Bayesian optimal design using stochastic gradient optimisation and surrogate utility functions

Sophie Harbisher

Thesis submitted for the degree of
Doctor of Philosophy

School of Mathematics, Statistics & Physics
Newcastle University
Newcastle upon Tyne
United Kingdom

December 2020

Acknowledgements

Thank you to EPSRC for providing the funding which has allowed me to pursue a PhD; Dr Dennis Prangle and Dr Colin Gillespie for their guidance, expertise and support; Everyone at the school of MSP for making my time there enjoyable; Finally, my friends and family for their encouragement.

Abstract

Experimental design is becoming increasingly important to many applications from genetic research to robotics. It provides a structured way of allocating resources in an efficient manner prior to an experiment being conducted. Assuming a model for the data, one approach is to introduce a utility function to quantify the worth of a design given some data and parameters. Typically experiment-specific utility functions are difficult to elicit and hence a pragmatic choice of utility concerning the information gained about the model parameters is used. Bayesian experimental design aims to maximise the expected utility accounting for uncertainty in the model parameters and the data which could be observed. For this approach, difficulties arise as the expected utility is typically intractable and computationally costly to approximate.

Modern applications often seek high dimensional designs. In these settings existing algorithms such as the MCMC scheme of Müller (1999) and ACE (Overstall and Woods, 2017), require a high number of utility evaluations before they converge. For the most commonly used utility functions this becomes a computationally costly exercise. Therefore there is a need for an efficient and scalable method for finding the Bayesian optimal design.

The contributions of this thesis are as follows. Firstly, stochastic gradient optimisation, a scalable method widely used in the field of machine learning, is applied to the Bayesian experimental design problem. The second contribution is to consider a utility function based on the Fisher information matrix as a Bayesian utility function by showing it has a decision theoretic justification. These utilities are often available in a closed-form so are fast to compute. The final contribution is to investigate surrogate functions for expensive utilities as an efficient way of finding promising regions of the design space.

