy design is variable number of experiments. If the cost of performing experiments is negligible, then experiments can continue to be performed until the total utility reaches a predefined threshold or ceases to grow. Also, suboptimality is not necessarily permanent for our case of the linear problem: since $\mathbf{d}$ are unsorted, any suboptimal $\mathbf{d}$ can be made into $\mathbf{d}^*$ with $|\mathbf{d}| < |\mathbf{d}^*|$ if any $\mathbf{d}^*$ contains $\mathbf{d}$ as a subsequence.

Greedy design choice generally has no optimality guarantees. Any suboptimality can compound, which can make greedy design a poor choice for studies with a large number of experiments. The greedy design algorithm described in Algorithm 2 can be easily adapted to fit continuous design spaces by performing a continuous optimization over d_k for experiment k. For the linear problem, we are fortunate that $\mathbf{d}$ may remain unsorted and that recovery to a nearly-optimal reward is always possible; improvement only requires more experiments. In contrast, other problems may have situations in which suboptimality errors can compound, warranting exploration and lookahead. Our final three methods incorporate a backup mechanism in different forms that propagates information from future experiments to past experiments, sometimes providing optimality guarantees.

2.3 Classical Dynamic Programming

Our next algorithm is the most fundamental: dynamic programming through policy iteration [82]. A dynamic programming algorithm acts on a Markov Decision Process (MDP), which is composed of a state space, action space, reward function, and dynamics. Dynamic programming (DP) attempts to back up the reward signal from future states (e.g. posterior belief states) to previous states (e.g. the starting prior) through iterative one-step lookahead. To apply DP, we must transform our Bayesian sOED problem into an MDP.

The actions a are synonymous to designs in this problem, where the action space $\mathscr{A}$ is equivalent to the design space $\mathscr{D} = \{0.1, 0.2, 0.3, \ldots, 1.0\}$. The action space is fixed and not conditioned on

experiment number or past experiments.

Generally, each state s in the state space $\mathscr{S}$ contains a physical state and belief state. The belief state quantifies the current confidence of the unknown parameters $\boldsymbol{\theta}$. Here, we represent the belief state with a normal distribution $x_k = \mathscr{N}(\boldsymbol{\mu}_k, \Sigma_k)$. There is no physical state in the linear problem. The physical state can present penalties for certain actions or provide constraints on the actions. We allow actions to be chosen freely without penalty here. Any state that is reached after $N = 10$ actions is a terminal state; all other states are non-terminal states. As noted earlier, the objective is only dependent on $\mathbf{d}$. Hence, the collection of $\mathbf{d}$ where $|\mathbf{d}| = 10$ forms a discrete state space.

The reward function provides a reward r of a next state s' given the previous state s and action a, namely $\mathscr{R}(s'|s,a)$. We choose to provide a reward of 0 for all transitions to non-terminal states and a reward equal to the batch objective when reaching a terminal state. Shen and Huan [79] prove that this reward function is equivalent under expectation to providing an incremental reward equal to the greedy objective after every experiment. We will see how DP will back up the reward signal from the terminal state to the start states in an iterative fashion.

The dynamics of an MDP detail the probability of visiting a certain next state and receiving the corresponding reward given a state and action, denoted as $\mathscr{F}$ with underlying distribution $p(s', r|s, a)$. Our Bayesian updates are deterministic and our reward function is deterministic, so therefore our dynamics are deterministic.

The above completes our MDP description. A dynamic programming algorithm will produce a policy $\pi(a|s)$ that details the probability of selecting an action a in state s. Intuitively, it is the design selection strategy. For example, greedy design is on a smaller granularity and will have $\pi(d^*|s) = 1$ for d^* satisfying (2.10). Due to their simplicity, we generally produce output policies that are deterministic for the algorithms we choose to study; although, the policies may certainly be stochastic during training.

The policy induces a value function on the state space $v_\pi(s),\ s \in \mathscr{S}$. The value function quantifies the expected reward received from the given state under the current policy:

$$v_\pi(s) = \mathbb{E}_\pi[R_k + R_{k+1} + \ldots + R_{N-1}|S_k = s] = \mathbb{E}_\pi[R_k + v_\pi(s')|S_k = s]. \tag{2.11}$$

The value function represents the mathematical measure of goodness of existing in a given state under the current policy. An optimal policy π^* with an associated underlying MDP satisfies the following Bellman equation uniquely [82]:

$$v_{\pi^*}(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + v(s')]. \tag{2.12}$$

The above equation can be unraveled as in (2.11) and written in terms of an objective as follows:

$$\begin{aligned} \pi^* = \underset{\pi}{\operatorname{argmax}}\, U(\pi) = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{s_1,s_2,\ldots,s_N|\pi,s_0}\left[\Sigma_{k=0}^{N-1} r_k\right] \\ s.t. \qquad a_k = \pi(s_k), \qquad s_{k+1}, r_k = \mathscr{F}(s_k, a_k). \end{aligned} \tag{2.13}$$

However, the same objective can be written in terms of our Bayesian sOED setup as follows:

$$\begin{aligned} \pi^* = \underset{\pi}{\operatorname{argmax}}\, U(\pi) = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{y_0,y_1,\ldots,y_{N-1}|\pi,x_0}\left[\Sigma_{k=0}^{N-1} u_k\right] \\ s.t. \qquad d_k = \pi(x_k), \qquad y_k, u_k = G(\boldsymbol{\theta}, d_k) + \varepsilon. \end{aligned} \tag{2.14}$$

Note how taking the expectation over y_k is equivalent to taking the expectation over s_{k+1}, as the observation determines a single state, and the reward is completely determined by the next state. Therefore, our OED formulation (2.14) is equivalent to our reinforcement learning formulation (2.13), and we have the same π^*.

Dynamic programming attempts to learn an optimal policy through policy iteration. In policy iteration, the value function is initialized randomly, possibly to 0. The policy is improved at each state by selecting the best design greedily at each state. The policy is then evaluated by turning Equation 2.12 into an update equation as follows:

$$v(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r + v(s')]. \tag{2.15}$$

The above equation sets the value function at a state equal to the best sum of reward and value function of all possible next states. Again, this reward is zero for all non-terminal states and equal to the batch objective for terminal states. Sweeping over all states many times allows the objective,

our only reward signal, to be backed up to the start state and all other states in the entire state space. The pseudocode for policy iteration is shown in Algorithm 3.

Algorithm 3: Policy Iteration in Linear Regression

Input : Set of ordered $\mathbf{d} = \mathscr{S}$
Number of *sweeps*

Create container of tuples $(0, \emptyset)$ for all states in $\mathscr{S}$
for *each sweep* **do**
  for $s \in \mathscr{S}$ **do**
    Initialize *best value* to 0
    Initialize *best a* to $\emptyset$
    for $a \in \mathscr{D}$ **do**
      Determine s' using Bayesian update (2.3)
      if *if s' is not terminal* **then**
        Value of action is $v(s')$
      else
        Value of action is batch objective (2.7)
      end
      if *value* $>$ *best value* **then**
        Set *best a* to *a*
        Set *best value* to *value*
      end
    end
    Update state's container as (*best value*, *best a*)
  end
end

Output: Value and best action of each state

2.3.1 Results

Policy iteration was correctly able to back up the optimal reward of 1.798 to the start state of the empty set and therefore is optimal. Due to the deterministic nature of our problem, if the exact $v_\pi(s)$ are known for $\mathbf{d}$ of length k, then the exact state-values will be learned for $\mathbf{d}$ of length $k-1$ after one backup. The exact state values are learned for $\mathbf{d}$ of length 9 after one backup; hence, the problem was solved after 10 backups due to the start state having length 0.

The ridgeline plot in Figure 2.3 details the evolution of the distribution of $v_\pi(s)$ for fixed length d. Values are higher for earlier states, as there are more chances to act optimally throughout the trajectory; acting optimally from the start state yields the optimal reward. For instance, note that

nearly every **d** after experiment 2 has higher value than nearly any **d** of length 7.

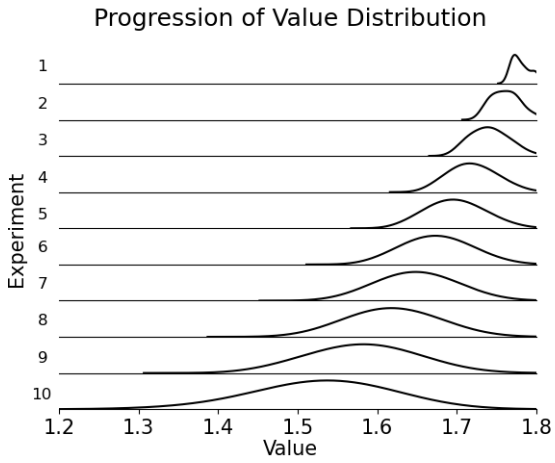

Figure 2.3: Ridgeline plot of state-values after each experiment.

The number of states of length 0 to 9 that have optimal value are, respectively, 1, 2, 3, 4, 5, 5, 5, 4, 3, 2. There is one terminal state that, upon being reached, provides the optimal reward. Since $\mathbf{d}^*$ has four 0.1 designs and six 1.0 designs, the above numbers count the number of states of length k that are composed of designs of 0.1 and 1.0 that have at most four 0.1 designs and six 1.0 designs. Even though some of these states would not have the highest intermediate objective after k experiments, they ultimately can reach the optimal policy if the above constraints are satisfied.

2.3.2 Summary

Policy iteration produces π^* through successfully backing up the terminal reward from later states to earlier states. Thus, a farsighted viewpoint is promising for this problem. Also, the plot of state values shows that careful planning is going to be required for this problem and that poor actions can't be completely remedied through future actions.

Policy Iteration is guaranteed to converge to π^* given enough sweeps. If the transitions are deterministic (as in our case), the sweeps are fast. Furthermore, the best action can then be easily determined from the $v_\pi(s)$ or can be encoded into the states throughout the computations. In general, sweeps need to be made over the entire state space for convergence to be guaranteed. We can mitigate some computation by partitioning the state space by experiment number. For example, we

can sweep states in order of decreasing $|\mathbf{d}|$ and achieve the correct state values. This technique will guide our algorithm in the next section.

One drawback of policy iteration is that if the transition probabilities are unknown, the cost of sampling transitions can further the computational difficulties. Classical policy iteration requires that these transition probabilities are known in advance. In the following section, we simulate these transitions beforehand offline, and in Section 2.5 we gradually approximate them in an online setting. A second main drawback is that policy iteration must visit every state to guarantee convergence. As such, it is computationally infeasible when the state space is large. The final main drawback if that if extra experiments are mandated after the DP analysis is complete (e.g. due to insufficient information gain), another entire analysis is required from scratch.

2.4 Approximate Dynamic Programming

Our fourth algorithm extends DP to approximate dynamic programming (ADP) [52]. While DP traditionally is performed on a discrete state space, ADP allows for a continuous state space. To provide a comparison of our algorithms, we remain in the same discrete state space as before. In the previous section, policy iteration randomly swept the state space and performed iterative updates. Had the state space been ordered and updates been performed for monotonically decreasing $|\mathbf{d}|$, only one sweep of the state space would have been needed to be performed. In ADP, we exploit this structure during our update process by first generating random trajectories of designs and partitioning the states in the trajectories by corresponding experiment number; in this case, we partition based on $|\mathbf{d}|$. Due to the complicated dynamics of problems with continuous action spaces, we adopt a model-free approach that abstains from using the dynamics to produce an update rule. To use a model-free approach, we switch from using state-values to action-values in the ADP algorithm. These action-values are analogous to state values by conditioning a state-value on an action instead of a best action as follows:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[R_k + R_{k+1} + \ldots + R_{N-1} | S_k = s, A_k = a] \\ &= \mathbb{E}_\pi[R_k + \max_{a'} q(s',a') | S_k = s, A_k = a]. \end{aligned} \tag{2.16}$$

The optimization problem is identical to that of regular dynamic programming, but instead we optimize on action-value functions as such:

$$q(s,a) = \max_a \sum_{s',r} p(s',r|s,a)[r + \max_{a'} q(s',a')]. \tag{2.17}$$

The action-values quantify the expected reward received through the immediate reward at the current experiment and the action-value of the next state and best action. Again, we provide a reward of 0 for non-terminal transitions and a reward equal to the batch objective for terminal transitions. Since we learn the action-value function, we can find the best action at a state by determining the maximum of all action-values; for state-values, we needed to use the dynamics and perform a one-step lookahead since the state-value function does not include the action as input.

Our ADP algorithm is a learning algorithm, meaning it performs better after seeing greater amounts of data. We begin by generating 100,000 trajectories, totaling one-million transitions. Then, we construct 10 regressors that input a state-action pair and output its predicted value. Since the rewards of transitioning to terminal states are known, we first regress on the generated experiment 9 inputs and use the rewards given by transitioning to a terminal state as targets. The particular choice was a k-nearest neighbors regressor with 5 nearest neighbors. When fitting our regressor for experiment 8, we use the generated state-action pairs of experiment 8 as inputs and construct targets by taking the maximum predicted value over all actions in the next state using the regressor for experiment 9. Specifically, this turns the above equation into an update as

$$q(s,a) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r + \max_{a'} q(s',a')], \tag{2.18}$$

where the left side of the equation is the input to the regressor, and the right side is the target. This continues for each experiment by using our previously-fitted regressor to create targets and using our generated data as inputs. The process stops with fitting the regressor for experiment 0, yielding 10 regressors in total. Forward inference is fast because we simply use the corresponding regressor to compute 10 action values and choose the best action, for each experiment, totaling 100 state-action function evaluations. Pseudocode for the algorithm is shown in Algorithm 4.

Algorithm 4: Approximate Dynamic Programming for Linear Regression

Input : Generated transitions
Number of experiments N

Initialize *list* of regressors
for $k \in \{N-1, N-2, \ldots, 1, 0\}$ **do**
    Initialize *list* of $(state, action, target)$ triples
    for *transitions from experiment k to* $k+1$ **do**
        if $k == N-1$ **then**
            Transition target is $U(s')$ (2.7)
        else
            Transition target is $\max_{a'} q(s', a')$ (2.17)
        end
        Append $(state, action, target)$ to *list* of transitions
    end
    Regress state-action pairs on target values
    Append regressor to *list* of regressors
end

Output: List of trained action-value regressors

2.4.1 Results

The ADP algorithm produced the design sequence (0.1, 0.1, 0.1, 0.1, 0.3, 1.0, 1.0, 1.0, 1.0, 1.0) and reward of 1.779. This is a top-10 design noting Table 2.1. The regressors were successfully able to capture the differences in data and propagate the values backward in a bootstrapping fashion. After examining a few different runs of the algorithm, it was obvious that the internal k-nearest neighbors algorithm is non-deterministic, as the results were variable; however, designs always had high (> 1.77) reward. The most important observation was that a linear regressor is useless for estimating state-action values: it always chose the greatest or least action due to its linearity. This shows that even for a relatively simple problem, nonlinearity is required.

Since we are bootstrapping, it is important to analyze error propagation. The above violin plot in Figure 2.4 displays the distribution of error for each experiment. Each prediction was compared with the correct value using the state-values of the DP experiment. Error is absolute and not signed. Surprisingly, the error was not compounded and decreases in range the further the reward signal was backed up. We doubt this is due to poor fitting by the algorithm, as our final predicted design sequence was close to optimal; however, we have not tested this phenomenon. For example, this

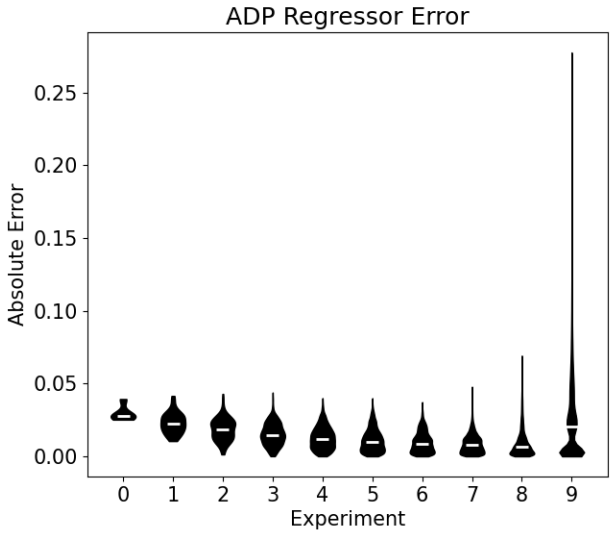

Figure 2.4: Violin plot of regressor error. White hash is median.

may occur simply because the number of states decreases until reaching the start state. Predicted values were generally lower than actual values; we do not have a reason for this phenomenon. We acknowledge that correct ordering of state-action values is more important than minimizing error, and some error is acceptable if it does not interfere with said order.

2.4.2 Summary

Through structuring our state space and incorporating learning, we are able to approximate policy iteration with regressors in one sweep. Similar to DP, ADP produced a near-optimal policy through backing up the terminal values to the start state. We present no optimality guarantees, although with representative training data and a perfect regressor, the results should be optimal. The regressor error did not compound over time, even though we bootstrapped. The most important observation was that nonlinearity is necessary when working with state-action values, and we invoke a powerful nonlinear approximator in the next section.

ADP is our first method that is traditionally meant for continuous spaces. The raw state representations can be used with ADP, and they can be further improved through preprocessing techniques. We zeroed out the means here as they do not provide any information about our goal, but it was

unnecessary. The transition probabilities were not required, as we only needed sample data, not even from an optimal policy. In the next section, we will take the raw state representations as input due to the power of our function approximator.

ADP fails to easily incorporate more experiments into its analysis. The entire study needs to be run again if information gain is not high enough. Optimality is not necessarily guaranteed due to using an approximator and generated data. Also, as posed, this algorithm has poor sample efficiency due to its reliance on a large batch of random sample transitions. The transitions are not focused, and when working on a problem that has a more burdensome update, we will need directed learning. Goal-directed learning from interaction is at the heart of reinforcement learning algorithms that we explore in next section.

2.5 Deep Reinforcement Learning

The last algorithm we consider is deep reinforcement learning (DRL) through deep Q-networks (DQN) [67]. While the previous ADP algorithm took as input a large batch of sample transitions and did not sample any more transitions throughout, RL algorithms employ a directed search through trading off exploration and exploitation. Had our transitions been expensive, ADP would have been infeasible; RL algorithms are more sample efficient and do not require a batch of data beforehand. Since we are applying deep reinforcement learning, the model used to guide exploration and exploitation is a deep neural network. This network takes a state as input and outputs estimated action-values. A highlight of deep Q-networks is the output layer: there is one output neuron per action, allowing a single forward pass of the network to determine all action values. The update follows a bootstrapping Temporal-Difference (TD) update [82], where the following equation holds under optimality:

$$Q(S_t, A_t) = R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t). \tag{2.19}$$

Similar to the approaches in previous sections, this equation is turned into an update by using the right side as a target and the left side as input to the neural network. Gradient descent is performed on the weights of the neural network to minimize this error in batch form. Actions are usually cho-

sen by highest predicted action value, but an ε-greedy strategy is employed to promote exploration. This exploration parameter can be annealed to zero as the policy improves. There are many hyperparameters that we do not note in this paper; there was no tuning required, and as will be shown, the optimal solution was found quickly and easily with a small network. There are other considerations such as copying to another target network every so often, but these details are irrelevant to our current analysis. DQN incorporates bootstrapping, is off-policy due to our exploration parameter, and employs function approximation. These three properties comprise the deadly triad [82], and convergence is not guaranteed. Pseudocode for DQN is shown in Algorithm 5.

Algorithm 5: Deep Q-networks for Linear Regression

Input : Exploration strategy $f(\varepsilon)$
Discount factor γ
Replay memory $\mathscr{N}$ capacity N
Target average reward T
Learning rate α
Batch size M
Initialized neural network Q

do
  for *each experiment time step* **do**
    Choose random number x between 0 and 1
    if $x < f(\varepsilon)$ **then**
      Choose random action a_t
    else
      Select action a_t that maximizes $Q(s_t, a)$ over $a \in \mathscr{A}$
    end
    Take action a_t, receive r_t ((2.7) if terminal, else 0), state s_{t+1} (2.3)
    Store (s_t, a_t, r_t, s_{t+1}) in $\mathscr{N}$
    if *state is terminal* **then**
      Append r_t to list of rewards
      Update smoothed average reward statistic
    end
    Sample batch of size M from $\mathscr{N}$
    Set target for each batch sample as terminal reward or (2.19)
    Perform gradient descent step with learning rate α
  end
while *smoothed average reward* $< T$;

Output: Trained policy Q

2.5.1 Results

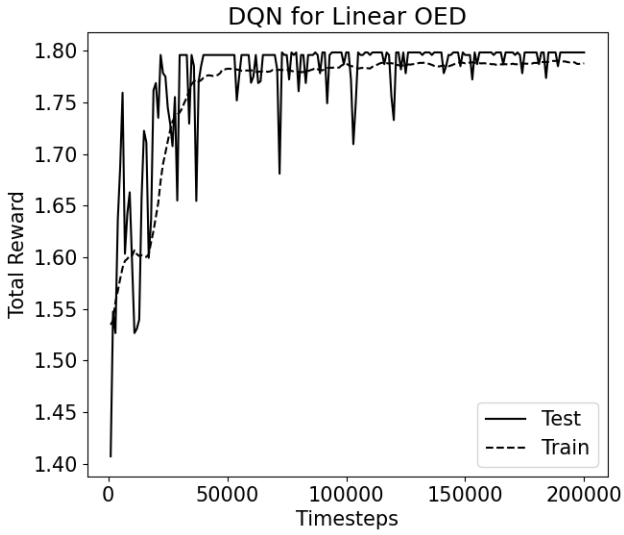

Figure 2.5: Training dynamics of DQN. Results show past 100 smoothed rewards.

Our RL agent successfully arrived at an optimal policy in 5,000 experiments or 50,000 time steps. Shown in Figure 2.5, the agent continually updated its policy with ease, and it must have found the optimal policy well before 50,000 time steps. Since there was random exploration, it took a considerable number of time steps to ensure the smoothed average reward was greater than that of the second-best policy. Designs 0.1 and 1.0 were chosen most frequently and dominated the selection after 1,000 experiments. Due to the random nature of the algorithm and weight updates, there is considerable noise even when smoothed. We expect that these results were simply due to the simplicity of the problem and deterministic dynamics; more challenging problems will require a deeper neural net, more sophisticated exploration strategies, and possibly a learned model.