Contents

1 Introduction **1**

2 Background **7**

2.1 Bayesian parameter inference 7

2.1.1 Markov chain Monte Carlo 8

2.1.2 Importance sampling. 9

2.1.3 Bayesian asymptotics 10

2.2 Fisher information 11

2.2.1 Binomially distributed random variables 13

2.2.2 Normally distributed random variables 14

2.3 Experimental design 16

2.3.1 Classical experimental design 16

2.3.2 Bayesian experimental design 17

2.4 Utility functions 18

2.5 Scoring rules. 19

2.5.1 Bayesian decision theoretic approach 20

2.6 Existing methods for Bayesian optimal design 21

2.6.1 Müller algorithm. 21

2.6.2 Approximate co-ordinate exchange algorithm 23

2.6.3 Other optimal design algorithms 25

3 Stochastic gradient descent for optimal design 27

3.1 Stochastic gradient descent 27

3.1.1 Gradient descent 28

3.1.2 Vanilla SGD 29

3.1.3 SGD with momentum 30

3.1.4 Adaptive moment estimation 30

3.1.5 Other SGD algorithms 32

3.1.6 Illustration of SGD algorithms 32

3.2 Constrained SGD 34

3.3 Gradient estimation 35

3.3.1 Gradient computation 35

3.3.2 Software choice 39

3.4 SGD applied to optimal design 39

4 Utility functions 41

4.1 Common Bayesian utility functions 41

4.2 Fisher information gain 42

4.2.1 Derivation of the Fisher information gain 43

4.2.2 FIG for data models. 44

4.2.3 Scaled FIG 46

4.2.4 Adversarial FIG 47

4.2.5 Limitations 49

4.3 Utility based on classical D-optimality criterion 49

4.3.1 D-optimality utility function 50

5 Examples **53**

5.1 Simple death model 53

5.1.1 FIG utility 55

5.1.2 AFIG utility 56

5.1.3 *D*-optimality utility 57

5.2 Geospatial model 58

5.2.1 FIG utility 59

5.2.2 AFIG utility 63

5.2.3 *D*-optimality utility 63

5.3 Summary 64

6 Simulation study **67**

6.1 Compartmental pharmacokinetic model 67

6.2 FIG utility 69

6.2.1 Scaling FIG contributions 71

6.2.2 Tuning choices 72

6.2.3 Comparison between methods 73

6.2.4 Summary 76

6.3 *D*-optimality utility 78

6.4 AFIG framework 80

6.5 Comparison of optimal designs under different utility functions 81

6.6 Discussion 83

7 Surrogate utility function **87**

7.1 Surrogate posterior using neural networks 89

7.1.1 Neural network approximation of posterior 89

7.2 Bayesian optimal design using a posterior approximation . . . 90
7.3 SGD using surrogate utility. 91
7.4 Delayed acceptance Markov chain Monte Carlo 95
7.4.1 Application to optimal design 95
7.4.2 Example: two observation simple death model 96
7.4.3 Example: five observation simple death model 97
7.4.4 Summary 100
7.5 Regression trees 100
7.5.1 Application to optimal design 102
7.5.2 Example: one observation simple death model. 102
7.5.3 Example: two observation simple death model 104
7.6 Discussion 106
8 Conclusion **109**
8.1 SGD methods 109
8.1.1 SGD for Bayesian optimal design. 110
8.1.2 FIG utility 111
8.2 Surrogate utilities 112
8.3 Summary 112
Bibliography **114**

List of Algorithms

2.1 Metropolis-Hastings Markov chain Monte Carlo algorithm targetting the posterior for θ given data y observed at design τ. 9
2.2 Importance sampling 10
2.3 Müller algorithm using MCMC to target the optimal design. The batch size, M, is the number of utility evaluations per iteration of the scheme. 22
2.4 The approximate co-ordinate exchange algorithm. The tuning parameters are: N_1 and N_2, the number of iterations of phase 1 and phase 2 respectively; b_1 and b_2, the number of simulations used to establish the candidate design and compute the associated acceptance probability; Q, the number points at which to evaluate the expected utility to fit the model in phase 1. 24
3.1 General algorithm for SGD for n iterations. 28
3.2 Automatic differentiation of $g(x_1, x_2, \ldots, x_m)$ using forward propagation of gradients for a computational graph on N nodes, $n_1, n_2, \ldots, n_N$. The independent variables $x_1, x_2, \ldots, x_m$ and dependent variable g correspond to nodes $n_1, n_2, \ldots, n_m$ and n_N respectively. $Ch(n)$ is a function which gives the indices of the children of node n. 36
3.3 Automatic differentiation of $g(x_1, x_2, \ldots, x_m)$ using backwards propagation of gradients for a computational graph on N nodes, $n_1, n_2, \ldots, n_N$. The independent variables $x_1, x_2, \ldots, x_m$ and dependent variable g correspond to nodes $n_1, n_2, \ldots, n_m$ and n_N respectively. $Pa(n)$ is a function which gives the indices of the parents of node n. 37
3.4 Stochastic gradient optimisation of expected utility $\tilde{\mathcal{J}}(\tau) = \mathcal{J}(\tau) + \mathcal{P}(\tau)$. . 39
4.1 Stochastic gradient optimisation of expected Fisher information gain with adaptive weights. 47
4.2 Stochastic gradient optimisation of expected adversarial Fisher information gain $\mathcal{K}(\tau, B)$. 49
7.1 Delayed acceptance Metropolis-Hastings Markov chain Monte Carlo algorithm with target $f(x)$ and proposal density $q(\cdot)$. An approximation to to the target is denoted by $\tilde{f}$. 96
7.2 Pseudo code for tree construction. 101

7.3 Pseudo code for narrowing down the design space to only include near optimal designs. 102

List of Figures

3.1 The value of the objective function (left) and path (right) through the iterations of the vanilla SGD (top), SGD with momentum (middle) and Adam (bottom) algorithms where the objective function is the Rosenbrock function (Equation 3.12) with $a = 1$ and $b = 10$. All algorithms share the same initial state indicated by a green circle. The minimum of the function is at $(1, 1)$, represented by a red cross. 33

3.2 Example of a computational graph used in automatic differentiation of the function $g(x, y) = e^x \sin(x + y)$. 38

3.3 Forward propagation of gradients with respect to x. If the partial derivative of $g(\cdot)$ with respect to y was required the seed variables would be set as $n'_1 = 0$ and $n'_2 = 1$. 38

3.4 Backward propagation of gradients for the function $g(x, y) = e^x \sin(x + y)$. . 38

5.1 Simulated prior predictive population sizes for the simple death model with initial population size of 50. 55

5.2 The expected FIG utility surface of the simple death model (left) and associated trace plots of observation time τ against computation cost, measured in utility gradient evaluations, for 11 independent runs of the SGD algorithm (right). The optimal design τ^* is shown by the dotted line. . 56

5.3 The expected $\log|\mathcal{I}|$ utility surface of the simple death model (left) and associated trace plots of observation time τ against computation cost ,measured in utility gradient evaluations, for 11 independent runs of the SGD algorithm (right). The optimal design τ^* is shown by the dotted line. . 57

5.4 Example fields drawn from the geospatial model as defined in Equation 5.18 for $\theta_1 = 0.1$, $\theta_2 = -0.3$, $\sigma_1 = 0$, $\sigma_2 = \sqrt{0.5}$ and $\ell = 1$. The z axis shows the simulated function value over the unit square (x and y axis) 59

5.5 The expected FIG utility for the designs returned by the SGD and ACE algorithms (right) and trace plots of the utility over the computation of the algorithms (left) for the geospatial example. For this example the expected utility is deterministic and hence no estimation is required. The horizontal line indicates the utility for a uniformly spaced grid over the design space. The results shown are from 100 independent runs starting from different initial conditions consistent between methods. 62

5.6 The expected D-optimality utility for the designs returned by the SGD and ACE algorithms (left) and trace plots of the utility over the computation of the algorithms (right) when using the $\log|\mathcal{I}|$ utility function for the geospatial example. 64

5.7 An example design returned from the geospatial model using the FIG (left), AFIG (middle), and $\log|\mathcal{I}|$ (right) utility functions. 64

6.1 Realisations y from the compartmental PK model using the prior as defined in Equation 6.6 and with σ_1^2 and σ_2^2 fixed at 0.1 and 0.01 respectively. . . . 68

6.2 Trace plots of relative contribution to expected FIG (left), each observation time (middle) and utility (right) over the iterations of the SGD algorithm using the unweighted (top) and weighted (bottom) FIG utility for the PK example. 71

6.3 Trace plots of expected FIG utility for the PK example from 100 independent runs of the SGD algorithm against iterations (top) and computational cost (bottom) for various choice of batch size K used in the Monte Carlo estimate of the expected utility (Equation 3.16). The expected utility displayed is computed from the simulated utilities made during the algorithm. 73

6.4 Box plot showing the expected utility of the returned designs from 100 independent runs from different initial states. Each expected utility is a Monte Carlo estimate using $20,000$ realised utilities at the returned designs. The vertical line is the expected utility of a uniform grid over the search space. 74

6.5 Plots of the returned designs from the various runs of the SGD, ACE and Müller_1 algorithms when using the FIG utility. Each returned design was sorted by time and indexed by observation. The intensity of the colour represents the degree of repetition over the 100 independent runs of the algorithms. 75

6.6 (left) Monte Carlo estimate for the expected utility (using 2×10^6 particles) for different proportions of observations at times ≈ 1.2 and ≈ 8.2. The x axis shows the number of observations taken at time 1.2. (right) Expected utility as one design point is varied with all others at fixed values: four at time 1.2; ten at time 8.2. 76

6.7 Traceplot of the expected FIG utility through the iterations of the ACE (left) and SGD (right) algorithms for the 100 independent runs under the fixed computational budget. 77

6.8 Box plot showing the expected $\log|\mathcal{I}|$ utility of the returned designs from 100 independent runs from different initial states. The expected utility shown is from a Monte Carlo estimate using $20,000$ realised utilities at the returned designs. Both plots are the same but have different x-axis ranges to help inspection of results. 79

6.9 Plots of the returned designs from the various runs of the SGD, ACE and Müller_1 algorithms when using the $\log|\mathcal{I}|$ utility. Each returned design was sorted by time and indexed by the observation. The intensity of the colour represents the degree of repetition over the 100 independent runs of the algorithms. 80

6.10 Example traces of the elements of B (left) and the design (right) when using SGD algorithm to maximise the expected AFIG utility. 81

6.11 Plots of the returned designs from the various runs of the SGD algorithm using simultaneous updates for the expected AFIG utility. Each run uses a computational budget of $500,000$ utility evaluations. Each returned design was sorted by time. 82

6.12 Plots of 20 returned designs (from ACE) for the pharmacokinetic model under the SIG utility. 83

6.13 Optimal designs returned under the FIG, AFIG, SIG and $\log|\mathcal{I}|$ utility functions for the pharmacokinetic model. 84

6.14 Example bivariate posteriors for the parameters of the pharmacokinetic model for each of the designs in Figure 6.13 for a draw (θ, y) from the prior and model. Colour represents posterior density, with yellow showing highest density and purple lowest. From top to bottom, they correspond to FIG, SIG, $\log|\mathcal{I}|$ and AFIG. The priors are indicated by the white contours. 85