2.5.2 Summary

Through trading off exploration and exploitation, DQN was able to quickly converge to an optimal policy. The values of terminal states were backed up through Q-learning updates, a type of TD update. The algorithm had no trouble converging to the optimal policy and started learning a correct strategy (choosing 0.1 and 1.0 designs) nearly instantaneously. This provides evidence that DRL

can be successfully applied to more complicated RL problems in the Bayesian sOED domain.

These results showcase the efficiency of our algorithm. Even with $1,000,000$ transitions, the ADP algorithm was not optimal. In contrast, DQN needed only $50,000$ transitions to become optimal. Further, it is able to detail its progress while still working to achieve optimality. Quantifying intermediate progress follows the theme of sOED in which we allocate resources as needed. Preprocessing is not required, as neural networks are able to learn state features automatically. Consequently, the final policy is applicable to states that the agent did not even experience during training.

The only blemish of DQN from our discussion here is the lack of convergence guarantees; however, the algorithm may certainly converge in practice as shown in our example. Great care must be taken through cataloging hyperparameters, graphing training dynamics, and comparing with other successful applications for such DRL methods to succeed practically. Considering the advantages of sample efficiency, online updates, automatic model-learning, and the ever-increasing computational power of GPUs, we end our analysis of the five algorithms by acknowledging that trading convergence guarantees for the sheer power of RL methods may be justified and unavoidable.

2.6 Conclusion

Through quantifying information gain by KL-divergence, we were able to create a batch design objective function that quantifies expected information gain between the prior confidence distribution and posterior distribution after completion of all experiments. The objective has a corresponding closed-form analytic expression that allows for fast evaluation. The batch design objective is modified for greedy design, as the objective is only dependent on the increase in KL-divergence over a single experiment. Through reformulating the Bayesian sOED problem as a RL problem, we were able to successfully apply DP, ADP, and DRL techniques to compute optimal policies. Each algorithm was able to produce optimal or near optimal policies in this chapter, possibly due to the existence of a closed-form analytic expression that did not require sampling observations. The objective in the next chapter does not have a corresponding closed-form analytic expression, which requires Monte Carlo sampling to evaluate. As such, we may not simply exhaust all states in the state set, and some algorithms will be clearly suboptimal.

Chapter 3

Optimal Experimental Design in Nonlinear Parameter Estimation with State and Velocity Data

We continue our investigation of Bayesian sOED through attempting to estimate parameters in a time dependent ordinary differential equation (ODE) from measurement of the solution and its derivative at various points in time. This problem has been well-studied in the literature, and we now give a brief review. In the case when the ODEs can be solved analytically, then a nonlinear least squares (NLS) method is viable. Two fundamental methods in solving such equations are the methods of Taylor series and gradient descent. Levenberg [57] and Marquardt [63] independently developed the Levenberg-Marquardt algorithm that optimally interpolates between a Taylor series method and gradient descent method. Bard [9], Domselaar and Hemker [86], and Benson [11] all consider the case when a closed-form solution is not available. Bard showcases a thorough overview of nonlinear parameter estimation. Domselaar and Hemker improve the initial parameter estimates with a multiple shooting technique and provide confidence intervals. Benson considers when the solution vector contains missing data.

Varah [87] addresses drawbacks in the literature that (i) the solution is sensitive to initial conditions, (ii) estimating the parameters is necessary, (iii), the algorithms require heavy computation,

and (iv) the linear case does not reduce to a faster algorithm. Varah's approach combines cubic spline functions and a least squares metric to fit the data as introduced by de Boor [31]. Voit and Almeda [89] provided an example that highlights the ease of computation of Varah's method as opposed to traditional least squares. Brunel [21] constructed a new estimator using a weighted integral of the squared deviation. Gugushvili and Klassen [43] implemented kernel smoothing as opposed to spline smoothing. Other improvements have been made to Varah's method such as those in Wu et al. [91], Brunel et al. [22], and Dattner and Gugushvili [30].

Another avenue used to estimate the parameters is to first solve the ODE numerically and then incorporate Nonlinear Least Squares. Hairer et al. [46] and Mattheij and Molenaar [64] apply RK4 before applying NLS. Xue et al. [92] establish asymptotic properties for the estimators which may include numerical and measurement error. Instead of RK4, Ramsey [73] approximates the solution through a linear combination of basis functions where a penalized optimization problem provides the coefficients. The next step of Ramsey's method is similar to Varah's in that yet another optimizer based on the unknowns is established. Qi and Zhao [72] similarly establish asymptotic properties of this estimator.

Parameter estimation is the goal of this work, and more specifically, we apply Bayesian parameter estimation techniques to infer the most information about our parameters given a fixed number of experiments. Bayesian parameter estimation is a more recent field. Early attempts include solving the ODE analytically to form the likelihood term. Afterwards, MCMC techniques are applied to sample the posterior [39, 75, 40]. Other works extend the generalized profiling approach by Ramsey [73] to a Bayesian analogue [23, 24, 53]. Bhaumik and Ghosal [13] extend Varah [87] and Brunel [21] to Bayesian parameter estimation through first assuming a prior on the coefficients of the basis functions of a B-spline. They compute the posterior of the parameters using the posterior of the coefficients of basis functions. Bhaumik and Ghosal [14] modify the previous method such that the resultant Bayes estimator is asymptotically efficient.

In this chapter, we consider two distinct first order scalar ODEs as follows:

$$\frac{dz}{dt} = \theta_1 z - \theta_2 z^2, \tag{3.1}$$

$$\frac{dz}{dt} = \theta_2\sqrt{\theta_1^2 - z^2}. \tag{3.2}$$

Hereafter throughout the remainder of the document, we will refer to the ODE in (3.1) as ODE1 and similarly that in (3.2) as ODE2. In this chapter, we assume knowledge of the solution and solution derivative values at the design points. Each observation of the solution and derivative is corrupted with noise. We choose to study two separate ODEs, as the parameter estimation associated with ODE1 is linear, and that with ODE2 is non-linear; techniques from the previous chapter can be applied to ODE1, while they cannot be applied to ODE2.

The parameters $\boldsymbol{\theta} = (\theta_1, \theta_2)$ are drawn from a joint uniform distribution $\mathscr{U} = \mathscr{U}_1 \times \mathscr{U}_2$ where $\mathscr{U}_1 = (0.05, 0.95)$ and $\mathscr{U}_2 = \mathscr{U}(0.05, 0.35)$. The slightly narrower region for $\mathscr{U}_2$ is necessary for continuity of the solution of ODE2. The designs are chosen from the discrete set $\mathscr{D} = \{0, 1, 2, \ldots, 9\}$, where the design is synonymous with the time choice. We allow $N = 2$ experiments.

Both ODEs are separable. ODE1 has initial condition $z(\boldsymbol{\theta}, 0) = 2$ and ODE2 has initial condition $z(\boldsymbol{\theta}, 0) = 0$, leading to the following solutions (see Figure 3.1):

$$z(\boldsymbol{\theta}, t) = \frac{2\theta_1 e^{\theta_1 t}}{2\theta_2(e^{\theta_1 t} - 1) + \theta_1}, \tag{3.3}$$

$$z(\boldsymbol{\theta}, t) = \begin{cases} \theta_1 \tan(\theta_2 t)|\cos(\theta_2 t)| & \theta_2 t < \pi/2 \\ -\theta_1 \tan(\theta_2 t)|\cos(\theta_2 t)| & \theta_2 t > \pi/2. \end{cases} \tag{3.4}$$

The solution derivatives, hereafter known as velocities, are as follows (see Figure 3.2):

$$\dot{z}(\boldsymbol{\theta}, t) = \frac{2\theta_1^2 e^{\theta_1 t}(\theta_1 - 2\theta_2)}{(2\theta_2(e^{\theta_1 t} - 1) + \theta_1)^2}, \tag{3.5}$$

$$\dot{z}(\boldsymbol{\theta}, t) = \begin{cases} \theta_1\theta_2|\cos(\theta_2 t)| & \theta_2 t < \pi/2 \\ -\theta_1\theta_2|\cos(\theta_2 t)| & \theta_2 t > \pi/2. \end{cases} \tag{3.6}$$

The corresponding forward models are of the form:

$$(z(\boldsymbol{\theta}, d_k), \dot{z}(\boldsymbol{\theta}, d_k)) = G_k(\boldsymbol{\theta}, d_k). \tag{3.7}$$

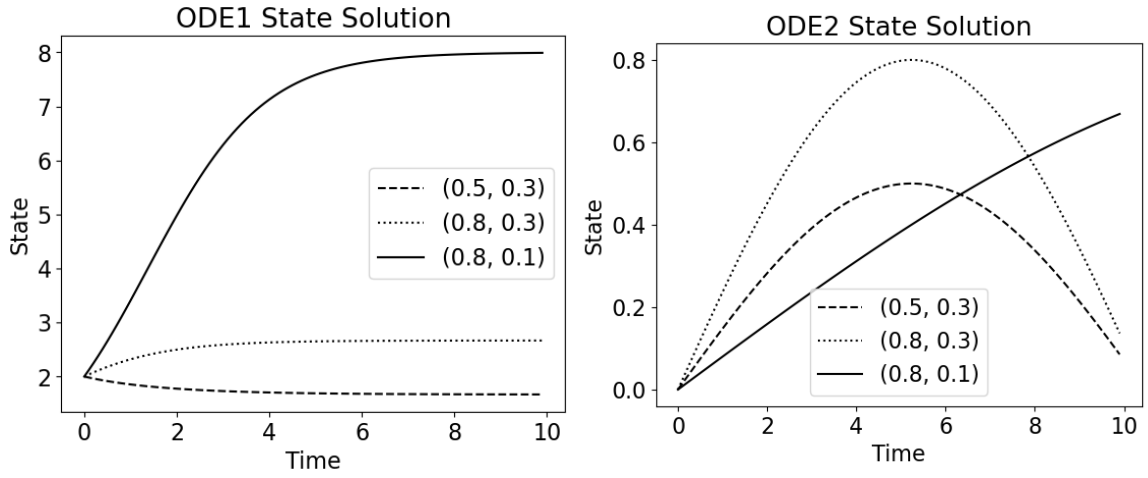

Figure 3.1: Graph of solutions of ODE1 (left) and ODE2 (right) for choices of (θ_1, θ_2).

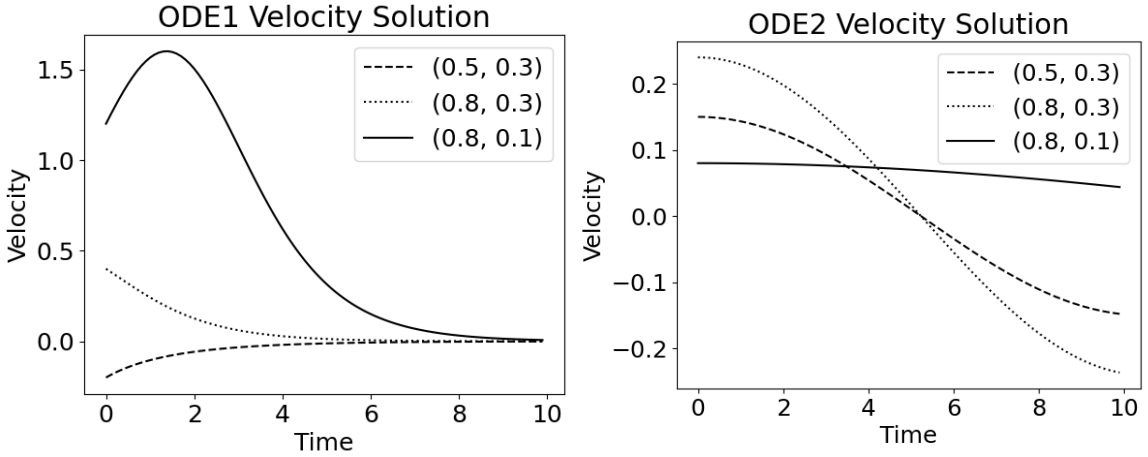

Figure 3.2: Graph of velocity derivatives of ODE1 (left) and ODE2 (right) for choices of (θ_1, θ_2).

We assume that measurements are corrupted with normal noise, where

$$y_k = (u_k, v_k) = G_k(\boldsymbol{\theta}, d_k) + \varepsilon_k, \qquad \varepsilon_k \sim \mathcal{N}\left(\boldsymbol{\mu} = \mathbf{0}, \Sigma = \begin{pmatrix} \sigma_u & 0 \\ 0 & \sigma_v \end{pmatrix}\right). \tag{3.8}$$

Here, σ_u and σ_v are assumed to be known constants. In our experiments, we have $\sigma_u = \sigma_v = 0.1$.

We now compare and contrast these ODE models and the linear model from Chapter 2. These ODE models are similar to the linear model in the previous section, as there are two unknown parameters with known constant observation noise. Also, they both take a single-dimensional design as input. A major difference is the multi-dimensional output of the state and velocity. The

linear problem had a normal posterior and allowed for quick Bayesian updates. Through repeated attempts, we were unable to simplify the posterior distribution of this ODE problem to a known distribution with a set of structured parameters (e.g. mean, standard deviation). Instead, we resorted to approximate methods and encoded the posterior as a discrete 2-dimensional grid when necessary, thereby heavily increasing computation.

Following this section, we detail a variety of select algorithms to compute a design policy. Each algorithm boasts certain strengths, and some are even optimal. The algorithms include batch design with single and multiple observations, continuous batch design with black-box Bayesian optimization, greedy design, multi-armed bandit optimization, approximate dynamic programming, and deep reinforcement learning. We will conclude by summarizing results with emphasis on optimality. To our knowledge, deep reinforcement learning has not been compared to all other methods above or applied to inverse problems of differential equations models.

3.1 Batch Design

The first algorithm we use to analyze the OED problem on ODE1 and ODE2 is batch design [35, 4]. Similar to that in the previous section, we start by discretizing the possible $\mathbf{d}$, in this case with integer $d \in \mathscr{D} = \{0, 1, 2, \ldots, 9\}$. We maximize the objective with one design and then with two designs, known as one-step and two-step batch design. The results of one-step batch design will be used when choosing the first design of a greedy approach in Section 3.4. In the next section, we employ black-box Bayesian optimization to optimize the continuous version of one-step and two-step batch.

The linear problem from the previous chapter yielded successive normal posteriors that could be described with a set of unstructured parameters, the mean and standard deviation. Furthermore, there was a closed-form expression for the objective that only used design as input. Such a problem was easily solved by naive techniques; our approach with the current ODE1 and ODE2 will be much more complex. Instead, we adopt an approach by Huan and Marzouk [51] wherein we approximate the objective through Monte Carlo simulation. The objective can be reduced to a double integral of the parameters and observations, mandating only evaluations of the likelihood term. Since our

approach uses a uniform prior, we can easily sample randomly from the uniform prior distribution. In the case of other sequential algorithms, we will need to sample from the posterior after one completed experiment.

Our results show that the solutions of one-step batch design and two-step batch design do not coincide. One-step batch design is greedy from the perspective of two-step batch design because it maximizes a one-step objective. Two-step batch design may choose a design that is suboptimal from the perspective of one-step batch design but attains a greater objective than the one-step solution ever could have achieved with any second design.

3.1.1 Methodology

Our objective function is the same as that of batch design in the previous chapter with fewer ($N = 2$) experiments:

$$\mathbf{d}^* = \arg\max_{\mathbf{d}\in\mathscr{D}^2} U(\mathbf{d}). \tag{3.9}$$

Due to the lack of an analytic expression for $U(\mathbf{d})$, we adopt a sampling methodology by Huan and Marzouk [51] to compute the objective. This method randomly samples parameters $\boldsymbol{\theta}$ and observations $\mathbf{y}$ conditioned on $\boldsymbol{\theta}$ to evaluate the likelihood function and ultimately the objective. We now present the objective from Lindley [60]:

$$\begin{aligned} U(\mathbf{d}) &= \int_{\mathscr{Y}} \int_{\Theta} u(\mathbf{d},\mathbf{y},\boldsymbol{\theta}) p(\boldsymbol{\theta},\mathbf{y}|\mathbf{d}) d\boldsymbol{\theta} d\mathbf{y}, \\ &= \int_{\mathscr{Y}} \int_{\Theta} u(\mathbf{d},\mathbf{y},\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d}) d\boldsymbol{\theta} p(\mathbf{y}|\mathbf{d}) d\mathbf{y}. \end{aligned} \tag{3.10}$$

We again use KL-divergence as the utility function as follows:

$$u(\mathbf{d},\mathbf{y}) = \int_{\Theta} p(\tilde{\boldsymbol{\theta}}|\mathbf{y},\mathbf{d}) \ln\left[\frac{p(\tilde{\boldsymbol{\theta}}|\mathbf{y},\mathbf{d})}{p(\tilde{\boldsymbol{\theta}})}\right] d\tilde{\boldsymbol{\theta}}. \tag{3.11}$$

Inputting the KL-divergence as our utility function and simplifying yields the following:

$$\begin{aligned}
U(\mathbf{d}) &= \int_{\mathscr{Y}} \int_{\Theta} u(\mathbf{d},\mathbf{y}) p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d}) d\boldsymbol{\theta} p(\mathbf{y}|\mathbf{d}) d\mathbf{y}, \\
&= \int_{\mathscr{Y}} u(\mathbf{d},\mathbf{y}) p(\mathbf{y}|\mathbf{d}) d\mathbf{y}, \\
&= \int_{\mathscr{Y}} \int_{\Theta} p(\tilde{\boldsymbol{\theta}}|\mathbf{y},\mathbf{d}) \ln\left[\frac{p(\tilde{\boldsymbol{\theta}}|\mathbf{y},\mathbf{d})}{p(\tilde{\boldsymbol{\theta}})}\right] d\tilde{\boldsymbol{\theta}} p(\mathbf{y}|\mathbf{d}) d\mathbf{y}, \\
&= \int_{\mathscr{Y}} \int_{\Theta} p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d}) \ln\left[\frac{p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})}{p(\boldsymbol{\theta})}\right] d\boldsymbol{\theta} p(\mathbf{y}|\mathbf{d}) d\mathbf{y}, \\
&= \int_{\mathscr{Y}} \int_{\Theta} \ln\left[\frac{p(\mathbf{y}|\boldsymbol{\theta},\mathbf{d})}{p(\mathbf{y}|\mathbf{d})}\right] p(\mathbf{y}|\boldsymbol{\theta},\mathbf{d}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} d\mathbf{y}, \\
&= \int_{\mathscr{Y}} \int_{\Theta} \{\ln[p(\mathbf{y}|\boldsymbol{\theta},\mathbf{d})] - \ln[p(\mathbf{y}|\mathbf{d})]\} p(\mathbf{y}|\boldsymbol{\theta},\mathbf{d}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} d\mathbf{y}.
\end{aligned} \tag{3.12}$$

Note how the fourth equality can be written as $\mathbb{E}_{\mathbf{y}|\mathbf{d}}[D_{KL}(p(\cdot|\mathbf{y},\mathbf{d})||p(\cdot))]$ (refer to (1.5)). Therefore, when using the utility function of KL-divergence, our objective is the expected KL-divergence. Continuing the method set out by Huan and Marzouk [51], we apply Monte Carlo sampling to evaluate the double integral. Using their notation, we have the following:

$$U(\mathbf{d}) \approx \frac{1}{m_{out}} \sum_{i=1}^{m_{out}} \{\ln[p(\mathbf{y}^{(i)}|\boldsymbol{\theta}^{(i)},\mathbf{d})] - \ln[p(\mathbf{y}^{(i)}|\mathbf{d})]\}. \tag{3.13}$$

We draw random samples $\boldsymbol{\theta}^{(i)}$ from the prior distribution and sample observations $\mathbf{y}^{(i)}$ from the likelihood. We denote m_{out} as the number of samples in the outer Monte Carlo estimate, as we continue by sampling to evaluate the inner evidence term:

$$p(\mathbf{y}^{(i)}|\mathbf{d}) = \int_{\Theta} p(\mathbf{y}^{(i)}|\boldsymbol{\theta},\mathbf{d}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} \approx \frac{1}{m_{in}} \sum_{j=1}^{m_{in}} p(\mathbf{y}^{(i)}|\boldsymbol{\theta}^{(i,j)},\mathbf{d}). \tag{3.14}$$

Here, there are a number m_{in} samples of $\boldsymbol{\theta}^{(;j)}$ drawn from the prior. Most importantly, note how the two needed terms for the sampling are the prior and likelihood, again assumed to be known. In our case, we know the exact value of the states and velocities given the $\mathbf{d}$ and $\boldsymbol{\theta}$, and we know the distribution of the noise ($\mathscr{N}(0,0.1)$), so we can compute the likelihood. Huan and Marzouk [51] show that reusing the samples $\boldsymbol{\theta}^{(i)}$ from the outer integral inside the inner integral (or vice versa), prevents underflow in the evidence estimate and contributes a very small effect to the bias.

We first begin by determining the best one-step design. It is found through evaluating (3.13) and (3.14) at 100 unique $d \in \mathscr{D} = \{0.0, 0.1, 0.2, 0.3, \ldots, 9.8, 9.9\}$. Since these results will be used in Section 3.4, we perform the experiments on a finer-grained design space for accuracy. We let the number of m_{out} samples be 10^4 and the number of m_{in} samples be 10^4. In our second experiment, we extend to the two-step batch case. We calculate the objectives of all designs $\mathbf{d} \in \mathscr{D} \times \mathscr{D} = \{0, 1, 2, \ldots, 9\} \times \{0, 1, 2, \ldots, 9\}$. This amounts to $\binom{11}{2} = 55$ unique unordered pairs, although we test all 100 ordered pairs to gauge convergence of the algorithm.

Again, we sample our parameters from a joint uniform prior distribution $\boldsymbol{\theta} \sim \mathscr{U}_1 \times \mathscr{U}_2 = \mathscr{U}(0.05, 0.95) \times \mathscr{U}(0.05, 0.35)$. Uniform priors enable fast sampling, and we refrain from reusing samples as described above. The likelihood function can be broken down into

$$p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{d}) = p((\mathbf{u}, \mathbf{v})|\boldsymbol{\theta}, \mathbf{d}) = p(\mathbf{u}|\boldsymbol{\theta}, \mathbf{d}) p(\mathbf{v}|\boldsymbol{\theta}, \mathbf{d}) \tag{3.15}$$

assuming independence of the state and velocity observations. Similarly, we use independence to transform a likelihood of a number of observations into a product of individual likelihoods:

$$p((\mathbf{u}, \mathbf{v})|\boldsymbol{\theta}, \mathbf{d}) = \prod_k p((u_k, v_k)|\boldsymbol{\theta}, d_k). \tag{3.16}$$

The pseudocode of the multi-step batch algorithm is showcased in Algorithm 6.

3.1.2 Results

The results of one-step batch design are shown in Figure 3.3 and Table 3.1. There is a notable peak of ODE1 around $d_0 = 5$ with objective score 2.98. Specifically, a single observation of state and velocity of ODE1 yields highest objective around $d_0 = 5$, assuming one experiment. In similar fashion, the objective of ODE2 with a single design is maximized at $d_0 = 9$ with objective score 1.15. The assumption of a single experiment is taken when we adopt a greedy paradigm that attempts to maximize the objective at each step of the experiment. Both ODEs generally preferred designs further from 0, and the objective for ODE2 is monotonically increasing.

Sometimes, OED is assumed to take an observation at the first time $d_0 = 0$ when absence of

Algorithm 6: Batch Design with Double Integral Sampling

Input : Design $\mathbf{d}$
Number of outer integral samples n_{out}
Number of inner integral samples n_{in}

Set $total\_util = 0$
for n_{out} **do**
  Sample $\boldsymbol{\theta} = (\theta_1, \theta_2)$ from prior
  Compute true state and velocity $\mathbf{z}, \dot{\mathbf{z}}$ (3.3, 3.4, 3.5, 3.6)
  Simulate state and velocity observations $\mathbf{u}, \mathbf{v}$ (3.8)
  $likelihood = \prod_k p((u_k, v_k)|\boldsymbol{\theta}, d_k)$(3.16)

  Compute Evidence
  Set $evidence = 0$
  for n_{in} **do**
    Sample $\tilde{\boldsymbol{\theta}} = (\tilde{\theta}_1, \tilde{\theta}_2)$ from prior
    Compute true state and velocity $\tilde{\mathbf{z}}, \tilde{\dot{\mathbf{z}}}$ (3.3, 3.4, 3.5, 3.6)
    Simulate state and velocity observations $\tilde{\mathbf{u}}, \tilde{\mathbf{v}}$ (3.8)
    $evidence \mathrel{+}= \prod_k p((\tilde{u_k}, \tilde{v_k})|\tilde{\boldsymbol{\theta}}, d_k)$(3.16)
  end

  $total\_util \mathrel{+}= \log(likelihood) - \log(evidence)$
end
Compute $expected\_util = total\_util/n_{out}$

Output: Expected Utility

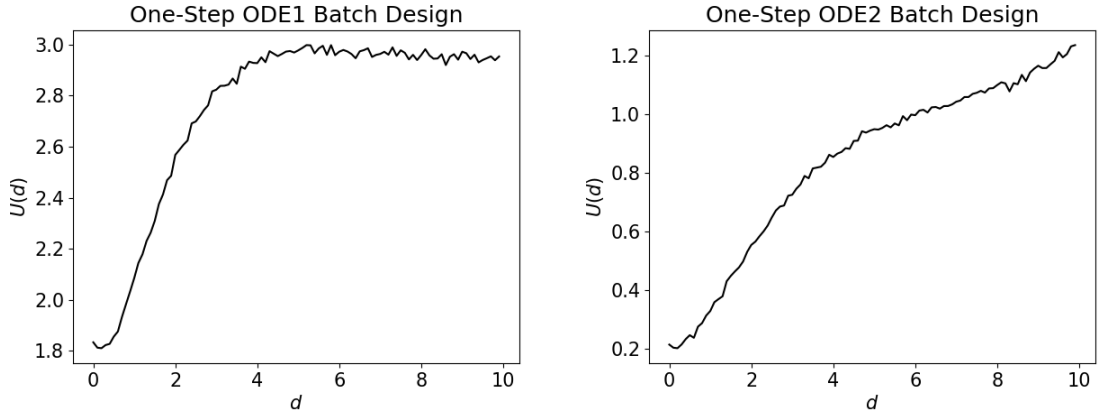

Figure 3.3: Objective graph of one-step batch design for ODE1 (left) and ODE2 (right).

	$U(d)$	
d	ODE1	ODE2
0	1.83	0.20
1	2.07	0.31
2	2.67	0.54
3	2.81	0.74
4	2.93	0.86
5	2.98	0.94
6	2.98	1.02
7	2.97	1.06
8	2.96	1.10
9	2.95	1.15

Table 3.1: Results of one-step batch for integer d.

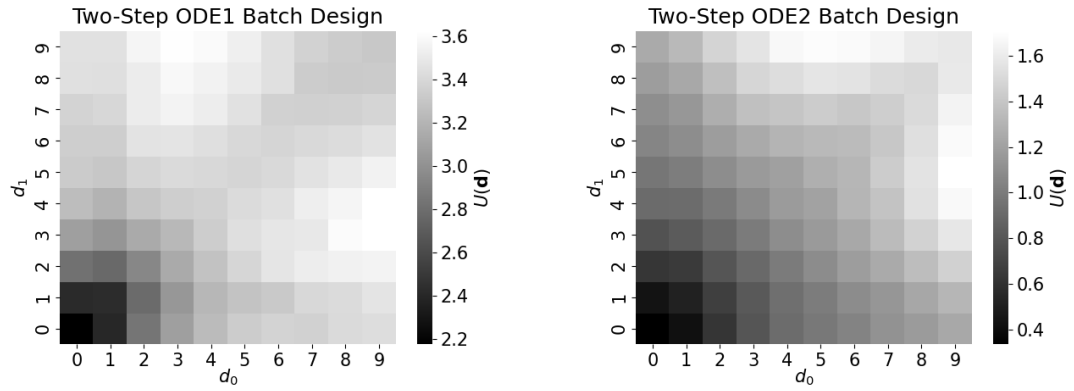

Figure 3.4: Graph of two-step batch design of ODE1 (left) and ODE2 (right).

information is present at that timestep. In our case, we have boundary conditions known for the state. Therefore, the velocity measurement provides useful information at $d_0 = 0$ that contributes to a non-zero objective.

Next, we study ODE1 and ODE2 in the context of two-step batch design. Looking at the results in Figure 3.4, we see symmetry across the main diagonal, as the objective (3.12) is invariant to the order of design. If we were to adopt a strict forward view, then we would only consider designs $d_0 < d_1$. Very generally, choosing larger designs is better than choosing smaller designs. For both ODEs, choosing a small-medium design and a large design yields the highest reward (see Table 3.2). For ODE1, $\mathbf{d}^* = (3,9)$ with objective score 3.67 and for ODE2, $\mathbf{d}^* = (6,9)$ with score 1.72.

ODE1		ODE2	
d	$U(\mathbf{d})$	**d**	$U(\mathbf{d})$
(3, 9)	3.67	(6, 9)	1.72
(4, 9)	3.67	(5, 9)	1.71
(3, 8)	3.63	(4, 9)	1.69
(2, 9)	3.61	(7, 8)	1.65
(4, 8)	3.60	(9, 9)	1.60
(3, 7)	3.57	(3, 9)	1.59
⋮	⋮	⋮	⋮
(1, 1)	2.44	(1, 1)	0.51
(0, 1)	2.42	(0, 1)	0.43
(0, 0)	2.14	(0, 0)	0.34

Table 3.2: Results of two-step batch design. Designs ranked from best (top) to worst (bottom).

3.1.3 Discussion

Batch design employs multi-step lookahead but lacks feedback. Multi-step lookahead allows a seemingly suboptimal design to be taken first while providing a higher objective in the long-run. The primary deficiency is that batch design policy is on the granularity of a design sequence $\mathbf{d}$ and not an individual design d. Other methods we consider will make sequential decisions with respect to the current belief state.

To compute the objective, we need to sample $\boldsymbol{\theta}$, observations $\mathbf{y}$, and compute the likelihood. Therefore speed of sampling and reusing samples can speed up computation. Also, we compiled the likelihood function to increase the speed. One notable strength of batch design is parallelization; we can easily parallelize Monte Carlo samples of the entire double integral.

Two-step batch disagrees with one-step batch in that the first design of $d_0 = 5$ in ODE1 is not used in two-step batch design: instead, $\mathbf{d}^* = (3,9)$. In contrast, sequential design incorporates feedback, so even though $(3,9)$ is optimal for the two-step batch design objective, sequential design has knowledge of the posterior and may surpass batch design through its own objective. As long as a backwards view is available, two-step batch design of ODE2 agrees with one-step batch design. The design 9 is used in the best two-step design sequences in ODE2, and a backwards view would allow smaller design 6, 5, 4, etc. to be taken second. In contrast, the forward view would mandate both

designs to be taken as 9. These results form a baseline that can be used to showcase the advantages of a sequential approach, pending higher objective scores.

3.2 Bayesian Optimization

Next, we extend $\mathscr{D}$ to a continuous space and study the continuous batch design problem. All $\mathbf{d}$ now may have continuous values and will lie in the design space $\mathscr{D} = [0,9]^2$. In many problems, evaluating the underlying model is expensive and noisy. One technique we will see in the next chapter to handle such problems is multi-armed bandits that navigate the exploration vs. exploitation dilemma to maximize the one-step objective while being penalized constantly for suboptimal choices [74, 42, 41]. We instead employ a Bayesian sOED technique [27] to maximize the objective through as few evaluations as possible, namely black-box Bayesian optimization via Gaussian processes.

Gaussian processes are a set of dependent random variables for each data point for which each finite subset is a multivariate normal distribution [90]. We use Gaussian processes as predictors, as they can output an objective estimate and variance of confidence at each point in $\mathscr{D}$. Gaussian processes work naturally with Bayesian optimization, as they provide analytic posteriors for the mean and covariance of the posterior distributions. Bayesian optimization sequentially uses an acquisition function to determine the next design to test; the acquisition function manages the exploration vs exploitation dilemma by taking into account uncertainty and the objective score at a grid point. The resultant algorithm is coined the GP-UCB algorithm [7, 8]. Srinivas et al. apply Gaussian processes to experimental design using GP-UCB and derive regret bounds on the maximal information gain [81].

The primary difference between discrete batch design and Bayesian optimization, as stated here, is the goal of the algorithm. While discrete batch design attempts to determine the accuracy of the objective and therefore the optimal design sequence, continuous batch design wants to determine the optimal design sequence in a relatively small number of steps. Through the use of an acquisition function, the choice of design sequence is directed to valuable and uncertain areas of the design space.

Our results show that Bayesian Optimization with Gaussian processes can reproduce the results of discrete batch design in many fewer samples. Using an order of magnitude fewer samples (i.e. 10 instead of 100), the algorithm is able to produce a $\mathbf{d}$ with corresponding objective relatively near the one found for discrete batch. Furthermore, the estimates of the objective at other grid points are also similar. The upcoming figures in this section will illuminate the strategy and power of the algorithm as an alternative online approach.

3.2.1 Methodology

We apply Gaussian processes to predict the objective and variance of confidence at any point in the design space. For observations $\mathbf{y}$ and designs $\mathbf{d}$ from model

$$\mathbf{y} = f(\mathbf{d}) + \varepsilon \tag{3.17}$$

with $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, the posterior over f has mean μ, covariance k, and variance σ^2 where

$$\mu(\mathbf{d}) = k(\mathbf{d})^T (K + \sigma^2 \mathbf{I})^{-1} \mathbf{y}, \tag{3.18}$$

$$k(\mathbf{d}, \mathbf{d}') = k(\mathbf{d}, \mathbf{d}') - k(\mathbf{d})^T (K + \sigma^2 \mathbb{I})^{-1} k(\mathbf{d}'), \tag{3.19}$$

$$\sigma^2(\mathbf{d}) = k(\mathbf{d}, \mathbf{d}). \tag{3.20}$$

Here, $k(\mathbf{d}) = [k(d_0, \mathbf{d}), \ldots, k(d_{n-1}, \mathbf{d})]$ and $K_{(\mathbf{d},\mathbf{d}')} = k(\mathbf{d}, \mathbf{d}')$. Most fundamental to Bayesian optimization with Gaussian processes is an algorithm called GP-UCB [7, 8]. This algorithm is similar to the UCB algorithm of multiarmed bandits in that the next point is chosen to maximize the expected mean plus a hyperparameter constant times the variance at the point. The acquisition function can be written as follows:

$$\mathbf{d}^* = \underset{\mathbf{d} \in \mathscr{D}}{\operatorname{argmax}}\, \mu_{t-1}(\mathbf{d}) + \sqrt{\beta_t}\, \sigma_{t-1}(\mathbf{d}). \tag{3.21}$$

We use the fmfn Bayesian optimization package [68] for this analysis. We manually limit the number of grid points to test at 10. The algorithm's output is an objective score and uncertainty at each point in $\mathscr{D}$. The actual calculation of the objective is equivalent to that of discrete batch design:

we choose $\mathbf{d}$ where $|\mathbf{d}| = 2$ and then evaluate the double integral in (3.12).

3.2.2 Results

The results of one-step Bayesian optimization are shown in Table 3.3 and Figure 3.5. First, we analyze the case of ODE1. The first three designs are 3.37, 9.00, and 0.97. Up until this point, the algorithm acted in an exploratory manner, selecting points that are relatively uniform in the design space. Afterwards, the algorithm decided to exploit its best design of 9.00 again and check to determine if the high objective of 2.94 was not simply noise. The algorithm tested two other points near the boundary of 9.00, and similar objectives were found. Afterwards, the algorithm concluded by testing regions of high uncertainty. Of the last four designs, a design of 5.86 produced the best objective score of 2.98. Figure 3.5 displays the acquisition function, points tested, estimated objective, and standard deviation of confidence estimated by the Gaussian Process. Clearly, the regions in which fewer test points have been taken have more uncertainty.

The results for ODE2 are similar. The algorithm begins by choosing designs 4.92 and 0.00. Due to the great difference in objective, the algorithm chooses to check a point near 5 again. Receiving high objective, it checks a region of high uncertainty on the design space boundary of 9.00. The algorithm checks three more points in an exploratory fashion before testing two more points near the boundary and one more exploratory point. The results of the algorithm are shown in Figure 3.5. There is most uncertainty in the area of the design space that is unchecked and expected to have low objective.

There were two major similarities in the results for ODE1 and ODE2. First, they both grappled with the exploration vs. exploitation dilemma quite well through leveraging the acquisition function. After every few exploitation points, an exploration point would be chosen and vice versa. Some tuning of the hyperparameter constant was necessary, and therefore, some knowledge of the noise level is helpful beforehand. Second, both algorithms understood the need to check the design space boundaries. These boundaries were chosen early in both algorithms and contributed to the speed of convergence.

Next, we turn out attention to the two-step case in which we test $\mathbf{d}$ where $|\mathbf{d}| = 2$. The results

	ODE1		ODE2	
#	**d**	$U(\mathbf{d})$	**d**	$U(\mathbf{d})$
1	3.37	2.8	4.92	0.93
2	9.00	2.94	0.0	0.21
3	0.97	2.05	4.85	0.92
4	9.00	2.94	9.00	1.12
5	8.89	2.93	7.42	1.06
6	8.75	2.92	2.35	0.60
7	5.86	2.98	6.38	1.01
8	7.11	3.01	8.33	1.10
9	4.59	2.94	8.74	1.10
10	0.00	1.82	5.77	1.00

Table 3.3: Results of one-step Bayesian optimization applied to ODE1 and ODE2.

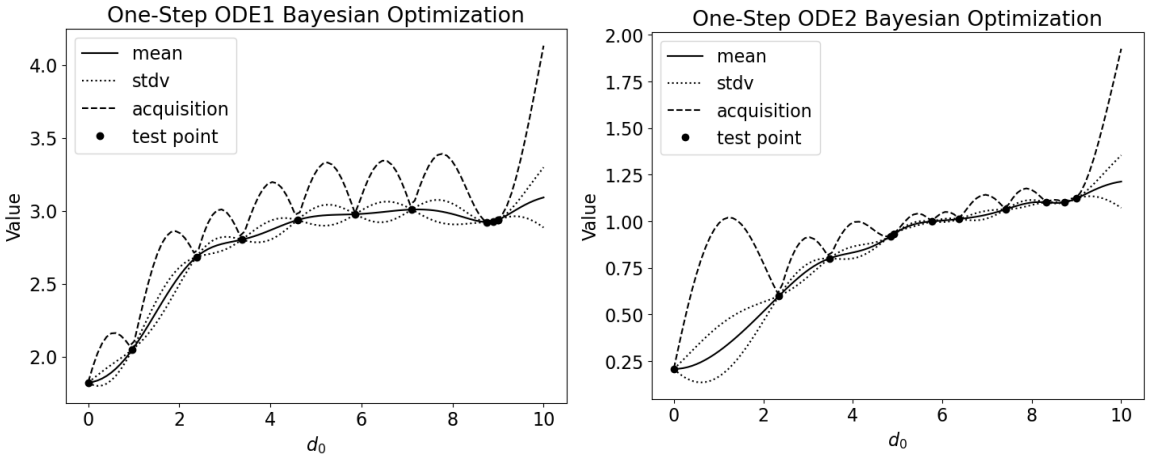

Figure 3.5: One-step Bayesian optimization of ODE1 (left) and ODE2 (right) with 10 test points.

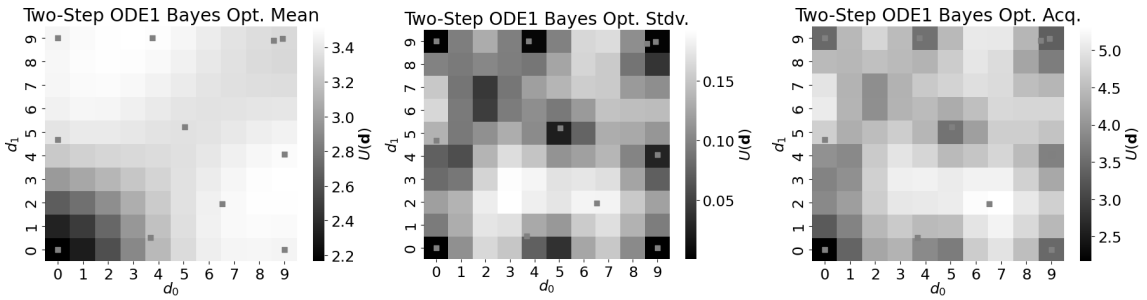

Figure 3.6: 2D Bayesian Optimization of ODE1. Mean (left), standard deviation (center), and acquisition function (right).

are shown in Table 3.4 and Figures 3.6 & 3.7. We look at the results for ODE1 first. The first two points are chosen, and the algorithm decides to test the third point (8.57, 8.88) near the second (8.91, 8.98) perhaps to test the noise level and due to the higher objective. This trend does not continue with the next two points: a higher objective is found at (0.00, 9.00), but the algorithm decides to test the design space at the complete opposite diagonal. Points seven and eight are nearly optimal, but they still do not match the objective of 3.67 as found in discrete batch design. Looking at the associated heatmaps in Figure 3.6, the test points are rather uniform. With more points, we expect the sampling to congregate at the optimal values near (3, 9) and (9, 3). The second plot of Figure 3.6 shows uncertainty; Usually, the regions that are near test points have high uncertainty with one exception. The acquisition function is still favoring exploration, as it nearly mirrors the standard deviation (with respect to relative values).

Next we look at the results for ODE2. The algorithm begins with an identical strategy of choosing a random point and one near the boundary. Due to the exceptionally large increase in objective, the next point chosen is not near the second point and is an exploratory move. Similar to that in ODE1, the algorithm chooses points rather uniformly with extra emphasis on the boundary. In Figures 3.6 and 3.7, the chosen points are at least one in each corner and one along each edge of the boundary. Again, $\mathbf{d}^*$ is not found yet, but the point (4.48, 9.00) is close to the point (6, 9) as found in batch design. Also, the acquisition function favors areas with high uncertainty due to the small number of points being chosen.

	ODE1		ODE2	
#	**d**	$U(\mathbf{d})$	**d**	$U(\mathbf{d})$
1	(3.67, 0.50)	3.19	(3.45, 1.07)	0.88
2	(8.91, 8.98)	3.32	(8.89, 8.93)	1.57
3	(8.57, 8.88)	3.32	(1.57, 5.32)	1.07
4	(0.00, 9.00)	3.46	(9.00, 0.00)	1.27
5	(9.00, 0.00)	3.47	(3.98, 9.00)	1.69
6	(0.00, 4.66)	3.28	(0.00, 0.00)	0.34
7	(9.00, 4.05)	3.51	(9.00, 4.48)	1.68
8	(3.75, 9.00)	3.51	(5.67, 5.26)	1.30
9	(5.05, 5.21)	3.36	(0.00, 3.11)	0.82
10	(0.00, 0.00)	2.17	(6.11, 0.00)	1.06

Table 3.4: Results of two-step Bayesian optimization applied to ODE1 and ODE2.

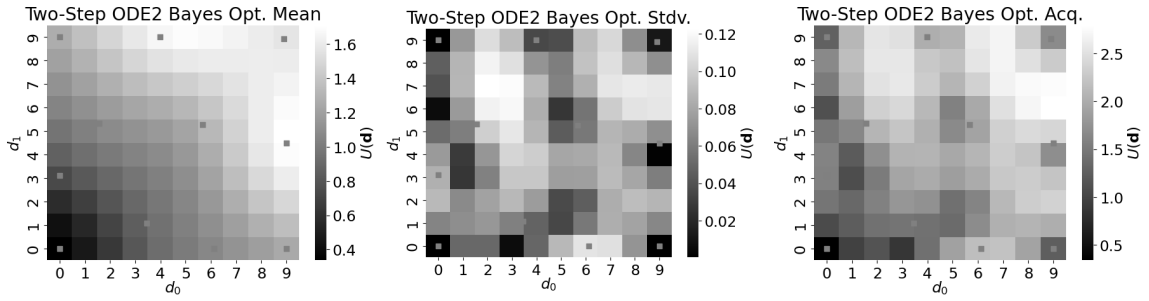

Figure 3.7: 2D Bayesian Optimization of ODE2. Mean (left), standard deviation (center), and acquisition function (right).

3.2.3 Discussion

Bayesian optimization allows for continuous optimization of batch designs. It continues to evaluate the integral in (3.12) to compute expected utility. Unlike batch design, Bayesian optimization is a sequential algorithm that leverages an acquisition function to handle the exploration vs. exploitation dilemma. By considering the most profitable and least certain regions of the design space, the acquisition function naturally moves between choosing more exploratory and more exploitative points. Similar to batch design, Bayesian optimization is only based on granularity of a design sequence $\mathbf{d}$ and does not use information after each experiment, only after computing the entire objective.