7.1 Example structure of a neural network used to estimate the mean $\mu = \mu(y, \tau)$ and variance $\Sigma = \Sigma(y, \tau)$ of the approximate posterior. 90

7.2 Comparison of the surrogate and true expected posterior precision utility for the simple death model for a one observation design. 92

7.3 Comparison of the expected posterior precision utility using the true posterior (left) and the neural network surrogate (right) for the simple death model for a two observation design. 93

7.4 SGD using a surrogate function for the posterior precision utility for the simple death model for a one observation (left) and two observation (right) design. The optimal design found using SGD on the surrogate is shown in red with the path of the SGD algorithms indicated for the two observation case. The known optimal is indicated by the purple marker. 94

7.5 Run time (left column), ESS/s (centre column) and ESS per expensive utility evaluation (right column) for the two observation simple death model for $M = 1$ (top row), 2 (middle row), and 4 (bottom row). 98

7.6 Run time (left column), ESS/s (centre column) and ESS per expensive utility evaluation (right column) for the five observation simple death model for $M = 1$ (top row), 2 (middle row), and 4 (bottom row). 99

7.7 An example regression tree to predict Y from variables X_1 and X_2. Here γ_i for $i = 1, 2, 3$ are the (constant) thresholds at which the splits occur. 101

7.8 The true expected utility surface found using numerical integration and regression tree prediction (left) for the expected utility for the simple death model with a one observation design. Areas which result in the top quartile (shown in yellow) for expected utility according to the regression tree output are shown in the plot on the right. 103

7.9 Output of the Gillespie Boys algorithm showing the number of times a sampled design was visited (left) and the estimated expected utility (right) for the simple death model with a one observation design. Optimal design $\tau^* \approx 1.61$ indicated by blue vertical line. 104

7.10 The true expected utility surface found using numerical integration (top left), the neural network surrogate (top right) and regression tree prediction (bottom left) for the expected utility surface in the simple death model with a two observation design, (τ_1, τ_2). Also included is a plot showing areas which give expected utilities in the top quantile of the regression tree output (bottom right). Areas which result in the top quantile (shown in yellow) for expected utility according to the regression tree output are shown in the plot on the right. The optimal design $\tau^* = (1.1, 2.51)$ is indicated by a cross. 105

7.11 Output of the Gillespie Boys algorithm showing number of times the particle has been visited (left) and the estimated expected utility (right). The optimal design $\tau^* = (1.1, 2.51)$ is indicated by a red cross. 106

List of Tables

2.1 Quantities associated with the logarithmic and Hyvärinen scores. 20

3.1 Description and default values of the parameters in the Adam algorithm (Section 3.1.4). 32

5.1 The optimal designs output by the SGD algorithm for the geospatial example under the FIG utility shown for various choices of γ and ℓ. The design space is a unit square centred at the origin. 61

6.1 Settings for the algorithms used in the simulation study under the default computational cost of the ACE algorithm (1.8×10^7). 72

7.1 The mean results when searching for the optimal two observation design for the simple death model using MCMC and DA-MCMC within the Müller algorithm. 97

7.2 The mean results when searching for the optimal five observation design for the simple death model using MCMC and DA-MCMC within the Müller algorithm. 100

Chapter 1

Introduction

Experimental design provides a structured way of allocating resources in order to gain the most knowledge and/or make a decision. This is especially relevant when a limited number of observations can be made in an experiment where the practitioner controls the value of one or more covariate values. Throughout this thesis a model will be assumed for the data observed in the experiment. The model will rely on some model parameters and the design (e.g. the time or location at which observations should be made). Modern applications such as placing sensors (Krause et al., 2009) or making observations in a numerical integration problem (Oates et al., 2019) have the resources to make many observations however this poses a computational challenge due to the size and dimension of the design space. Existing methods for experimental design are expensive, especially for high dimensional designs. This thesis will aim to address some of these difficulties using a scalable approach for finding the optimal deign and also investigating how the worth of a design can be quantified using computationally convenient functions.

There are two main areas of focus in experimental design: factorial design and optimal design. Experimental design in the early twentieth century focused on classical factorial designs with Fisher (1935) considered the foundational work on the area (Yates, 1964; Box, 1980). Factorial design is where an experiment has two or more factors which can take a finite number of discrete values. The interest lies in the effect the factors have on the response variable. Only the value of the response variable is required for such an analysis and thus a model does not need to be specified allowing processes with complex error structures to be considered. Examples of these include investigating crop yield from different fertilisers in different areas of a field (Yates, 1935) or conducting a chemical analysis on different specimens with various operators carrying out the testing (Snee, 1983). The aim of factorial design is to assess the value of the response variable for each of the factors as well as any interactions between them.

Optimal design considers a different problem. Often a model with a simple error structure is assumed which describes the relationship between the response and some design variables

which the practitioner controls. These models can also involve some parameters which the practitioner cannot control but which have an effect on the response. These are often referred to as nuisance parameters. The aim of optimal design is to select the design which maximises the value of conducting the experiment. In the Bayesian paradigm this is quantified through a utility function or an optimality criterion in the classical setting. Often the interest lies in minimising the variance of the posterior (in the Bayesian setting) or of the estimator (in the classical setting) rather than looking at the value of the response variable as in the factorial experimental design. Examples which use optimal design include determining sampling times in pharmacokinetic studies (Ryan et al., 2015) or selecting the stress levels for degradation tests in engineering (Liu and Tang, 2010).

Kirstine Smith is credited with founding the field of optimal design. She proposed that in experiments which were constrained by resources, such as time or expenditure, the practitioner should make observations at a design which would maximise the knowledge gained. Her doctoral dissertation (Smith, 1918) considered optimal designs for polynomial regression models. Gustav Elfving was also instrumental in founding the field of experimental design. Whilst confined to a tent due to bad weather on an expedition in Greenland, Elfving considered the best locations to make observations in order to estimate parameters in linear models (Elfving et al., 1952). Fellman (1999) gives a review of Elfving's contribution to the emergence of optimal experimental design theory. The paper by Kiefer (1959) was also important, marking the start of the systematic study of the properties and construction of optimal designs. Their work and contribution is summarised by Wynn (1984). Lindley (1972) following work from Minton et al. (1962) introduced Bayesian optimal design by considering optimal design as a decision problem accounting for the uncertainties in both the data which could be observed and the parameter values. Müller (1999) presented the first practical, general purpose method for Bayesian design using a Markov chain Monte Carlo approach. For a more comprehensive history of experimental design see Atkinson and Bailey (2001).

Classical optimal design has been well studied in the literature (for example Box, 1982; Hinkelmann and Maman, 2005; Atkinson et al., 2007). It mainly focuses on selecting a design which optimises a criterion based on the Fisher information (Fisher, 1925), aimed at minimising the variance of the estimator of the model parameters. The classical setting fails to take into account any prior information about the model parameters, instead the Fisher information is evaluated at a selected parameter value, often the maximum likelihood estimate. In practice, prior to conducting an experiment there will be some knowledge of these from previous experiments or from the expertise of the practitioner. The Bayesian setting naturally incorporates this prior knowledge.

Methods for Bayesian optimal design have been developed and studied much more recently, corresponding to the availability of computing resources. Throughout this thesis a model

will be assumed for the data which relies on the design and some unknown parameter values. In the Bayesian setting a distribution will be provided for the model parameters which summarises the beliefs about them prior to the experiment taking place. The worth of a design for a particular value of data and parameters is quantified through a utility function. Ideally an experiment-specific utility function would be elicited however this is difficult in practice. Often more pragmatic choices of utility are used, typically functions of the posterior distribution of the model parameters. Bayesian experimental design aims to find the design which maximises the expected utility. This takes into account uncertainty by taking the expectation of the utility function over the model parameters and the data which could be observed. For this approach, difficulties arise as the expected utility is typically intractable and computationally costly to approximate.

Chaloner and Verdinelli (1995) provides a unified view of Bayesian optimal design from a decision theoretic viewpoint. The work of Müller (1999) has been widely extended by other authors as a basis for their work, for example Ryan et al. (2014) used a dimension reduction scheme for the design within the Müller algorithm and Amzal et al. (2006) extends the ideas of Müller using a particle based approach. More recently Overstall and Woods (2017) presented an algorithm considered state of the art capable of being applied in high dimensional design settings.

Many existing methods are unsuitable for modern applications as they scale poorly with the number of observations in a design. They may also require many utility evaluations, each of which can be computationally demanding to evaluate. A possible solution could be to develop methods which are scalable and efficient in terms of the number of utility evaluations required to find the optimal design. Alternatively computationally convenient utility functions could be considered to reduce the computational burden of implementing the optimal design methods.