Bayesian optimization was able to determine $\mathbf{d}$ that were reasonably close to $\mathbf{d}^*$ found in batch design. The best evidence was that heatmaps of the estimated objectives of Bayesian optimization were nearly identical to those from batch design in the previous section. With more points tested, it is clear that a $\mathbf{d}$ can be found that is at least as good as $\mathbf{d}^*$ found by batch design. While Bayesian optimization and batch design are applied to design sequences, our next algorithm will forgo exploration and only exploit based on the current experiment. It remains to be seen if a view that has finer granularity and incorporates feedback after each experiment will provide a higher objective score.

3.3 Multi-Armed Bandits

Multi-armed bandits, or contextual bandits, involve making a design choice over a set of available alternatives [74, 42, 41]. Each choice only involves the evaluation of a single experiment ($N = 1$), and the set of available alternatives is the set of design choices $\mathscr{D}$. Multi-armed bandits attempt to minimize regret, which is the expected difference between the sum of the optimal reward and realized reward across all of the available alternatives. Multi-armed bandits therefore do not have separate training and testing phases; the associated algorithm is always in the testing phase since regret is used as the objective.

Multi-armed bandits are most similar to one-step batch design. They only have one experiment and over time will be able to approximate the objective of each design, given enough samples. The

two methods are different in the way that rewards are calculated. In batch design, an expected utility is calculated, while in multi-armed bandits, the raw utility function is computed at the end of the episode for the reward. The goal of multi-armed bandits is not to compute the expected utility, but to minimize regret. Multi-armed bandits therefore need to shrewdly trade off exploration and exploitation, as poor design choices are penalized through regret.

We showcase the efficiency of the multi-armed bandit paradigm through four variants that are detailed by Sutton & Barto [82]. The first is optimistic starts, which aims to sample each design many times before settling on the one yielding the most reward. Epsilon greedy design explores on random timesteps and gradually anneals the exploration parameter to converge to the true objective. Upper confidence bound (UCB) provides an exploration bonus at each step based on number of times the design has been taken. Gradient bandit algorithms mirror gradient descent and choose designs randomly through the softmax function. Throughout, we highlight the strengths and weaknesses of these four methods.

3.3.1 Methodology

Multi-armed bandits, or contextual bandits, involve making a design choice with a set of available alternatives. In contrast to maximizing the expected KL-divergence, the bandit problem attempts to minimize regret, which is the expected sum of the differences between the optimal and realized reward. The simplest context of the multi-armed bandit problem is a set of levers called bandits whereby each lever yields a reward from a fixed underlying distribution. The distribution for each bandit is presumed unique.

In the context of RL, the multi-armed bandit problem has a fixed action space with episode length always equal to one. Each reward is a terminal reward that is used to update the estimate, or belief, of each arm. A simple example would be two arms with normally distributed rewards: arm 1 with distribution $\mathcal{N}(\mu_1, \sigma_1^2)$ and arm 2 with distribution $\mathcal{N}(\mu_2, \sigma_2^2)$. With enough pulls of each arm, corresponding to draws from each associated distribution, the estimates for $\hat{\mu}_1$ and $\hat{\mu}_2$ would converge to μ_1 and μ_2 by the Law of Large Numbers. In contrast to the goal of Reinforcement Learning, which is to maximize the expected sum of rewards, we try to minimize regret in the

multi-armed bandit setting as follows:

$$\underset{\pi}{\operatorname{argmin}}\, \mathbb{E}\left[N \cdot R^* - \sum_{i=1}^{N} R_i\right]. \tag{3.22}$$

Here, R^* is the maximum expected reward able to be achieved by a single action and R_i are the realized rewards. Intuitively, regret quantifies the expected reward lost by choosing suboptimal arms under the current policy π. In terms of the arms above, if $\mu_1 < \mu_2$ and arm 1 was chosen n times before arm 2 was chosen indefinitely, then the realized regret would be $n(\mu_2 - \mu_1)$. A notable dilemma is that it is easier to identify arm 2 as better if $\mu_1 << \mu_2$, but then the regret is higher for each pull of arm 1.

Since we are only computing utility (i.e. KL-divergence) and not expected KL-divergence, we now show the relevant computation:

$$u_{D_{KL}}(\mathbf{d},\mathbf{y}) = \int_{\Theta} p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d}) \ln\left[\frac{p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})}{p(\boldsymbol{\theta})}\right] d\boldsymbol{\theta}, \tag{3.23}$$

$$= \int_{\Theta} p(\mathbf{y}|\boldsymbol{\theta},\mathbf{d}) p(\boldsymbol{\theta}) / p(\mathbf{y}|\mathbf{d}) \ln\left[\frac{p(\mathbf{y}|\boldsymbol{\theta},\mathbf{d}) p(\boldsymbol{\theta}) / p(\mathbf{y}|\mathbf{d})}{p(\boldsymbol{\theta})}\right] d\boldsymbol{\theta}, \tag{3.24}$$

$$= \int_{\Theta} p(\mathbf{y}|\boldsymbol{\theta},\mathbf{d}) / p(\mathbf{y}|\mathbf{d}) \ln\left[\frac{p(\mathbf{y}|\boldsymbol{\theta},\mathbf{d})}{p(\mathbf{y}|\mathbf{d})}\right] p(\boldsymbol{\theta})\, d\boldsymbol{\theta}. \tag{3.25}$$

The above sum can be approximated with Monte Carlo sampling as follows:

$$u_{D_{KL}}(\mathbf{d},\mathbf{y}) = \int_{\Theta} p(\mathbf{y}|\boldsymbol{\theta},\mathbf{d}) / p(\mathbf{y}|\mathbf{d}) \ln\left[\frac{p(\mathbf{y}|\boldsymbol{\theta},\mathbf{d})}{p(\mathbf{y}|\mathbf{d})}\right] p(\boldsymbol{\theta})\, d\boldsymbol{\theta}, \tag{3.26}$$

$$= \sum_{i=1}^{m_{out}} p(\mathbf{y}|\boldsymbol{\theta}^{(i)},\mathbf{d}) / p(\mathbf{y}|\mathbf{d}) \ln\left[\frac{p(\mathbf{y}|\boldsymbol{\theta}^{(i)},\mathbf{d})}{p(\mathbf{y}|\mathbf{d})}\right]. \tag{3.27}$$

Therefore, the KL-divergence can be written in terms of the likelihood and evidence. The computation for the evidence term is shown in (3.13).

Next, we introduce some notation that will be useful for practical implementations of these methods. In the multi-armed bandit setting, each bandit (or arm) can be associated with an action and indexed likewise: $a_1, a_2, \ldots, a_M$ for a total of M bandits. Each arm has an associated reward distribution $\mathscr{D}_i$. For sake of simplicity, we assume that $\mathbb{E}[\mathscr{D}_i] = \mu_i$ where $|\mu_i| < \infty$. Denote each

reward from arm i as R_{i_j}. Pulling arm i a total of 5 times in a row would yield rewards $R_{i_1}, R_{i_2}, \ldots, R_{i_5}$.

Since we are trying to determine which arm has greatest expected value, it is helpful to associate a mean estimate $\hat{\mu}_i$ with each arm. Furthermore, multi-armed bandits require incremental updates to the relevant estimates. Denoting

$$\hat{\mu}_{i_k} = \frac{R_{i_1} + R_{i_2} + \ldots + R_{i_k}}{k}, \tag{3.28}$$

we can rewrite this equation to yield an incremental update, as follows:

$$\begin{aligned}
\hat{\mu}_{i_k} &= \frac{1}{k} \sum_{j=1}^{k} R_{i_j} \\
&= \frac{1}{k} \left[R_{i_k} + \sum_{j=1}^{k-1} R_{i_j} \right] \\
&= \frac{1}{k} \left[R_{i_k} + \frac{k-1}{k-1} \sum_{j=1}^{k-1} R_{i_j} \right] \\
&= \frac{1}{k} \left[R_{i_k} + (k-1)\hat{\mu}_{i_{k-1}} \right] \\
&= \frac{1}{k} \left[R_{i_k} + k \cdot \hat{\mu}_{i_{k-1}} - \hat{\mu}_{i_{k-1}} \right] \\
&= \hat{\mu}_{i_{k-1}} + \frac{1}{k} \left[R_{i_k} - \hat{\mu}_{i_{k-1}} \right].
\end{aligned} \tag{3.29}$$

This follows a similar theme in machine learning where the new estimate is the old estimate plus a step size times the difference in the new estimate. To execute this algorithm, we need a total count of the past times each arm was pulled and the current mean estimate to compute the new estimate (i.e. storing the actual reward sequence is unnecessary).

In our implementations below, we will allow 100 arms corresponding to the decimal actions 0.0 through 9.9 with spacing 0.1. Based off the results of one-step batch design, it is guessed that actions 5 and 9 will be chosen most often for ODE1 and ODE2, respectively. Typically the estimated means are initialized to the same value; zero is a good starting point. Had the reward distribution been unknown beforehand, the first few pulls could be used to calibrate the initial values of all arms. We discuss a few of the most common multi-armed bandit algorithms from Sutton and Barto [82], including epsilon greedy, optimistic initial values, upper confidence-bound, and gradient bandit

algorithms. We will be highlighting the strengths and weaknesses throughout. Sample pseudocode for the epsilon-greedy paradigm is shown in Algorithm 7.

3.3.2 Optimistic Initial Values

A naive algorithm is always choosing the best action greedily. This is obviously a poor decision, as this could stifle exploration. To combat this problem, we can choose optimistic initial values, whereby each estimated mean is initialized optimistically to an unrealistic (but still reasonable) reward value. For instance, with rewards less than 4, we may choose to initialize estimated means to 10.

This setup easily encourages exploration, as most rewards pulled will be much less than the associated estimated means. The resultant next estimated mean will be less than the previous. Therefore, all arms will initially be tested once. The best arms will show a smaller decrease in estimated mean than worse arms and will gradually be pulled more often. With enough samples, the bias of the initial values will go to zero. The method can still be suboptimal if the best arm has a few bad pulls; to combat this, a higher initial value will help, but that will in turn increase regret.

The results are shown in Figure 3.8. Since actions are not chosen unless they have the highest estimated mean, the expected utility of actions is poorly approximated, especially for the worst possible actions. In particular, expected utilities are higher than the true values. In both cases, the expected utility decreases sharply for each action, as it tests each algorithm many times. Once it starts to learn the best action, the regret plateaus, showing convergence. While this strategy produced sufficient results for the problem, it is impractical with a large action space and with high noise. The next algorithm we view will guarantee sufficient exploration at all timesteps.

3.3.3 Epsilon Greedy

Epsilon greedy action selection is just as its name suggests: we choose a random action $0 < \varepsilon < 1$ of the time and choose the apparent optimal action $1 - \varepsilon$ percent of the time. Assuming that

$$\mathbb{E}[R \mid A = a^*] = C_1, \quad \textit{and} \quad \mathbb{E}[R \mid A \sim \textit{Random}] = C_2, \tag{3.30}$$

Algorithm 7: Epsilon Greedy Design in Multi-Armed Bandits

Input : Discrete Design Space $\mathscr{D}$
Number of outer integral samples n_{out}
Number of inner integral samples n_{in}
Number of steps N

Initialize *container* of tuples for actions, scores, and counts
Initialize ε to 1
for N **do**
    Draw random number from 0 to 1
    if *random number* < ε **then**
        Set a as random design in $\mathscr{D}$
    else
        Set a as action with best average score
    end

    Perform experiment and receive observations u, v.

    Compute Evidence
    Set *evidence* = 0
    for n_{in} **do**
        Sample $\boldsymbol{\theta}$ from prior
        $evidence \mathrel{+}= p((u,v)|\boldsymbol{\theta}, a)$
    end

    Initialize *total_util* to 0
    for n_{out} **do**
        Sample $\boldsymbol{\theta}$ from prior
        Compute true state and velocity $z, \dot{z}$ (3.3, 3.4, 3.5, 3.6)
        Simulate state and velocity observations u, v (3.8)
        $total\_util \mathrel{+}= \frac{likelihood}{evidence} * \ln\left[\frac{likelihood}{evidence}\right]$
    end
    Increment *total_util* and *counter* for action a in *container*
    Anneal ε
end

Output: Actions and scores; Best Action

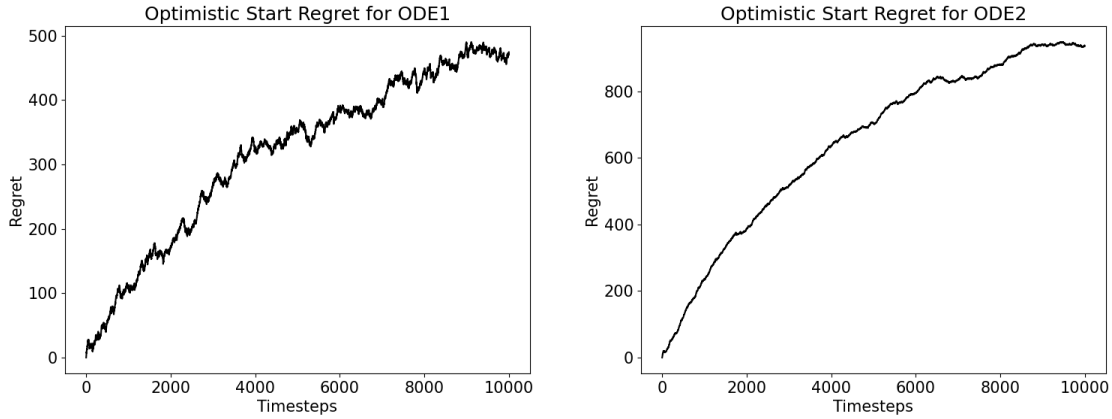

Figure 3.8: Realized regret for optimistic initial values algorithm using ODE1 (left) and ODE2 (right).

then the minimum regret at infinity per timestep is $\varepsilon(C_1 - C_2)$ by choosing the optimal action $1 - \varepsilon$ fraction of the time. Depending on the value of epsilon, this method could be very slow. Too high of an epsilon will provide high regret because the optimal action cannot be selected enough, and too low of an epsilon may prevent the best action from being found quickly. A better strategy is to quickly anneal epsilon from a high number to a low number to find the best candidate arms and then determine the subtle differences between them over time.

Our results are shown in Figure 3.9. The convergence of the epsilon greedy algorithm is shown in the regret. In each case, the regret per timestep was fairly constant and plateaued after reaching zero regret. This shows that the strategy of always allowing exploration may have superior convergence to other methods such as optimistic initial starts. The next algorithm we view will have directed exploration that does not select an action based on randomness but penalizes actions that are chosen often.

3.3.4 Upper Confidence-Bound

Clearly, some of the seemingly suboptimal actions are better than others; epsilon-greedy samples each action equally. A better method would be to sample each action with weights based on the current estimated means. Higher means should require higher weights. As long as care is taken to sample each action infinitely many times, convergence to an optimal policy is guaranteed. Moreover,

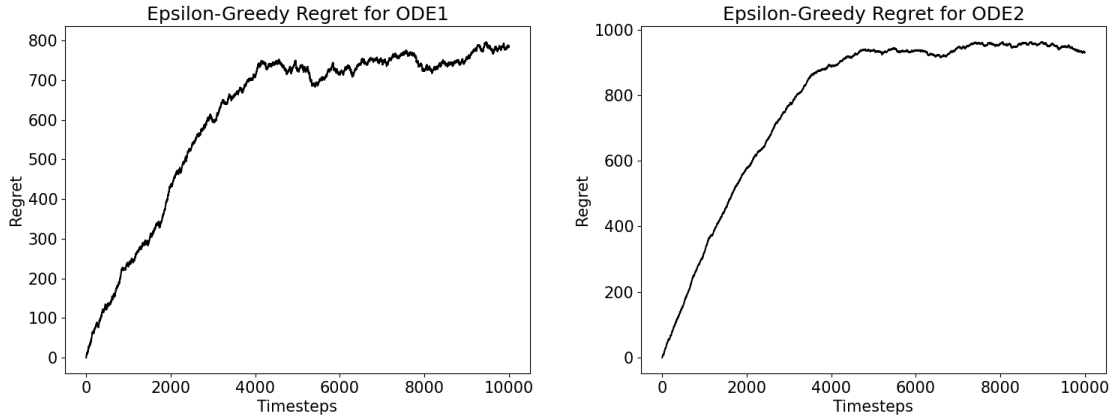

Figure 3.9: Realized regret for epsilon greedy algorithm using ODE1 (left) and ODE2 (right).

we only have point estimates of the means. It would help to incorporate uncertainty in each estimate and sample where there is less certainty. The following action selection rule is taken from Sutton and Barto [82]:

$$A_t \doteq \underset{a}{\text{argmax}} \left[Q_t(a) + c \sqrt{\frac{\ln(t)}{N_t(a)}} \right]. \tag{3.31}$$

Here, $N_t(a)$ is the count of the number of times that a has been previously taken, $Q_t(a)$ is the estimated action-value of a. The second term is used to measure uncertainty. If a is not taken, then $ln(t)$ increases while the denominator $N_t(a)$ does not. The constant c controls for the level of uncertainty. It is important to note that using the natural logarithm allows this growth to be unbounded, so every action will be taken an infinite number of times.

Results are shown in Figure 3.10. For our choice of hyperparameters, there is enough exploration that the regret doesn't fully converge to zero. With more time, we expect that it would. UCB clearly is more adaptive than the epsilon-greedy method through providing some measure of certainty on the estimated means. Also, UCB allows for convergence without annealing hyperparameters. The next algorithm we view does not greedily choose the next most promising point but instead determines a preference for each action and samples based on the preference.

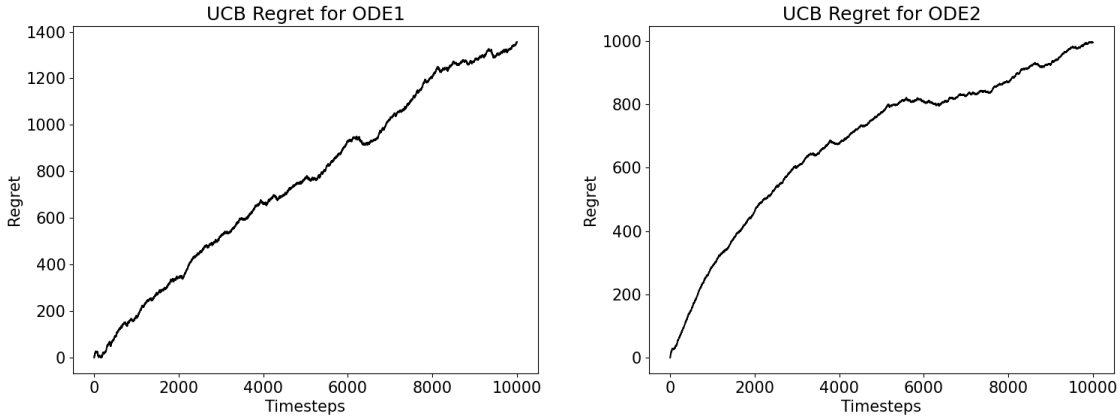

Figure 3.10: Realized Regret for UCB algorithm using ODE1 (left) and ODE2 (right).

3.3.5 Gradient Bandit Algorithms

Gradient bandit algorithms determine a preference for each action in contrast to estimating action values. Preference is modeled to only necessitate relative preference amongst all actions; adding a same preference to each action will not change the probabilities of selecting each action. Given a preference $H_t(a)$, we use the softmax distribution to calculate action probabilities:

$$\pi_t(a) = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}}. \tag{3.32}$$

The softmax distribution is paramount to multi-class deep learning methods, where it usually is used as the output layer. In similar theme, gradient bandit algorithms are trained using stochastic gradient ascent. If R_t are the rewards up to time t, then we denote $\bar{R}_t$ as the average reward up to time t. Gradient bandit increases preferences if the reward is greater than the baseline and decreases otherwise, as follows:

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A)), \text{ and} \tag{3.33}$$

$$H_{t+1}(a) \doteq H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a) \text{ for all } a \neq A_t. \tag{3.34}$$

The results of gradient bandit are shown in Figure 3.11. In the case of ODE1, regret does not seem to converge, but it does in the case of ODE2. This phenomenon may occur due to the smaller

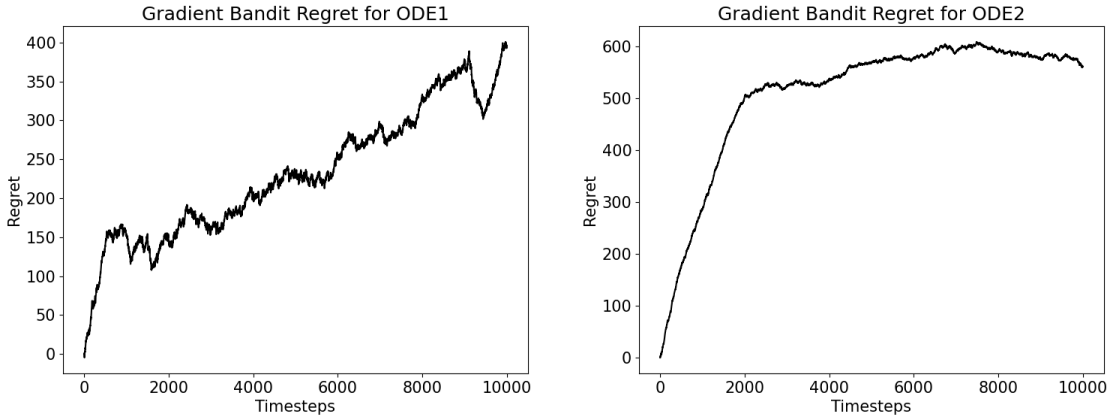

Figure 3.11: Realized Regret for optimistic starts algorithm using ODE1 (left) and ODE2 (right).

differences in the expected utilities for the best actions in ODE1. ODE2 shows a faster convergence but more realized regret, again probably due to the higher disparity of reward per actions. The slight increases in preference relative to the average reward most closely align with the update rules of DRL methods that we investigate last in this chapter.

3.3.6 Summary

Deterministic choice is generally suboptimal. Using optimistic initial values is slow but provides a better chance for optimality. Epsilon greedy methods attempt to mandate exploration to prevent suboptimality. Annealing epsilon slowly enough will guarantee optimality. Noting that some actions are easily seen as bad after a few actions, using an upper confidence bound estimate allows for choosing actions with highest estimated values and less certainty. Gradient Bandit algorithms are most similar to reinforcement learning algorithms, as they train with a baseline.

A comparison of the algorithms is shown in Figure 3.12. In both situations, the gradient bandit algorithm had the least regret and the UCB algorithm had the most regret after $10,000$ timesteps. Also, the epsilon-greedy algorithm generally performed worse than the optimistic starts algorithm. These results indicate that this problem is fairly easy and that only slight exploration is needed. The gradient bandit and optimistic starts algorithms both can quickly converge to a deterministic policy, which forgoes exploration entirely. In contrast, we can tune the hyperparameters for UCB and epsilon-greedy algorithms in future runs to hopefully produce less regret.

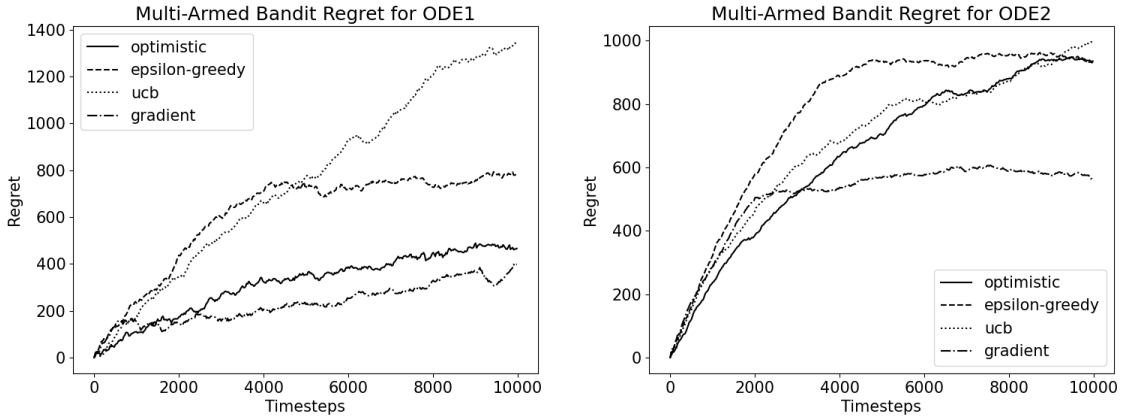

Figure 3.12: Realized regret for all algorithms using ODE1 (left) and ODE2 (right).

The results in this section demonstrate another addition to an objective function: online experimental results. These techniques can be used in situations where the underlying model is not known and the results of experiments must be used while determining the best designs. The notions of incremental updates will be seen shortly when we visit approximate dynamic programming and deep reinforcement learning algorithms.

3.4 Greedy Design

Our next algorithm is discrete greedy design. The goal of greedy design is to maximize a one-step objective after each single experiment without further lookahead [84, 34, 33, 45, 54, 32, 16]. The work of Terejanu et al. [84] closely matches our problem, as they applied Bayesian greedy experimental design to learn the parameters of a graphite nitridation experiment with known underlying model. Each set of experiments was conducted with a fixed set of model parameters $\boldsymbol{\theta}$. We extend these results by allowing $\boldsymbol{\theta}$ to change across sets of experiments, where $\boldsymbol{\theta}$ can be any value in the prior. We elaborate about this difference in the methodology section.

The algorithms we consider can incorporate both multi-step lookahead and also feedback. Greedy design is in stark contrast to batch design, as greedy design incorporates feedback but no multi-step lookahead, while batch design incorporates multi-step lookahead but no feedback. After each experiment in greedy design, the posterior belief state over $\boldsymbol{\theta}$ is calculated and used to determine the

next design. By leveraging a posterior, we are able to determine the probability of possible next states, which is helpful in choosing the best design. A central problem to this approach when comparing to other algorithms, namely batch, is that expected KL-divergence is not necessarily additive. We introduce a result by Shen and Huan [79] that shows expected KL-divergence is additive in our formulation of the sOED problem.

Our results show that greedy design is inferior to batch design for ODE1 and superior to batch design for ODE2. For ODE1, the first action is seemingly suboptimal, resulting in compounding suboptimality after the second experiment. Further sections will combine the multi-step lookahead of batch and the feedback of greedy design to guarantee optimality.

3.4.1 Methodology

Greedy design at experiment k begins with a known prior on our parameters p_k, a design space $\mathscr{D}$, and a known underlying model relating our model parameters and designs to observations (3.8). The objective for greedy design is as follows:

$$d_k^* = \arg\max_{d_k \in \mathscr{D}} U(d_k), \qquad \boldsymbol{\theta} \sim p_k. \tag{3.35}$$

Since the goal of greedy design is to maximize the expected utility over all possible designs, we first demonstrate how to compute the expected utility over a single design. Our experiments are of length one, so we compute the integral below in the same manner as one-step batch design. The one step batch design is taken from (3.12) as

$$U(d) = \int_{\mathscr{Y}} \int_{\Theta} \{\ln[p(y|\boldsymbol{\theta},d)] - \ln[p(y|d)]\} p(y|\boldsymbol{\theta},d) p(\boldsymbol{\theta}) d\theta dy. \tag{3.36}$$

The computation of this integral was discussed in (3.13) and (3.14). Terejanu et al. [84] allowed for designs in the discrete design space $\mathscr{D} = \{1,2,3,\ldots,10\}$, and we allow for designs in $\mathscr{D} = \{0,1,2,\ldots,9\}$. The best design $d_0 \in \mathscr{D}$ is found by performing discrete optimization in the design space above. Then, we sample an observation y_0 by using our forward model. Terejanu et al. assume a fixed underlying $\boldsymbol{\theta}$ throughout all of the experiments, and we allow $\boldsymbol{\theta}$ to change when a new set

of experiments is conducted. This $\boldsymbol{\theta}$ is known to the model but unknown to the experimenter. Using Bayes' Theorem, we compute the posterior $p(\boldsymbol{\theta}|y_0, d_0)$ which is used as the prior for the next experiment.

The above process repeats with the new posterior being used in place of the prior. In this manner, design d_k is chosen by performing one-step batch design using the posterior $p(\boldsymbol{\theta}|I_k)$. At each step, the objective of a single experiment is maximized, given the current posterior. The total number of experiments can be decided through a utility threshold, capped at a set number or determined by another method. Terejanu et al. allowed for 10 experiments and explored various strategies of allowing and disallowing repeated measurements.

Since the discrete one-step batch case determined that the design $d_0 = 5$ yielded the most utility after one experiment for ODE1 and $d_0 = 9$ for ODE2, we fix those designs for d_0 in each case. We must compute a Bayesian update and start our next experiment using the posterior. Specifically, we must sample from this new posterior and compute the next objective for each valid action. Through this Bayesian perspective, we introduce feedback that can be used to maximize the objective given these intermediate states.

Greedy design starts with a prior and determines the objective of each action through calculating the integral in (3.36). To compare actions equally, we use the same samples of $\boldsymbol{\theta}$ across each action when performing the Monte Carlo sampling. All of the previous setup is identical to batch design, aside from reusing the parameter samples. The primary difference is adopting a Bayesian viewpoint for the prior across experiments. In batch design, the Bayesian viewpoint was used to calculate the objective, but it was not used to generate samples.

Here, we fortunately can use a 2-dimensional grid to discretize the posterior. We create a 100×100 rectangular grid of sub-rectangles. Each sub-rectangle is represented by its center: we approximate the probability of the true $\boldsymbol{\theta}$ lying in that sub-rectangle by the probability equaling $\boldsymbol{\theta}$ at the center. In this fashion, we have 10,000 possible values of $\boldsymbol{\theta}$ to choose by the grid. Had the dimensionality of $\boldsymbol{\theta}$ been much greater, we would need to resort to other methods such as Markov Chain Monte Carlo (MCMC) to sample from the posterior.

The probability of each grid point is computed using Bayes' rule. We may disregard the evi-

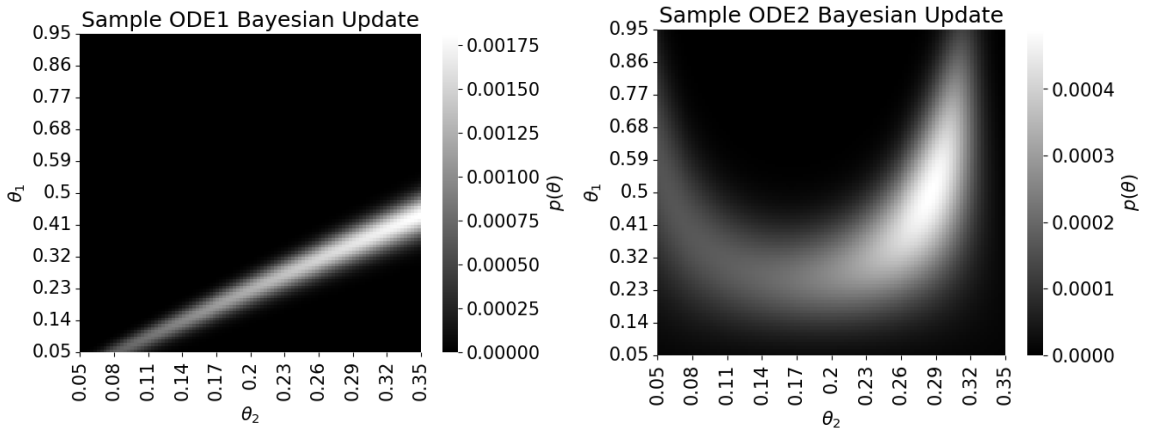

Figure 3.13: Sample Bayesian update grid for ODE1 (left) and ODE2 (right).

dence term and simply multiply the prior times the likelihood to create an unnormalized probability distribution over the grid points. Conveniently, we divide by the sum of the unnormalized proababilities to get true probabilities. We zero out probabilities that are low to prevent underflow. The result is the next posterior. Two sample updates are shown in Figure 3.13, and the pseudocode for an update is shown in Algorithm 8.

The goal of this entire work is to compare various methods across the same problem. Our utility function is KL-divergence, which is not necessarily additive. This presents a problem, as greedy design has two separate KL-divergences, going from the prior P to the first posterior Q_1, namely $D_{KL}(Q_1||P)$, and it also has the KL divergence from first posterior Q_1 to final posterior Q_2, namely $D_{KL}(Q_2||Q_1)$. Since $D_{KL}(Q_2||P)$ does not necessarily equal $D_{KL}(Q_1||P) + D_{KL}(Q_2||Q_1)$, then we need to determine a different way to compare these algorithms. Fortunately, Shen and Huan [79] show that under our Bayesian sOED framework that expected KL-divergence is additive under a fixed design policy. Therefore, we may compare results regarding expected KL-divergence to those of batch design. This allows us to simply add the individual KL-divergence scores across each experiment to form a sum of KL-divergences per episode. After enough sets of experiments have been conducted, we those sums should converge to the expected KL-divergence found by batch design. We ran 250 sets of experiments and the pseudocode is shown in Algorithm 8.

Algorithm 8: Bayesian Grid Update

Input : prior probability distribution P
state observation u
velocity observation v
design choice t
probability threshold T
list of (θ_1, θ_2) grid points

Initialize posterior distribution Q for posterior probabilities.
for (θ_1, θ_2) *pair in grid* **do**
    Compute actual state z using (θ_1, θ_2, t)
    Compute state likelihood using (z, u) and noise model
    Compute actual velocity $\dot{z}$ using (θ_1, θ_2, t)
    Compute velocity likelihood using $(\dot{z}, u)$ and noise model.
    Set $Q(\theta_1, \theta_2)$ = *state_likelihood* $\cdot$ *velocity_likelihood* $\cdot P(\theta_1, \theta_2)$
    if $Q(\theta_1, \theta_2) < T$ **then**
        set $Q(\theta_1, \theta_2) = 0$
    end
end
$Q \mathrel{/}= sum(Q)$

Output: Posterior grid Q

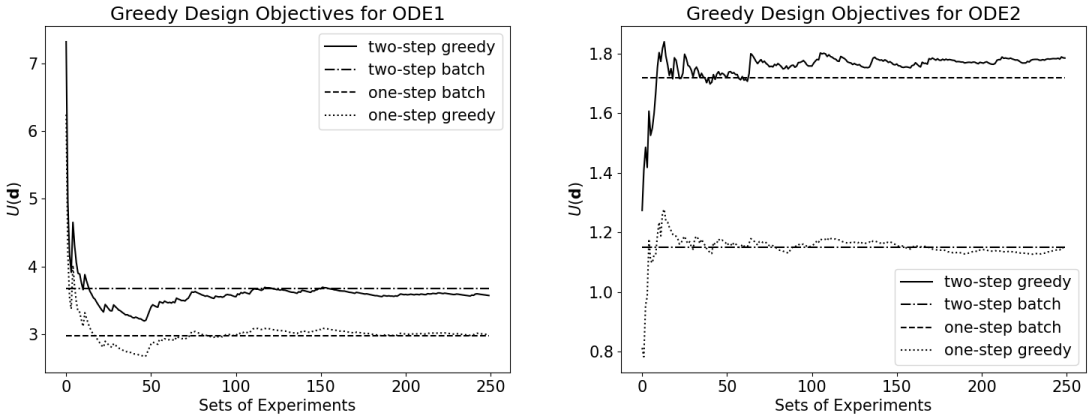

Figure 3.14: Numerical results of two-step greedy design applied for ODE1 (left) and ODE2 (right). Horizontal lines signify results from batch design for comparison.

Algorithm 9: Greedy Design with Double Integral Sampling

Input : Discrete Design Space $\mathscr{D}$
Prior distribution P Number of Experiments T
Number of outer integral samples n_{out}
Number of inner integral samples n_{in}

Initialize *list* for best designs. Sample $\boldsymbol{\theta}$ from prior
for k *in* $\{0, 1, \ldots, T-1\}$ **do**
    Initialize *container* of tuples for designs and scores
    for d *in* $\mathscr{D}$ **do**
        Calculate *objective* using Algorithm 6 and current prior
        Append (*objective*, d) to *container*
    end
    Determine best design d_k from *container*
    Append best design to *list*
    Sample observation using $\boldsymbol{\theta}$
    Construct posterior using Algorithm 8
end

Output: Best Greedy Design at each Experiment, namely *list*

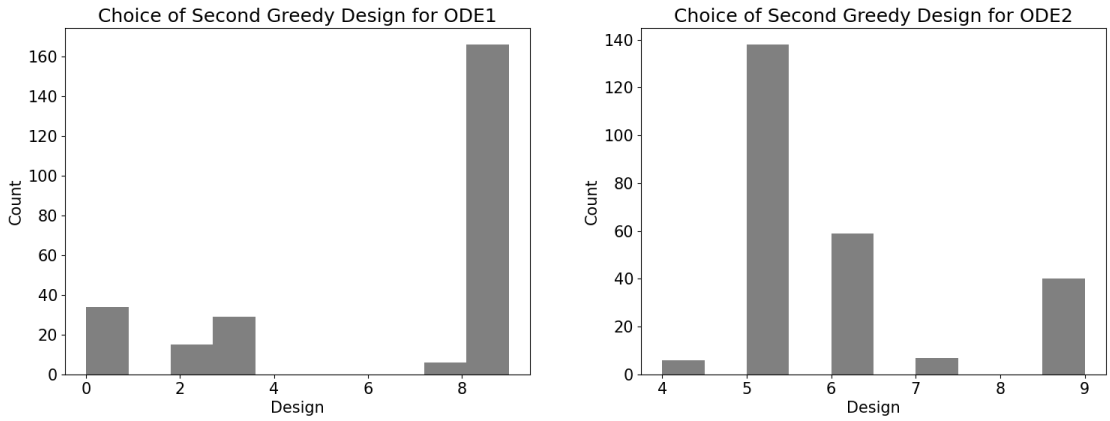

Figure 3.15: Counts of second design of greedy design with backwards view of ODE1 (left) and ODE2 (right).

3.4.2 Results

The results are shown in Figures 3.14 and 3.15. For ODE2, the best first design was $d_0 = 9$. If we adopt a forward view, we limit the design space for action two to set of size 1, namely $\mathscr{D}_1 = \{9\}$. This trivializes the problem, so we instead adopt a backwards view in which the algorithm can choose any action from the original design space for the first action. In effect, we are relaxing the constraints on the design space and will not be decreasing expected utility.

For ODE1, the expected utility after two steps was 3.57, which is 0.10 less than the utility of 3.67 that is accrued with batch design with $\mathbf{d}^* = (3,9)$. The objective of the first action $d_0 = 5$ is 2.98, which is equivalent to the one-step batch objective of $d_0 = 3$. The first action $d_0 = 5$ neglects lookahead and is only optimal in one-step greedy design. Therefore, in this case, the knowledge of the belief state gained through feedback in greedy design does not make up for the lost multi-step lookahead of batch design.

For ODE2, the utility was 1.78, which is greater than the utility of 1.72 that was attained with the batch design $\mathbf{d}^* = (6,9)$. This increase is expected, as greedy design chooses $d_0 = 9$ and may choose $d_1 = 6$ to match batch design, or it may choose a seemingly better action based on its lookahead simulations. Furthermore, one-step greedy design matches one-step batch at an objective score of 1.15, furthering our confidence in these results.

We display counts of d_1 in Figure 3.15. ODE1 generally chooses 9 as the next best action, and it never chooses the same action $d_0 = 5$ twice. ODE2 generally chooses a smaller number as its next action, but sometimes chooses $d_1 = 9$. The count results for ODE2 agree with the two-step batch case, as the best $\mathbf{d}$ were $(5,9)$ and $(6,9)$.

3.4.3 Discussion

Greedy design allows for an analysis of the future objective at the granularity of an experiment through one-step lookahead. Unlike batch design, greedy design actively incorporates feedback into its decisions. By considering the current belief on the parameters and simulating many trajectories, Greedy design can effectively estimate the objective at the next experiment. Notably, this viewpoint is shortsighted and neglects multi-step lookahead.

Since we needed samples from a posterior, we discretized a 2-dimensional grid and computed the posterior at each grid point. It allows for fast sampling after the initial posterior construction. We also invoked a result by Shen and Huan [79] to compare greedy design against other algorithms by showing how expected KL-divergence is additive under a fixed policy. Using this fact, we can sum the individual KL-divergences by greedy design across all sets of experiments to approximate the expected KL-divergence of batch design. The remaining algorithms in this chapter will rely on intermediate rewards, as they also analyze the future objective at the granularity of an experiment.

Greedy design with backwards view performs worse than batch design for ODE1 and better for ODE2. The action choice of $d_0 = 5$ is clearly suboptimal for ODE1, and the algorithm cannot recover the lost expected utility through use of past information. Greedy design chooses $d_0 = 9$ for ODE2 and can either match $\mathbf{d}^* = (6,9)$ from batch design with $d_1 = 6$ or choose a better action through one-step lookahead. In the coming sections, we will combine feedback of greedy design and multi-step lookahead of batch design to gradually learn an optimal policy.

3.5 ADP

Our next algorithm is approximate dynamic programming (ADP) [52]. In this implementation ADP, we leverage a state-value function and a lookahead model to best choose the current design. The goal of ADP is to navigate the exploration vs exploitation dilemma to gradually learn the value function of the optimal policy. Using this value function, we can look one step ahead and determine which designs will lead to the most valuable states. Since we have two experiments, there is a start state, the states after one experiment, and states after two experiments. The states after two experiments are terminal states. The value of being in those states is equal to the KL-divergence from the prior to the corresponding terminal state; as a result, we do not need value functions for those states. Also, the start state does not need a value function because we only use the value function of future states during lookahead. This leaves us with states after one experiment needing a value function. We construct a value function based on the design taken, and the two observations received, namely (d_0, u_0, v_0). The main challenge of ADP is learning the value function quickly and accurately.

ADP is most similar to greedy design, as lookahead is incorporated through the model dynamics. Whereas greedy design uses immediate KL-divergence to gauge goodness of design, ADP uses the value function as goodness of next state, which can be used to determine the best design to take. Greedy design works without training, as it calculates an integral (3.36) at each experiment using the current prior. The integral is costly to evaluate and does not allow for multi-step lookahead, only feedback. In contrast, ADP learns a value function for each state which has similar cost but allows for multi-step lookahead and has optimality guarantees in certain settings. The value function is learned through policy iteration, which is an iterative procedure that combines policy evaluation and policy improvement in a single step to gradually improve the policy. As in classical dynamic programming, the optimal policy satisfies

$$v_{\pi^*}(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + v(s')], \tag{3.37}$$

and the update rule for backing up the terminal reward signal is

$$v_\pi(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r + v(s')]. \tag{3.38}$$

Our results show that ADP surpasses batch and greedy design by attaining a greater expected utility. In each case, the value function gradually increased from initial values of 0 to values higher than that of batch and greedy at the start state. By combining lookahead and feedback, ADP can look ahead into the future and prevent suboptimal actions from being taken while also using all information given from the current prior. The final section in deep reinforcement learning will show how the dynamics of the lookahead model can be learned, albeit with loss of convergence guarantees.

3.5.1 Methodology

ADP has the four following steps:

1. *Explore*: Our goal is to compute the value function of any state that can be encountered in an experiment. We first run trajectories with random designs to determine a set of states. A tra-

jectory is formed by first sampling an underlying $\boldsymbol{\theta}$. Each step of the trajectory is computed by choosing a design d and computing an observation (u, v) via (3.8). The entire trajectory is formed by simulating N steps. It is with these states that we train a regressor to approximate the value function. We run 1000 random trajectories and store the state after the first experiment, which includes the first design and the state and velocity observations (d_0, u_0, v_0).

2. *Score States*: We cascade backward and determine the exact value function of each one of the 1000 states. Each state is the posterior after one experiment, encoded by d_0, u_0, and v_0; the reward is the KL-divergence at the end of two experiments (3.36). Therefore, at a single state, we look one step into the future and determine the highest KL-divergence among all actions. In effect, we run the greedy algorithm (Algorithm 9) from each one of the 1000 states to determine the individual state-values. Note that this requires sampling from the posterior (see Algorithm 8).

3. *Train Regressor*: Now that we have inputs (states from step 1) and outputs (scores from step 2), we may train a regressor function. In practice, we used a k-nearest neighbors function with number of neighbors equal to 3. Since each input/output pair was exact, we do not need to regularize the regressor with many neighbors, although it is necessary if the number of samples is too small.

4. *Evaluate*: From the start state of a uniform prior, we run many trajectories that attempt to maximize expected KL-divergence. Each trajectory tests designs in the set $\mathscr{D} = \{0, 1, 2, \ldots, 9\}$ and looks one state ahead. Instead of using the immediate KL-divergence to gauge which action is best, we use the learned value function from our trained regressor. Again, we choose the action with highest expected value. This process is identical to greedy design (Algorithm 9) using the value functions instead of the KL-divergence. Using the value function instead of the immediate KL-divergence (such as in greedy design) allows for lookahead due to the value function incorporating multi-step lookahead to the terminal state and prevents shortsightedness.

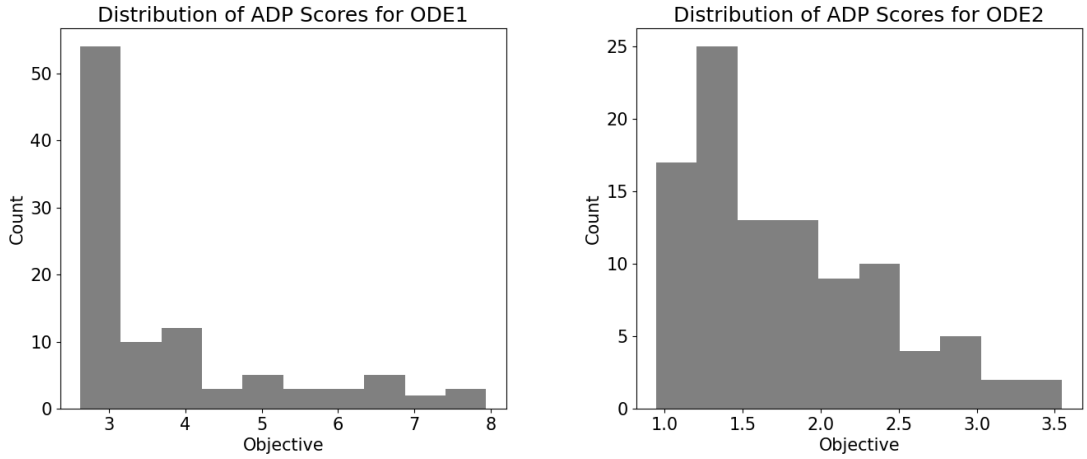

Figure 3.16: Testing scores of ADP for ODE1 (left) and ODE2 (right).

3.5.2 Results

The results of ADP applied to ODE1 and ODE2 with both state and velocity data are shown in Figure 3.16. The average score of ODE1 was 3.69 and that of ODE2 was 1.79. These scores were slightly better than those of batch and greedy design. In each case, there is a relatively broad range of scores, but the mean score was in both cases was near that of batch and greedy design. For ODE1, ADP contended with batch and was superior to greedy design. Therefore, multi-step lookahead may be more important that feedback for ODE1. For ODE2, ADP and greedy design were better than discrete and continuous batch design, which may indicate that feedback is more important than multi-step lookahead in this case. To gauge the performance of another algorithm with multi-step lookahead and greedy design, we analyze these phenomena in the next section.

3.5.3 Discussion

Approximate dynamic programming incorporates both multi-step lookahead and feedback through backing up the reward signal from the terminal states to previous states, all the way to the start state. Since past and future effects are taken into consideration when choosing a design, ADP can be optimal when enough samples and trajectories are taken and explored. ADP leverages a value-based approach to learning the value of a one-step lookahead for each state in a random set of trajectories. A downside of ADP is the lack of interaction with the environment. When the initial trajectories

were generated, we did not generate any more. Huan and Marzouk [52] navigate the exploration vs. exploitation dilemma by generating greedy and exploration trajectories after training. In the next section, we navigate the dilemma at each timestep rather than after fitting many functions.

The results of ADP were no worse than batch and greedy design. The first actions of ADP agreed with the designs from batch design, namely $d_0 = 3$ for ODE1 and $d_0 = 9$ for ODE2. Therefore, a better action is chosen for ODE1 initially than in greedy design, and in both cases the second action can match that of batch design or change based on the current state. For the results so far in the chapter, ODE1 seems to benefit more from multi-step lookahead and ODE2 from feedback. The algorithm used in the next section will incorporate both multi-step lookahead and feedback in an online fashion.

3.6 Deep Reinforcement Learning

Our last algorithm in this chapter employs deep reinforcement learning techniques. Reinforcement learning aims to find an optimal policy through interaction with an environment through exchanging state, action, and reward signals. Similar to ADP, we have states based on the designs chosen and observations received. Then, we may choose actions from our design space $\mathscr{D}$ at each timestep to receive a new observation which determines the next state. Rewards are given at the end of the episode ($N = 2$ timesteps here) based on the KL-divergence at the end of the episode from final terminal state to the start state. We view the reward function as a black box where the environment knows the current belief state; we ourselves never compute Bayesian updates and only use the three signals provided. It is possible to compute Bayesian updates ourselves and use a model-based approach, but instead we use a model-free approach. DRL has currently provided algorithms with state-of-the-art results on a variety of goal-directed tasks [67, 88, 80, 85], but the power of DRL comes with the lack of convergence guarantees.

DRL is similar to ADP in the previous section, as they both attempt to learn a value function on the states. The backup equation is identical to that in the previous section as

$$Q(S_t, A_t) \leftarrow R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t). \tag{3.39}$$

The primary difference between ADP and DRL is that DRL is an online learning approach while ADP can easily be completed in an offline setting, as we have done here. For ADP, we specified how many trajectories would be used to collect data and trained regressors solely on those trajectories. In our chosen DRL algorithm of DQN [67], we continually gather data until the algorithm converges. ADP algorithms can surely train multiple times; for example, Huan and Marzouk [52] retrain their regressors after a few hundred trajectories. However, DRL algorithms typically train after a single trajectory, and often after a single timestep (e.g. DQN). Our results show that DRL can successfully recover an optimal policy in roughly 15,000 timesteps. DRL is able to leverage the state, action, and reward signals to train a policy incrementally.

3.6.1 Methodology

We follow the methodology of Shen and Huan [79]. Our states are formed by the designs chosen so far by the experimenter and the observations. We encode the state as a length 9 vector. The first 3 components are analogous to a one-hot vector where component i is active (equal to 1) if it is currently timestep i. The remaining components encode d_0, u_0, v_0 and the last three components encode d_1, u_1, and v_1. The state vector has form

$$s = [i_0, i_1, i_2, d_0, u_0, v_0, d_1, u_1, v_1]. \tag{3.40}$$

The actions are equivalent to designs in this problem, where actions can be chosen from $\mathscr{D} = \{0, 1, 2, \ldots, 9\}$. The rewards are equal to terminal KL-divergence after the final timestep (3.36). We compute the KL-divergence through a single sample rather than an expectation, which introduces a great amount of stochasticity into the final reward signal. This requires generalization on behalf of the neural network, and we decide to limit the network to only 2 hidden layers to prevent overfitting. The algorithm mirrors that of Chapter 2 (Algorithm 5).

One notable change in our methodology as compared to Shen and Huan [79] is the implementation of DQN with the OpenAI agent-environment interaction paradigm [19]. By transforming the Bayesian sOED problem into a RL problem that adheres to the OpenAI standard for environments, we can use off-the-shelf [58, 47] implementations of DRL algorithms that can be notoriously

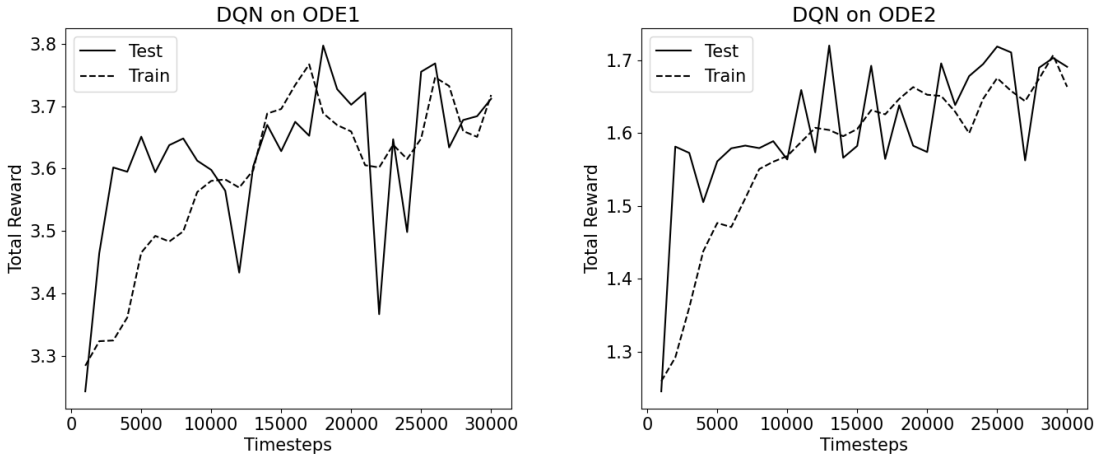

Figure 3.17: DRL training and testing data ODE1 (left) and ODE2 (right).

tricky to implement ourselves. This provides other researchers the ability to tackle Bayesian sOED problems with DRL methods while only having to design the environment.

3.6.2 Results

The results of DQN applied to ODE1 and ODE2 are shown in Figure 3.17. Looking at the results for ODE1, we see that the policy reach an objective score of 3.7 after roughly 15000 timesteps, which equates to 7500 episodes. The best objective score of batch design was 3.67. Generally DQN had scores around 3.7 past 15000 timesteps. Looking at the actions chosen by DQN, action 3 was chosen roughly 50% of the time and the same for action 9. However, due to the continual updates and exploration, other actions were chosen a non-negligible percentage of the time, around 5%.

The testing results for ODE2 are slighly lower than two-step batch design. The testing results fluctuate from an objective score of 1.6 to 1.7, and two-step batch has highest objective score of 1.72. The results reach a score of 1.6 after 12000 timesteps, corresponding to 6000 episodes. Unlike the results for ODE1, the training dyamics continue a steady rise past the 10000 timestep mark when epsilon-exploration was annealed to 1%.

3.6.3 Discussion

DRL methods perform updates on the individual level of a state and usually after each action is taken. Through interacting with the environment, DRL methods attempt to iteratively learn an optimal policy through the state, action, and reward signals. DQN is a model-free policy that doesn't take into account the underlying mathematical model. It learns an action-value for each action that determines the goodness of taking the action in the current state. Evaluating all possible action-values for a given state finds the best action. DRL methods back up information from terminal states to start states to iteratively learn better policies and attempt to achieve optimality.

The highlight of our results was the speed: in fewer than 15000 experiments, the policies were already near-optimal, and in the case of ODE1, already better than the objective score found from batch design. If we annealed epsilon even more quickly, we expect the policies to initially improve more quickly as well. However, the appeal of DRL is the possible optimality, and we are more interested in improving the objective score.

The results of DQN found good policies in general, but they never converged. There are myriad reasons for this lack of convergence, including a non-zero learning rate, not annealing epsilon to zero, not having a flexible enough neural network, and most likely, not using a more powerful reinforcement learning algorithm. DQN is possibly the most fundamental DRL algorithm, and it can suffer from poor convergence and poor predictive power. For example, we could leverage an algorithm incorporating one-step model-based lookahead since we are assuming knowledge of the underlying model. Some of the most well-known DRL algorithms for board games [67, 88, 80, 85] employ model-based techniques, and we see these algorithms as a promising next step to this work.

3.7 Conclusion

Through viewing the expected information gain as a double integral over observations and parameters, we were able to successfully apply Monte Carlo sampling and evaluate the objective for batch design. Continuous batch design leverages an acquisition function that tests designs based on estimated objective and uncertainty. If the results of all experiments need to be used, then multi-armed

bandits provide a one-step framework for minimizing the regret of choosing suboptimal actions. Greedy design incorporated past feedback through sampling from the latest posterior distribution over the parameters. ADP and DRL use the RL formulation of the Bayesian sOED problem to back up the reward from the terminal states to the start states, enabling multi-step lookahead and feedback. ADP uses the underlying model to guide the choice of design, while DRL is model free and learns solely through the agent-environment interaction. In the next chapter, we do not allow velocity data to be observed, as it is not as practical of an assumption as observations on the state data. Therefore, every design produces half the amount of data, and we analyze the anticipated decrease in information gain.

Chapter 4

Optimal Experimental Design in Nonlinear Parameter Estimation with only State Data

In the previous chapter, we assumed both state *and* velocity data were observed. Here, we only assume state data. Our analysis this chapter will be performed again on ODE1 (3.1) and ODE2 (3.2), and we will continue our goal of Bayesian parameter inference. Since only state data is allowed in this chapter, we are unable to exploit the linearity of ODE1 and perform linear regression for parameter inference as in Chapter 2, mandating approaches shown in Chapter 3.

We assume the same experimental setup: parameters $\boldsymbol{\theta} = (\theta_1, \theta_2)$ are drawn from a joint uniform distribution $\mathscr{U} = \mathscr{U}_1 \times \mathscr{U}_2 = (0.05, 0.95) \times (0.05, 0.35)$. There are $N = 2$ experiments with designs chosen from $\mathscr{D} = \{0, 1, 2, \ldots, 9\}$. Observations y_k are corrupted with normal noise $\varepsilon \sim \mathscr{N}(0, \sigma_u)$ where $\sigma_u = 0.1$.

This chapter mirrors the previous, as we apply most of the same algorithms to recover $\boldsymbol{\theta}$ with as much certainty as possible. We disregard the multi-armed bandits and ADP algorithms, as they mirror most of the same goals as Bayesian optimization and DRL, respectively. We aim to compare ODE1 against ODE2 and also the results of this chapter versus the previous. For instance, the boundary conditions of the state on the ODEs are given, so a design of 0 will yield no new infor-

mation. Other challenges arise, such as the greater number of unknown parameters than data points after a single experiment, yielding an underdetermined system. This chapter provides comparisons for parameter estimation of ODEs with the more practical assumption of only state data.

4.1 Batch Design

We analyze the Bayesian sOED problem on ODE1 and ODE2 first with batch design [35, 4] to yield another baseline. Since we are only allowed state observations and our ODEs both contain two unknown parameters, the data from a single experiment induces an underdetermined system. Therefore, one-step batch design needs two observation-design pairs to produce a determined system. Similarly, two-step batch design will allow for three design choices.

Batch design without velocity data follows the same framework as that in Chapter 3, in that computing the likelihood is central to evaluating the objective of (3.12). The one change is that

$$p(\mathbf{y}|\boldsymbol{\theta},\mathbf{d}) = p(\mathbf{u}|\boldsymbol{\theta},\mathbf{d}) \tag{4.1}$$

instead of including the velocity term as in (3.15). The likelihood of a number of observations is transformed into a product of individual likelihoods as in (3.16). The pseudocode from Algorithm 6 can be adapted to this problem by removing the observations of $\mathbf{u}$ and adjusting the likelihood term as explained above.

We begin by performing one-step batch and two-step batch with $\mathscr{D} = \{0,1,2,\ldots,9\}$. One-step batch produces the best greedy action for use in Section 4.3. Throughout, we compare and contrast the results of ODE1 and ODE2 against each other and against the results of Chapter 3.

4.1.1 Results & Discussion

The results of one-step batch applied to ODE1 and ODE2 are shown in Figure 4.1 and Table 4.1. Looking at the heatplots, batch design has optima when a medium design (e.g. 3, 4, 5, 6) is chosen and also the largest design is chosen, namely 9. Comparing these results to those of Chapter 3, we see that they are virtually equivalent. Likewise, the highest objectives were found generally when

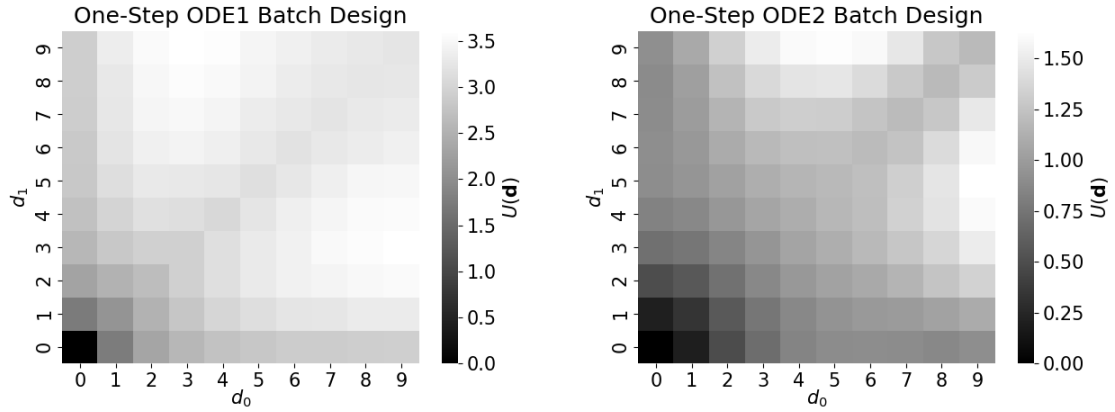

Figure 4.1: Graph of one-step batch design for ODE1 (left) and ODE2 (right). Only knowledge of the state observations is assumed.

ODE1		ODE2	
d	$U(\mathbf{d})$	**d**	$U(\mathbf{d})$
(3, 9)	3.61	(5, 9)	1.62
(4, 9)	3.56	(4, 9)	1.60
(3, 8)	3.56	(6, 9)	1.59
(2, 9)	3.53	(3, 9)	1.51
(4, 8)	3.53	(7, 9)	1.48
(3, 7)	3.52	(5, 8)	1.47
⋮	⋮	⋮	⋮
(1, 1)	2.09	(1, 1)	0.34
(0, 1)	1.75	(0, 1)	0.20
(0, 0)	0.00	(0, 0)	0.00

Table 4.1: Results of one-step batch design where only state knowledge is assumed. Designs are ranked from best (top) to worst (bottom).

larger designs were chosen.

Looking at Table 4.1, we see that $\mathbf{d}^* = (3,9)$ for ODE1 and $\mathbf{d}^* = (5,9)$ for ODE2. Comparing to the results in 3, we see that $\mathbf{d}^*$ is the same for ODE1 and nearly equivalent for ODE2, as there it was $\mathbf{d}^* = (6,9)$. Furthermore, the optimal objective score only decreased by 0.06 from 3.67 to 3.61 for ODE1 and by 0.10 from 1.72 to 1.62 for ODE2. This suggests that the velocity data may be uninformative, or that the state and velocity data provide similar information. Since the boundary conditions are known for state data, a design of 0 provides no information, as reflected by an objective score of 0.

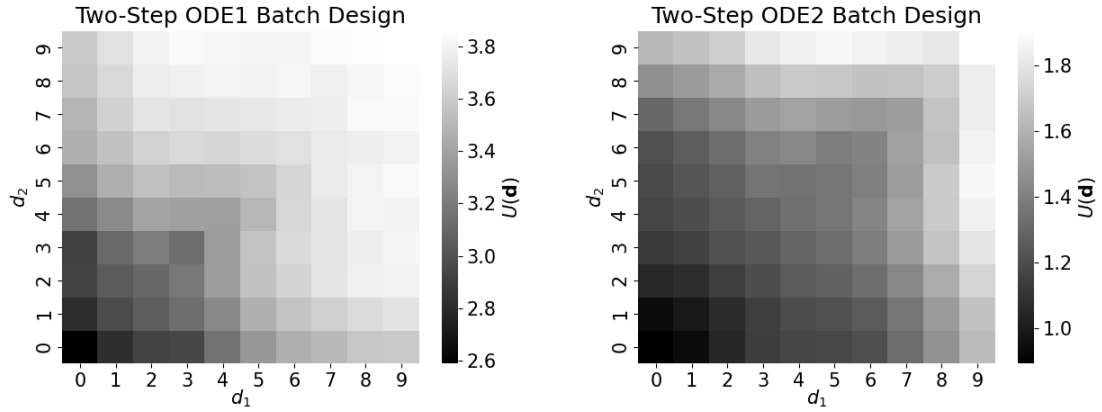

Figure 4.2: Graph of two-step batch design for ODE1 (left) and ODE2 (right). Only knowledge of the state observations is assumed. Results assume $d_0 = 3$ for ODE1 and $d_0 = 5$ for ODE2.

Next, we study the case of two-step batch design with three designs. The results are displayed in Figure 4.2 and Table 4.2. The heatplots do not show all data, as they are conditioned on a choice of d_0. For ODE1, conditioning on $d_0 = 3$ yields most reward for larger (d_1, d_2) without much exception. The boundaries at $d_1 = 9$ and $d_2 = 9$ have the highest objective scores, and indeed $\mathbf{d}^* = (3,9,9)$ for ODE1. The same trend followed for ODE2 when we condition $d_0 = 5$. The larger designs received the highest objective scores, and $\mathbf{d}^* = (5,9,9)$.

Looking at Table 4.2, we see that the one-step optimal batch designs are subsets of the two-step optimal batch designs. For both ODE1 and ODE2, $\mathbf{d}^*$ tested the design 9 twice. The objective score increased from 3.61 to 3.89 for ODE1 and from 1.62 to 1.90 for ODE2. Therefore, significantly more information was learned after testing a design twice to the point that it was optimal for two-step design. In each case, the two-step design in this chapter had greater objective score than the two-step design in Chapter 3. This is in spite of two-step batch design in Chapter 3 taking four measurements (two state measurements and two velocity measurements) while two-step batch design only taking three state measurements here. These results indicate that taking multiple state observations may be superior to taking both state and velocity measurements.

ODE1		ODE2	
d	$U(\mathbf{d})$	**d**	$U(\mathbf{d})$
(3, 9, 9)	3.89	(5, 9, 9)	1.90
(2, 8, 9)	3.86	(5, 5, 9)	1.88
(3, 8, 9)	3.56	(4, 9, 9)	1.87
(3, 8, 8)	3.85	(5, 6, 9)	1.86
(2, 6, 9)	3.84	(4, 5, 9)	1.85
(3, 6 ,9)	3.84	(4, 6, 9)	1.85
⋮	⋮	⋮	⋮
(0, 1, 1)	2.10	(0, 1, 1)	0.33
(0, 0, 1)	1.74	(0, 0, 1)	0.20
(0, 0, 0)	0.00	(0, 0, 0)	0.00

Table 4.2: Results of two-step batch design where only state knowledge is assumed. Designs are ranked from best (top) to worst (bottom).

4.2 Bayesian Optimization

Instead of testing all possible **d**, we now turn to Bayesian optimization in which designs are chosen to navigate the exploration vs. exploitation tradeoff. We apply the GP-UCB algorithm [8, 7, 81] to the continuous batch design case in which designs are chosen from the range $\mathscr{D} = [0,9]^2$. We again use the package *fmfn* [68] and use Gaussian processes [90] to quantify the objective and uncertainty at any point in $\mathscr{D}$. The only change in the algorithm is that the objective is calculated with only state observations as in the previous section. We apply Bayesian optimization only to one-step batch design, as we aim to compare the results against two-step batch design with state and velocity data in the previous chapter.

4.2.1 Results & Discussion

The results of two-step Bayesian optimization are shown in Table 4.3 and Figure 4.3. Looking at the choice of design for ODE1, the algorithm initially tested a random point and then started examining the boundary for the next six design choices. After trying another point near the middle of the grid, it chose to test a boundary point that is symmetric to the point with highest objective so far (point 6). Then the algorithm concludes by testing the origin. The best d recovered so far is $(4.01, 9.00)$ with objective score 3.56. This is the second-best **d** for ODE1, and Bayesian optimization was able

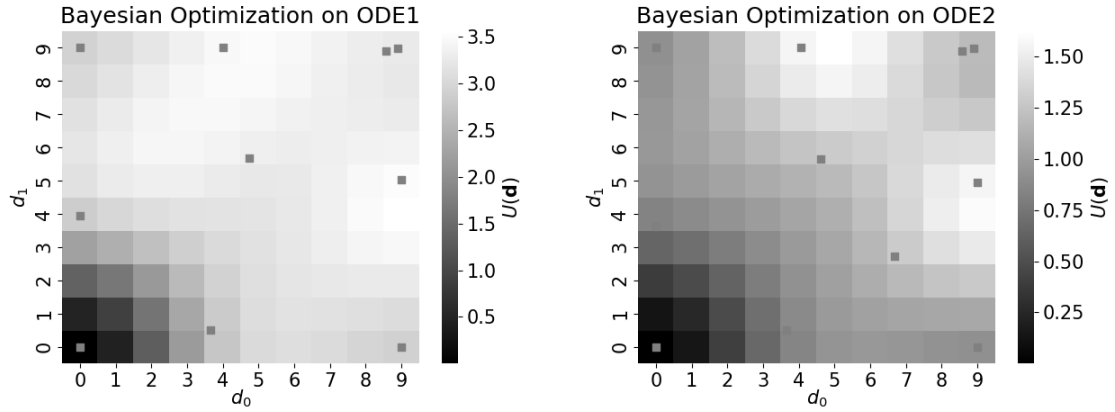

Figure 4.3: Heatplot of one-step Bayesian Optimization for ODE1 (left) and ODE2 (right). Only knowledge of the state observations is assumed.

to find it in only 9 attempts.

The algorithm follows a very similar theme for ODE2 except that it chooses a random point in the middle of the design space at choice 7 instead of 8. The most likely reason is that this is the first design with d_1 not on the boundary and it achieved such high score on the previous attempt that it was enticing to test a design in the form $\mathbf{d} = (\cdot, 5)$. The best design found was $(9.00, 4.94)$ with objective 1.63. This is very near $\mathbf{d}^*$ of ODE2 with objective score 1.62. The results show how a comparable $\mathbf{d}^*$ can be found by using many fewer attempts.

Both algorithms successfully found the optimal regions of the design space, as both heatmaps looked similar to their batch design counterparts in the previous section. The design choices aimed to search the boundaries of the grid as well as a few points near the middle. Bayesian optimization provided a relatively better $\mathbf{d}$ in this section for both ODE1 and ODE2 than in the previous chapter; although our results are of a single run. These results show that the batch problem may be easy in two dimensions, as Bayesian optimization was able to recover a near-optimal policy from the perspective of batch design in only 10 attempts. Future research directions may include examining a threshold based on the number of designs that batch optimization fails.

	ODE1		ODE2	
#	**d**	$U(\mathbf{d})$	**d**	$U(\mathbf{d})$
1	(3.67, 0.50)	2.78	(3.67, 0.50)	0.81
2	(8.91, 8.98)	3.26	(8.91, 8.98)	1.19
3	(8.57, 8.88)	3.26	(8.57, 8.88)	1.19
4	(0.00, 9.00)	2.92	(0.00, 9.00)	0.92
5	(9.00, 0.00)	2.93	(9.00, 0.0)	0.92
6	(9.00, 5.01)	3.50	(9.00, 4.94)	1.63
7	(3.75, 9.00)	2.74	(4.62, 5.64)	1.21
8	(4.75, 5.68)	3.23	(0.00, 3.63)	0.79
9	(4.01, 9.00)	3.57	(4.06, 9.00)	1.60
10	(0.00, 0.00)	0.00	(0.00, 0.00)	0.00

Table 4.3: Choice of design for Bayesian optimization applied to ODE1 and ODE2 where only state data is observed.

4.3 Greedy Design

Greedy design [84, 34, 33, 45, 54, 32, 16] is in stark contrast to discrete batch design and continuous batch design in that it forgoes multi-step lookahead and sequentially conducts experiments to greedily maximize the one-step batch objective. We apply one-step and two-step greedy design to ODE1 and ODE2 without velocity data. Since a single design does produces an underdetermined system in this chapter, we consider the one-step greedy design case to include two designs and the two-step case to include three designs. For all but the first experiment, we use the same objective from the previous chapter as

$$d_k^* = \arg\max_{d_k \in \mathscr{D}} U(d_k), \qquad \boldsymbol{\theta} \sim p_k. \tag{4.2}$$

where p_k is the posterior distribution at experiment k. In the case of the first experiment, we must optimize the objective over both d_0 and d_1 to produce a determined system. The computation of the utility can be found in the previous chapter in (3.13) and (3.14). To sample from the posterior, we again create a 2-dimensional grid that encodes the probability of 10,000 pairs of $\boldsymbol{\theta}$. The grid has 100 equally spaced points on each side. We perform one-step greedy design and two-step greedy design on $\mathscr{D} = \{0, 1, 2, \ldots, 9\}$ and compare results to batch design of this chapter and to greedy design of

Chapter 3.

4.3.1 Results & Discussion

The numerical results for ODE1 and ODE2 are shown in Figure 4.4. The two-step greedy design of ODE1 has value 3.89 which is identical to that of batch. One-step batch design has $\mathbf{d}^* = (3,9)$ and two-step has $\mathbf{d}^* = (3,9,9)$. Therefore, two-step greedy design chooses d_2 of the sequence $\mathbf{d} = (3,9,d_2)$. Since $(d_0,d_1) = (3,9)$ is a subsequence of $\mathbf{d}^*$ of batch design, greedy design can always d_2 to equal batch design. Therefore, we expect it to be somewhat better with the feedback from the first experiment. We see that the objective of one-step greedy design for ODE1 is slightly lower than that of batch. With more samples, we expect the two to be equal, although the discrepancy is only by 0.03. Therefore, we conclude that the extra feedback is not necessarily helpful in the case of ODE1.

A similar phenomenon occurs for two-step greedy design of ODE2. The objective produced by two-step greedy design is 1.93 and that of batch is 1.90. However, the graphs show that one-step greedy objective is roughly 0.03 greater than batch. With more samples, we expect the one-step greedy objective to match that of batch and decrease by 0.03. Since expected KL-divergence is additive, the two-step greedy design should decrease by roughly by 0.03 as well, yielding no improvement. Again, the one-step greedy design is a subsequence of $\mathbf{d}^*$ of two-step batch design.

Choices of d_2 are shown in Figure 4.5. These results match those of two-step batch design: the designs chosen most often complete $\mathbf{d}^*$ of two-step batch design. In the case of ODE2, the second-most chosen design completes the second-best $\mathbf{d}$ of two-step batch design. For ODE1, $d_2 = 9$ is chosen roughly 70% of the time and d_2 chosen as 5 or 9 for ODE2 is chosen more than 85% of the time. Therefore, other actions than those completing the best batch designs are chosen a non-negligible amount of time.

4.4 Deep Reinforcement Learning

We apply DQN [67] to the RL problem that was transformed from a Bayesian sOED problem as discussed in Section 2.5. We proceed in the manner of Section 3.6 by following Shen and Huan

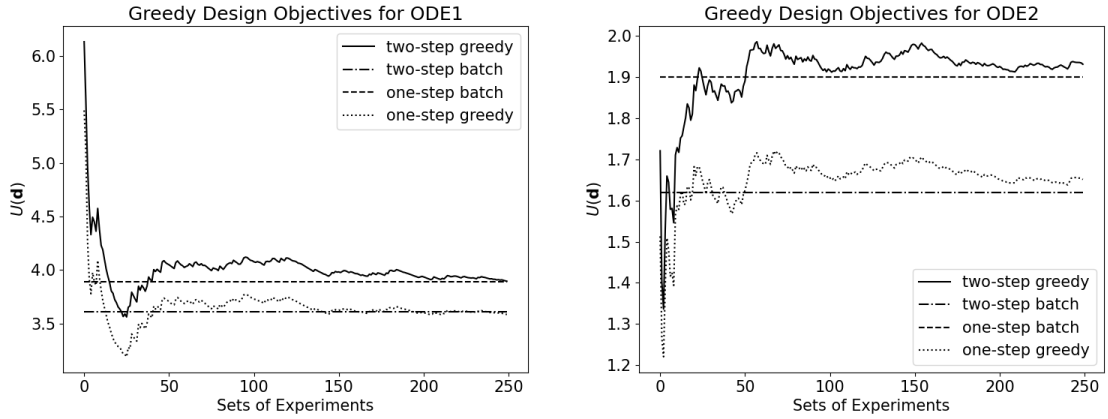

Figure 4.4: Numerical results of two-step greedy design applied for ODE1 (left) and ODE2 (right) without velocity data. Horizontal lines signify results from batch design for comparison.

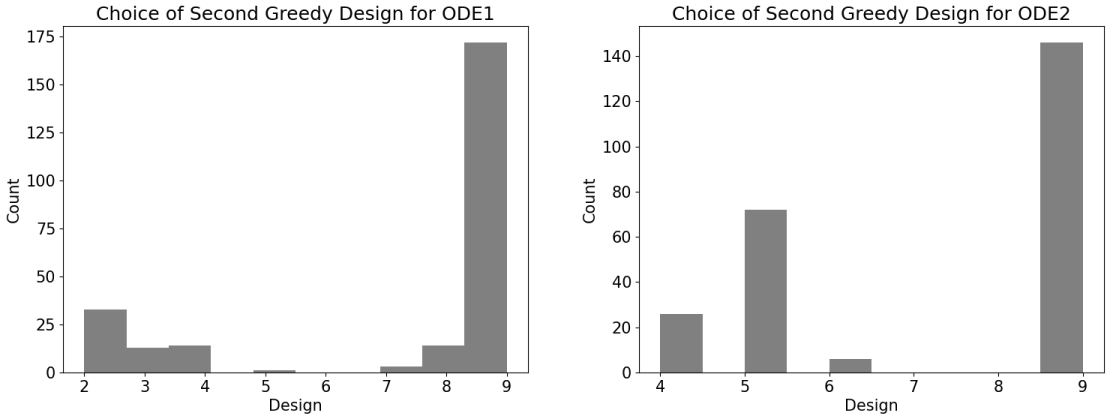

Figure 4.5: Counts of second design of greedy design with backwards view of ODE1 (left) and ODE2 (right) without velocity data.

[79]. The state of the underlying MDP is formed through the current experiment number, designs chosen, and observations measured. The state is a length 10 vector, where the first 4 components are analogous to a one-hot vector where component i is active if the MDP is currently at timestep i. The vector has form

$$s = [i_0, i_1, i_2, i_3, d_0, u_d, d_1, u_1, d_2, u_2]. \tag{4.3}$$

The actions are chosen from $\mathscr{D} = \{0, 1, 2, \ldots, 9\}$. The rewards are equal to the utility function of KL-divergence as computed in Section 3.3. We run DQN for $30,000$ timesteps which equates to $10,000$ episodes and analyze the results.

4.4.1 Results & Discussion

The results of DQN applied to ODE1 and ODE2 without velocity data are shown in Figure 4.6. For ODE1, the policy attains an objective score of about 3.8 past 10000 timesteps. On certain occasions, the score reached 3.90, which is greater than the batch and greedy scores of 3.89. The policy did not show much forgetting past 5000 timesteps and shows good convergence to a final policy. Again, due to the stochasticity inherent in the exploration, the policy will never converge completely.

For ODE2, the results were roughly an objective score of 1.85 past 10000 timesteps. These results are less than the 1.90 score achieved by batch and 1.93 by greedy design. Considering how the results seemed to plateau past 10000 timesteps, a more powerful DRL method may be required in addition to annealing the exploration and learning rates. In both graphs, there were poor policies during the exploration phase which were reconciled after the exploration decreased.

The results for DQN were slightly worse than those of batch and greedy design. Even though DQN is meant to optimize an objective that takes into account multi-step lookahead and feedback, the complicated architecture of DRL methods can prevent reaching the optimal solution, even if just slightly. However, the results highlight the speed of DRL methods, as a near-optimal policy was found in both situations within a minute on a single processor. We look forward to investigating the impact of more powerful architectures such as the actor-critic [66] architecture in future works.

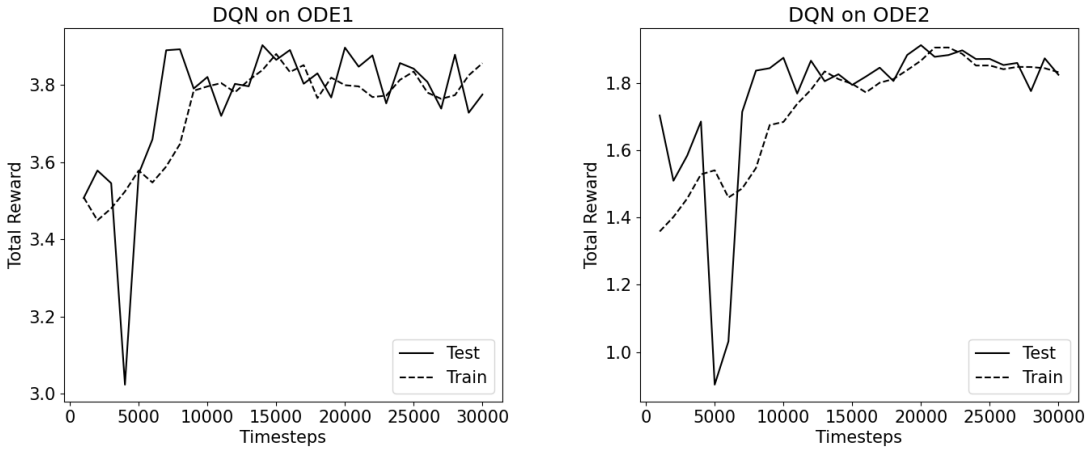

Figure 4.6: Training and testing results of applying DQN to ODE1 (left) and ODE2 (right) without velocity data.

4.5 Conclusion

We successfully applied batch design, black-box Bayesian optimization, greedy design, and DRL to the inverse problems of ODE1 and ODE2 with no velocity data. The lack of velocity data halved the amount of data received per design. Consequently the information gain when allowing state data from only two designs was lower than that of allowing both the state and velocity data from two designs. However, our results showed that incorporating state data from three designs was superior to state and velocity data from two designs. Therefore, the velocity data may not be informative, or it may share most of the same information as the state data. Regardless, the results show that Bayesian sOED for inverse problems can be solved just as easily with only state data, welcoming the use of these methods on widespread practical problems of interest.

Chapter 5

Discussion

In this work, we applied batch design, black-box Bayesian optimization, multi-armed bandits, greedy design, dynamic programming, approximate dynamic programming, and deep reinforcement learning to solve the Bayesian sOED problem. The observations were related to the experimental designs through underlying parameters that we aim to infer. Through quantifying information gain on probability distributions with KL-divergence, we constructed objectives for each of the above paradigms to maximize the information gain after completion of a fixed number of experiments. In the case of linear regression, we constructed a closed-form analytic expression for the objective, which allowed for exhaustive search over the design space. When using ODEs as underlying models, the objective did not have a closed form solution and mandated Monte Carlo sampling. Finally, we removed the velocity observation and showed how these methods can be applied to more practical problems of interest.

A central theme of this work is that of sample efficiency. An algorithm is said to be more sample efficient than another if its performance is better after seeing the same number and same quality of samples. For example, the RL training and testing graphs show the performance of DQN. A way to gauge the sample efficiency of DQN against another DRL method is to compare the training and testing graphs across many (e.g. 100) iterations of the algorithm. To compare a RL algorithm against another baseline algorithm such as batch, we compare performance after the same number of experiments are performed in each case.

This work presents numerous novelties. In Section 2.3, we performed dynamic programming across a discrete design space. We have not seen this algorithm performed in this manner in the literature. Terejanu et al. [84] apply greedy design to Bayesian sOED in which the underlying model parameters are fixed throughout all sets of experiments. In Section 3.4, we extend their work by allowing the true underlying parameters to change after completing a set of experiments. Our work provides a novel application of DRL to Bayesian sOED for inverse problems using differential equations, adding to the work of Shen and Huan [79]. We provide a comprehensive comparison of DRL methods with other Bayesian sOED methods on select inverse problems. Finally, our formulation of the Bayesian sOED problem for RL allows off-the-shelf algorithms to be used by researchers for easy DRL training.

We intend to apply the methodologies discussed in this work to at least one practical problem of interest. A prime model for study is the Lotka-Volterra differential equations system [62]. The Lotka-Volterra equations are used to model biological systems with two interacting populations. Since this system does not have an analytic solution, we must use numerical techniques such as ODE solvers to compute the likelihood function. Pending a theoretically and computationally sound algorithm, we may apply the algorithm to past datasets or new data if the opportunity arises. Other future directions include testing more powerful DRL algorithms, investigating the tradeoff between more observation data and more designs, extending the episode length to more experiments, and possibly creating a benchmark suite for Bayesian sOED problems. The Bayesian sOED problem has a wide scope with many avenues to explore which will provide many novel research directions for the future.

Chapter 6

Bibliography

[1] A. Alexanderian, N. Petra, G. Stadler, and O. Ghattas. A-optimal design of experiments for infinite-dimensional bayesian linear inverse problems with regularized ℓ_0-sparsification. *SIAM Journal on Scientific Computing*, 36(5):A2122–A2148, 2014.

[2] A. Alexanderian, N. Petra, G. Stadler, and O. Ghattas. A fast and scalable method for a-optimal design of experiments for infinite-dimensional bayesian nonlinear inverse problems. *SIAM Journal on Scientific Computing*, 38(1):A243–A272, 2016.

[3] A. Alexanderian and A. K. Saibaba. Efficient d-optimal design of experiments for infinite-dimensional bayesian linear inverse problems. *SIAM Journal on Scientific Computing*, 40(5):A2956–A2985, 2018.

[4] A. C. Atkinson. *Optimum Experimental Design*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[5] A. C. Atkinson and A. N. Donev. Optimum experimental designs. Technical report, Clarendon Press, 1992.

[6] A. Attia, A. Alexanderian, and A. K. Saibaba. Goal-oriented optimal design of experiments for large-scale bayesian linear inverse problems. *Inverse Problems*, 34(9):095009, 2018.

[7] P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

[8] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[9] Y. Bard. Nonlinear parameter estimation. Technical report, 1974.

[10] J. Beck, B. M. Dia, L. F. Espath, Q. Long, and R. Tempone. Fast bayesian experimental design: Laplace-based importance sampling for the expected information gain. *Computer Methods in Applied Mechanics and Engineering*, 334:523–553, 2018.

[11] M. Benson. Parameter fitting in dynamic models. *Ecological Modelling*, 6(2):97–115, 1979.

[12] J. M. Bernardo. Expected information as expected utility. *the Annals of Statistics*, pages 686–690, 1979.

[13] P. Bhaumik and S. Ghosal. Bayesian two-step estimation in differential equation models. *Electronic Journal of Statistics*, 9(2):3124–3154, 2015.

[14] P. Bhaumik and S. Ghosal. Efficient bayesian estimation and uncertainty quantification in ordinary differential equation models. *Bernoulli*, 23(4B):3537–3570, 2017.

[15] G. E. Box. Choice of response surface design and alphabetic optimality. In *Proceedings of the Conference on the Design of Experiments in Army Research, Development and Testing*, volume 28, page 238. US Army Research Office, 1982.

[16] G. E. Box. *Sequential experimentation and sequential assembly of designs*. Center for Quality and Productivity Improvement, University of Wisconsin-Madison, 1992.

[17] G. E. Box and H. Lucas. Design of experiments in non-linear situations. *Biometrika*, 46(1/2):77–90, 1959.

[18] G. E. P. Box. *Statistics for experimenters : design, discovery, and innovation*. Wiley series in probability and statistics. Wiley-Interscience, Hoboken, N.J., 2nd ed.. edition, 2005.

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[20] A. E. Brockwell and J. B. Kadane. A gridding method for bayesian sequential decision problems. *Journal of Computational and Graphical Statistics*, 12(3):566–584, 2003.

[21] N. J. Brunel. Parameter estimation of odes via nonparametric estimators. *Electronic Journal of Statistics*, 2:1242–1267, 2008.

[22] N. J. Brunel, Q. Clairon, and F. d'Alché Buc. Parametric estimation of ordinary differential equations with orthogonality conditions. *Journal of the American Statistical Association*, 109(505):173–185, 2014.

[23] D. Campbell and R. J. Steele. Smooth functional tempering for nonlinear differential equation models. *Statistics and Computing*, 22(2):429–443, 2012.

[24] D. A. Campbell. Bayesian collocation tempering and generalized profiling for estimation of parameters from differential equation models. 2007.

[25] B. P. Carlin, J. B. Kadane, and A. E. Gelfand. Approaches for optimal sequential decision analysis in clinical trials. *Biometrics*, pages 964–975, 1998.

[26] K. Chaloner. Optimal Bayesian Experimental Design for Linear Models. *The Annals of Statistics*, 12(1):283 – 300, 1984.

[27] K. Chaloner and I. Verdinelli. Bayesian experimental design: A review. *Statistical Science*, pages 273–304, 1995.

[28] J. A. Christen, P. Müller, K. Wathen, and J. Wolf. Bayesian randomized clinical trials: A decision-theoretic sequential design. *Canadian Journal of Statistics*, 32(4):387–402, 2004.

[29] D. R. Cox and N. Reid. *The theory of the design of experiments*. Chapman and Hall/CRC, 2000.

[30] I. Dattner and S. Gugushvili. Accelerated least squares estimation for systems of ordinary differential equations. *arXiv preprint arXiv:1503.07973*, 2015.

[31] C. De Boor and C. De Boor. *A practical guide to splines*, volume 27. springer-verlag New York, 1978.

[32] H. A. Dror and D. M. Steinberg. Sequential experimental designs for generalized linear models. *Journal of the American Statistical Association*, 103(481):288–298, 2008.

[33] C. C. Drovandi, J. M. McGree, and A. N. Pettitt. Sequential monte carlo for bayesian sequentially designed experiments for discrete data. *Computational Statistics & Data Analysis*, 57(1):320–335, 2013.

[34] C. C. Drovandi, J. M. McGree, and A. N. Pettitt. A sequential monte carlo algorithm to incorporate model uncertainty in bayesian sequential design. *Journal of Computational and Graphical Statistics*, 23(1):3–24, 2014.

[35] V. V. Fedorov. *Theory of optimal experiments*. Elsevier, 2013.

[36] R. A. Fisher. *The design of experiments*. Oliver & Boyd, Edinburgh ; London, 8th ed.. edition, 1966.

[37] I. Ford, D. Titterington, and C. P. Kitsos. Recent advances in nonlinear experimental design. *Technometrics*, 31(1):49–60x, 1989.

[38] R. Gautier and L. Pronzato. Adaptive control for sequential design. *Discussiones Mathematicae Probability and Statistics*, 1(20):97–114, 2000.

[39] A. Gelman, F. Bois, and J. Jiang. Physiological pharmacokinetic analysis using population modeling and informative prior distributions. *Journal of the American Statistical Association*, 91(436):1400–1412, 1996.

[40] M. Girolami. Bayesian inference for differential equations. *Theoretical Computer Science*, 408(1):4–16, 2008.

[41] J. Gittins, K. Glazebrook, and R. Weber. *Multi-armed bandit allocation indices*. John Wiley & Sons, 2011.

[42] J. C. Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society: Series B (Methodological)*, 41(2):148–164, 1979.

[43] S. Gugushvili and C. A. Klaassen. $\sqrt{n}$-consistent parameter estimation for systems of ordinary differential equations: bypassing numerical integration via smoothing. *Bernoulli*, 18(3):1061–1098, 2012.

[44] L. M. Haines, I. Perevozskaya, and W. F. Rosenberger. Bayesian optimal designs for phase i clinical trials. *Biometrics*, 59(3):591–600, 2003.

[45] M. Hainy, C. C. Drovandi, and J. M. McGree. Likelihood-free extensions for bayesian sequentially designed experiments. In *mODa 11-Advances in Model-Oriented Design and Analysis*, pages 153–161. Springer, 2016.

[46] E. Hairer, S. P. Norsett, and G. Wanner. Solving ordinary differential equations i: Nonstiff problems, volume 8 of computational mathematics, 1993.

[47] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[48] X. Huan. *Accelerated Bayesian experimental design for chemical kinetic models*. PhD thesis, Massachusetts Institute of Technology, 2010.

[49] X. Huan and Y. Marzouk. Optimal bayesian experimental design for combustion kinetics. In *49th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, page 513, 2011.

[50] X. Huan and Y. Marzouk. Gradient-based stochastic optimization methods in bayesian experimental design. *International Journal for Uncertainty Quantification*, 4(6), 2014.

[51] X. Huan and Y. M. Marzouk. Simulation-based optimal bayesian experimental design for nonlinear systems. *Journal of Computational Physics*, 232(1):288–317, 2013.

[52] X. Huan and Y. M. Marzouk. Sequential bayesian optimal experimental design via approximate dynamic programming. *arXiv preprint arXiv:1604.08320*, 2016.

[53] J. Jaeger. Functional estimation in systems defined by differential equations using bayesian smoothing methods. 2012.

[54] S. Kleinegesse, C. Drovandi, and M. U. Gutmann. Sequential bayesian experimental design for implicit models via mutual information. *Bayesian Analysis*, 16(3):773–802, 2021.

[55] K. Koval, A. Alexanderian, and G. Stadler. Optimal experimental design under irreducible uncertainty for linear inverse problems governed by pdes. *Inverse Problems*, 36(7):075007, 2020.

[56] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[57] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[58] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062. PMLR, 2018.

[59] D. V. Lindley. On a measure of the information provided by an experiment. *The Annals of Mathematical Statistics*, 27(4):986–1005, 1956.

[60] D. V. Lindley. *Bayesian statistics: A review*. SIAM, 1972.

[61] Q. Long, M. Scavino, R. Tempone, and S. Wang. A laplace method for under-determined bayesian optimal experimental designs. *Computer Methods in Applied Mechanics and Engineering*, 285:849–876, 2015.

[62] A. J. Lotka. Contribution to the theory of periodic reactions. Jan. 1909.

[63] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[64] R. Mattheij and J. Molenaar. *Ordinary differential equations in theory and practice*. SIAM, 2002.

[65] P. Müller. Simulation based optimal design. In D. Dey and C. Rao, editors, *Bayesian Thinking*, volume 25 of *Handbook of Statistics*, pages 509–518. Elsevier, 2005.

[66] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[67] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[68] F. Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014–.

[69] A. M. Overstall and D. C. Woods. Bayesian design of experiments using approximate coordinate exchange. *Technometrics*, 59(4):458–470, 2017.

[70] K. B. Petersen, M. S. Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7(15):510, 2008.

[71] L. Pronzato. *Design of experiments in nonlinear models : asymptotic normality, optimality criteria and small-sample properties*. Lecture notes in statistics (Springer-Verlag) ; v.212. Springer, New York, NY, 2013.

[72] X. Qi and H. Zhao. Asymptotic efficiency and finite-sample properties of the generalized profiling estimation of parameters in ordinary differential equations. *The Annals of Statistics*, 38(1):435–481, 2010.

[73] J. O. Ramsay, G. Hooker, D. Campbell, and J. Cao. Parameter estimation for differential equations: a generalized smoothing approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(5):741–796, 2007.

[74] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[75] S. Rogers, R. Khanin, and M. Girolami. Bayesian model-based inference of transcription factor activity. *BMC bioinformatics*, 8(2):1–11, 2007.

[76] E. G. Ryan, C. C. Drovandi, J. M. McGree, and A. N. Pettitt. A review of modern computational algorithms for bayesian optimal design. *International Statistical Review*, 84(1):128–154, 2016.

[77] K. J. Ryan. Estimating expected information gains for experimental designs with application to the random fatigue-limit model. *Journal of Computational and Graphical Statistics*, 12(3):585–603, 2003.

[78] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[79] W. Shen and X. Huan. Bayesian sequential optimal experimental design for nonlinear models using policy gradient reinforcement learning. *arXiv preprint arXiv:2110.15335*, 2021.

[80] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[81] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

[82] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[83] G. Terejanu, C. Bryant, and K. Miki. Bayesian optimal experimental design for the shock-tube experiment. In *Journal of Physics: Conference Series*, volume 410, page 012040. IOP Publishing, 2013.

[84] G. Terejanu, R. R. Upadhyay, and K. Miki. Bayesian experimental design for the active nitridation of graphite by atomic nitrogen. *Experimental Thermal and Fluid Science*, 36:178–193, 2012.

[85] G. Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[86] B. vanDomselaar and P. W. Hemker. Nonlinear parameter estimation in initial value problems. *Stichting Mathematisch Centrum. Numerieke Wiskunde*, (NW 18/75), 1975.

[87] J. M. Varah. A spline least squares method for numerical parameter estimation in differential equations. *SIAM Journal on Scientific and Statistical Computing*, 3(1):28–46, 1982.

[88] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[89] E. O. Voit and J. Almeida. Decoupling dynamical systems for pathway identification from metabolic profiles. *Bioinformatics*, 20(11):1670–1681, 2004.

[90] C. K. Williams and C. E. Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[91] H. Wu, H. Xue, and A. Kumar. Numerical discretization-based estimation methods for ordinary differential equation models via penalized spline smoothing with applications in biomedical research. *Biometrics*, 68(2):344–352, 2012.

[92] H. Xue, H. Miao, and H. Wu. Sieve estimation of constant and time-varying coefficients in nonlinear ordinary differential equation models by considering both numerical error and measurement error. *Annals of statistics*, 38(4):2351, 2010.

Chapter 7

Appendix

In Section 7.1, we derive formulas for the posterior distribution of the linear model in Chapter 2. In Section 7.2, we derive formulas for the KL-divergence between two multivariate normal distributions. The notation is introduced in Section 1.2.

7.1 Derivation of Posterior for Linear Model

We derive formulas for the posterior distribution of the linear model in Chapter 2. One new piece of notation is the design matrix X. With a linear model,

$$G(\boldsymbol{\theta}, d_k) = \theta_1 + \theta_2 \cdot d_k, \tag{7.1}$$

The design matrix is

$$X = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ d_0 & d_1 & d_2 & \cdots & d_n \end{pmatrix} \tag{7.2}$$

The analysis below assumes that multiple design matrices $\mathbf{X} = \{\mathbf{X}_i\}_{i=0}^m$ have been tested with corresponding observations $\mathbf{Y} = \{\mathbf{y}_i\}_{i=0}^m$. We start by using Bayes' Theorem and substituting the explicit

likelihood term.

$$
\begin{aligned}
p(\boldsymbol{\theta}|\mathbf{Y},\mathbf{X}) &\propto p(\mathbf{Y}|\boldsymbol{\theta},\mathbf{X})p(\boldsymbol{\theta}) \\
&= \left[\prod_{i=0}^{m-1} \mathscr{N}_{\mathbf{y}_i}(\mathbf{X}_i^T\boldsymbol{\theta},\sigma^2)\right] p(\boldsymbol{\theta}) \\
&= \left[\prod_{i=0}^{m-1} \left(\frac{1}{\sigma\sqrt{2\pi}}\right) \exp\left\{-\frac{1}{2}(\mathbf{y}_i-\mathbf{X}_i^T\boldsymbol{\theta})^T\sigma^{-2}(\mathbf{y}_i-\mathbf{X}_i^T\boldsymbol{\theta})\right\}\right] p(\boldsymbol{\theta})
\end{aligned}
$$

Next, we make substitutions for constants independent of $\boldsymbol{\theta}$ as z_i and arrange terms inside the exponential.

$$
\begin{aligned}
&= \left[\prod_{i=0}^{m-1} z_i \exp\left\{-\frac{1}{2}(\mathbf{y}_i-\mathbf{X}_i^T\boldsymbol{\theta})^T\sigma^{-2}(\mathbf{y}_i-\mathbf{X}_i^T\boldsymbol{\theta})\right\}\right] p(\boldsymbol{\theta}) \\
&= \left[\prod_{i=0}^{m-1} z_i\right]\left[\exp\left\{\sum_{i=0}^{m-1} -\frac{1}{2}(\mathbf{y}_i-\mathbf{X}_i^T\boldsymbol{\theta})^T\sigma^{-2}(\mathbf{y}_i-\mathbf{X}_i^T\boldsymbol{\theta})\right\}\right] p(\boldsymbol{\theta}) \\
&= \left[\prod_{i=0}^{m-1} z_i\right]\left[\exp\left\{-\frac{\sigma^{-2}}{2}\sum_{i=0}^{m-1}(\mathbf{y}_i-\mathbf{X}_i^T\boldsymbol{\theta})^T(\mathbf{y}_i-\mathbf{X}_i^T\boldsymbol{\theta})\right\}\right] p(\boldsymbol{\theta}) \\
&= \left[\prod_{i=0}^{m-1} z_i\right]\left[\exp\left\{-\frac{\sigma^{-2}}{2}\sum_{i=0}^{m-1}\mathbf{y}_i^2-\boldsymbol{\theta}^T\mathbf{X}_i\mathbf{y}_i-\mathbf{y}_i\mathbf{X}_i^T\boldsymbol{\theta}+\boldsymbol{\theta}^T\mathbf{X}_i\mathbf{X}_i^T\boldsymbol{\theta}\right\}\right] p(\boldsymbol{\theta})
\end{aligned}
$$

We substitute the normal distribution in for the prior and expand the exponent in the exponential.

$$
\begin{aligned}
&= \left[\prod_{i=0}^{m-1} z_i\right]\left[\exp\left\{-\frac{\sigma^{-2}}{2}\sum_{i=0}^{m-1}\mathbf{y}_i^2-\boldsymbol{\theta}^T\mathbf{X}_i\mathbf{y}_i-\mathbf{y}_i\mathbf{X}_i^T\boldsymbol{\theta}+\boldsymbol{\theta}^T\mathbf{X}_i\mathbf{X}_i^T\boldsymbol{\theta}\right\}\right] \\
&\qquad \cdot \quad \left(\frac{1}{2\pi(\det\Sigma)^{1/2}}\right)\exp\left\{-\frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu})\right\} \\
&= \left[\prod_{i=0}^{m-1} z_i\right]\left[\exp\left\{-\frac{\sigma^{-2}}{2}\sum_{i=0}^{m-1}\mathbf{y}_i^2-\boldsymbol{\theta}^T\mathbf{X}_i\mathbf{y}_i-\mathbf{y}_i\mathbf{X}_i^T\boldsymbol{\theta}+\boldsymbol{\theta}^T\mathbf{X}_i\mathbf{X}_i^T\boldsymbol{\theta}\right\}\right] \\
&\qquad \cdot \quad \left(\frac{1}{2\pi(\det\Sigma)^{1/2}}\right)\exp\left\{-\frac{1}{2}(\boldsymbol{\theta}^T\Sigma^{-1}\boldsymbol{\theta}-\boldsymbol{\theta}^T\Sigma^{-1}\boldsymbol{\mu}-\boldsymbol{\mu}^T\Sigma^{-1}\boldsymbol{\theta}+\boldsymbol{\mu}^T\Sigma^{-1}\boldsymbol{\mu})\right\}
\end{aligned}
$$

Next, we isolate the exponent of the product of exponential terms. In the exponent, we combine

like terms in preparation to complete the square.

$$\begin{aligned}
&\left[-\frac{\sigma^{-2}}{2}\sum_{i=0}^{m-1}\mathbf{y}_i^2-\boldsymbol{\theta}^T\mathbf{X}_i\mathbf{y}_i-\mathbf{y}_i\mathbf{X}_i^T\boldsymbol{\theta}+\boldsymbol{\theta}^T\mathbf{X}_i\mathbf{X}_i^T\boldsymbol{\theta}\right]\\
&\qquad+\left[-\frac{1}{2}(\boldsymbol{\theta}^T\Sigma^{-1}\boldsymbol{\theta}-\boldsymbol{\theta}^T\Sigma^{-1}\boldsymbol{\mu}-\boldsymbol{\mu}^T\Sigma^{-1}\boldsymbol{\theta}+\boldsymbol{\mu}^T\Sigma^{-1}\boldsymbol{\mu})\right]\\
=\quad&-\frac{1}{2}\Bigg(\left[\sigma^2\sum_{i=0}^{m-1}\boldsymbol{\theta}^T\mathbf{X}_i\mathbf{X}_i^T\boldsymbol{\theta}\right]-\left[\sigma^2\sum_{i=0}^{m-1}\boldsymbol{\theta}^T\mathbf{X}_i\mathbf{y}_i\right]-\left[\sigma^2\sum_{i=0}^{m-1}\mathbf{y}_i\mathbf{X}_i^T\boldsymbol{\theta}\right]+\left[\sigma^2\sum_{i=0}^{m-1}\mathbf{y}_i^2\right]\\
&\qquad+\boldsymbol{\theta}^T\Sigma^{-1}\boldsymbol{\theta}-\boldsymbol{\theta}^T\Sigma^{-1}\boldsymbol{\mu}-\boldsymbol{\mu}^T\Sigma^{-1}\boldsymbol{\theta}+\boldsymbol{\mu}^T\Sigma^{-1}\boldsymbol{\mu}\Bigg)\\
=&-\frac{1}{2}\Bigg(\boldsymbol{\theta}^T\left[\Sigma^{-1}+\sigma^{-2}\sum_{i=0}^{m-1}\mathbf{X}_i\mathbf{X}_i^T\right]\boldsymbol{\theta}-\boldsymbol{\theta}^T\left[\Sigma^{-1}\boldsymbol{\mu}+\sigma^{-2}\sum_{i=0}^{m-1}\mathbf{X}_i\mathbf{y}_i\right]\\
&\qquad-\left[\boldsymbol{\mu}^T\Sigma^{-1}+\sigma^{-2}\sum_{i=0}^{m-1}\mathbf{y}_i\mathbf{X}_i^T\right]\boldsymbol{\theta}+\left[\boldsymbol{\mu}^T\Sigma^{-1}\boldsymbol{\mu}+\sigma^{-2}\sum_{i=0}^{m-1}\mathbf{y}_i^2\right]\Bigg)
\end{aligned}$$

We make the substitutions

$$A=\left[\Sigma^{-1}+\sigma^{-2}\sum_{i=0}^{m-1}\mathbf{X}_i\mathbf{X}_i^T\right],\ b=\left[\Sigma^{-1}\boldsymbol{\mu}+\sigma^{-2}\sum_{i=0}^{m-1}\mathbf{X}_i\mathbf{y}_i\right],\ c=\left[\boldsymbol{\mu}^T\Sigma^{-1}\boldsymbol{\mu}+\sigma^{-2}\sum_{i=0}^{m-1}\mathbf{y}_i^2\right],$$

to yield the following:

$$=-\frac{1}{2}(\boldsymbol{\theta}^TA\boldsymbol{\theta}-\boldsymbol{\theta}^Tb-b^T\boldsymbol{\theta}+c)$$

Next, we subtract c and add $b^TA^{-1}AA^{-1}b$ inside the exponential. Note that we are working inside the exponent of an exponential. Therefore, adding a constant is equivalent to multiplying by a constant outside the expression, which can be neglected since we are working with the unnormalized posterior. We also make the substitution $\Sigma_{new}=A^{-1},\ \boldsymbol{\mu}_{new}=A^{-1}b$ to yield the following:

$$\begin{aligned}
&=-\frac{1}{2}(\boldsymbol{\theta}^TA\boldsymbol{\theta}-\boldsymbol{\theta}AA^{-1}b-b^TA^{-1}A\boldsymbol{\theta}+b^TA^{-1}AA^{-1}b)\\
&=\boldsymbol{\theta}^T\Sigma_{new}^{-1}\boldsymbol{\theta}-\boldsymbol{\theta}^T\Sigma_{new}^{-1}\boldsymbol{\mu}_{new}-\boldsymbol{\mu}_{new}^T\Sigma_{new}^{-1}\boldsymbol{\theta}+\boldsymbol{\mu}_{new}^T\Sigma_{new}^{-1}\boldsymbol{\mu}_{new}\\
&=(\boldsymbol{\theta}-\boldsymbol{\mu}_{new})^T\Sigma_{new}^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu}_{new}).
\end{aligned}$$

Since we can write the posterior as a constant times $\exp\left\{-\frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\mu}_{new})^T\Sigma_{new}^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu}_{new})\right\}$, the posterior is indeed $\mathcal{N}(\mu_{new},\Sigma_{new}) = \mathcal{N}(A^{-1}b,A^{-1})$. These formulas can be compared with those by Chaloner & Verdinelli [27].

7.2 Derivation of KL Divergence for Linear Model

We derive formulas for the KL-divergence between two multivariate normal distributions. We can decompose KL-divergence into the following two terms:

$$\begin{aligned}
D_{KL}(p(\cdot|\mathbf{y},\mathbf{d})||p(\cdot)) &= \int_\Theta \log\left(\frac{p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})}{p(\boldsymbol{\theta})}\right) p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})\; d\boldsymbol{\theta} \\
&= \int_\Theta (\log(p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d}))p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d}) - \log p(\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d}))\; d\boldsymbol{\theta}, \\
&= \int_\Theta \log(p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d}))p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})\; d\boldsymbol{\theta} - \int_\Theta \log p(\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})\; d\boldsymbol{\theta}.
\end{aligned}$$

We begin by creating a closed-form analytic expression for the first term. First, we substitute the explicit posterior and substitute $u = (2\pi)^{-1}(\det\Sigma)^{1/2}$:

$$\begin{aligned}
&\int_\Theta \log(p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d}))p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})\; d\boldsymbol{\theta} \\
&= \int_\Theta \log\left((2\pi)^{-1}(\det\Sigma)^{-1/2}\exp\left\{-\frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu})\right\}\right) p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})\; d\boldsymbol{\theta} \\
&= \int_\Theta \log\left(u\cdot\exp\left\{-\frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu})\right\}\right) p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})\; d\boldsymbol{\theta} \\
&= \int_\Theta \left[\log(u) - \frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu})\right] p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})\; d\boldsymbol{\theta}
\end{aligned}$$

The integral can be split over both terms, and the second term can be simplified to a conditional expectation over $\boldsymbol{\theta}$.

$$
\begin{aligned}
&= \int_{\Theta} \log(u) p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})\, d\boldsymbol{\theta} \; - \frac{1}{2} \int_{\Theta} (\boldsymbol{\theta}-\boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{\theta}-\boldsymbol{\mu}) p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})\, d\boldsymbol{\theta} \\
&= \log(u) - \frac{1}{2} \mathbb{E}_{\boldsymbol{\theta}|\mathbf{y},\mathbf{d}} \left[(\boldsymbol{\theta}-\boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{\theta}-\boldsymbol{\mu}) \right]
\end{aligned}
$$

The conditional expectation is equivalent to a conditional expectation over a χ^2_n distribution having mean n [70]. We substitute and simplify:

$$
\begin{aligned}
&= \log(u) - \frac{n}{2}, \\
&= \log((2\pi)^{-1} (\det \Sigma)^{-1/2}) - \frac{n}{2}, \\
&= -\log(2\pi) - \frac{1}{2} \log(\det \Sigma) - \frac{n}{2}.
\end{aligned}
$$

Similarly, we note that the prior has the same form as the posterior: it is a multivariate normal distribution with mean and covariance parameters. The same analysis holds, resulting in the following:

$$
\int_{\Theta} \log p(\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})\, d\boldsymbol{\theta} = -\log(2\pi) - \frac{1}{2} \log(\det \Sigma_{prior}) - \frac{n}{2}. \tag{7.3}
$$

Combining these two equations yields the formula for KL-divergence:

$$
\begin{aligned}
D_{KL}(p(\cdot|\mathbf{y},\mathbf{d}) || p(\cdot)) &= \int_{\Theta} \log \left(\frac{p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})}{p(\boldsymbol{\theta})} \right) p(\boldsymbol{\theta}|\mathbf{y},\mathbf{d})\, d\boldsymbol{\theta} \\
&= \frac{1}{2} [\log(\det(\Sigma_{prior})) - \log(\det(\Sigma_{posterior}))].
\end{aligned}
$$