Contribution

A promising solution to the problem of scalability in Bayesian optimal design methods could be stochastic gradient optimisation. This has been well studied and widely used in the field of machine learning, often to fit many parameters of a neural network. These methods should easily translate to the design optimisation problem for high dimensional designs as when an estimate of the gradient of the expected utility with respect to the design can be made, stochastic gradient optimisation can be used to traverse the design space to converge to an optimum. This thesis considers this approach and compares the results to existing methods.

Computation of the utility function is often expensive. Those which are a function of the posterior distribution often require costly inference methods like Monte Carlo

approximations. This computational burden is amplified due to each estimated expected utility requiring many realised utilities (used within another Monte Carlo estimate) and many methods needing multiple approximations of the expected utility. This thesis considers utility functions based on the Fisher information matrix in Bayesian optimal design. These are often available in closed-form so are fast to compute.

An alternative approach could be to use a surrogate function for the expensive utility computations. These offer a way of obtaining approximations to the utility that are cheap to evaluate. The surrogate can be used as a proxy and the design which is optimal under the surrogate could provide an approximation of the true optimal design. Furthermore the surrogate could provide a way of identifying regions of the design space where the expected utility is high. This could offer improvements in the efficiency of finding the optimal design by only considering promising regions when using expensive utility evaluations.

Outline of thesis

The structure this thesis is as follows. **Chapter 2** gives some background information that will be used throughout this thesis. An overview of Bayesian parameter inference is provided alongside some asymptotic results. The Fisher information (Fisher, 1925) is described in detail and defined for some distributions. These results will be used later in Chapter 4. Next, an introduction to experimental design is given alongside a short discussion on utility functions. Scoring rules are introduced and linked to utility functions using a Bayesian decision theoretic approach. Finally, a brief literature review is conducted. This describes the Müller and ACE algorithms in detail as these are used as benchmarks later in the thesis. A short overview is provided for some other algorithms that target the optimal design.

Chapter 3 considers stochastic gradient descent (SGD) for optimal design. First, background information is provided giving a roadmap of how the basic vanilla SGD update step is developed to incorporate features that help its performance in converging to the optimal design and are illustrated with an example. For many applications designs are subject to constraints. Including these into the SGD algorithm is described before estimation of the gradient and implementation is discussed. Finally, the application of SGD methods for optimal design is described.

In an optimal design problem the worth of a design has to be quantified. **Chapter 4** first considers some commonly used utility functions in Bayesian design problems. The Fisher information gain (FIG) utility is defined and shown to have a derivation from a Bayesian decision theoretic viewpoint. The FIG utility has some undesirable properties so some modifications are described which aim to address these issues. Finally, a utility based on an optimality condition used in classical design is defined. Although considered *pseudo*

Bayesian by the definitions of Ryan et al. (2016) this will be used to compare the optimal designs under different utility functions in subsequent chapters.

Chapter 5 looks at the designs returned for some example models under different utility functions. The first of these models, the simple death model, is used to verify that SGD does target the correct design which optimises the expected utility. The second, a geospatial model, investigates the performance of SGD to the current "state-of-the-art" ACE algorithm in a setting where there are a large number of highly correlated observations. **Chapter 6** conducts a simulation study using a pharmacokinetic model. This compares the designs returned under different utility functions and also assesses the designs output from the SGD, ACE and Müller algorithms.

Chapter 7 investigates the use of a surrogate utility function within optimal design. Some background and discussion on using neural networks to estimate a posterior distribution is given. This is then used to construct a surrogate of the utility function. The surrogate utility is then used within a delayed acceptance Markov chain Monte Carlo scheme in the Müller algorithm to investigate if there are any improvements in efficiency. The surrogate can also be used to identify good regions of the design space, those which yield a high expected utility, to reduce the size of the search space when implementing an algorithm using expensive utility realisations. This could potentially lead to efficiency improvements as the more expensive utility realisations are not being evaluated in the sub-optimal region of the design space.

Finally, **Chapter 8** summarises the findings of this thesis and gives suggestions for directions of future work.

Chapter 2

Background

This chapter will present the background material used in this thesis. First, Bayesian inference and Fisher information are introduced in Section 2.1 and Section 2.2 before describing the field of experimental design in Section 2.3. Section 2.4 outlines some commonly used utility functions. Finally, existing methods for Bayesian optimal design are discussed in Section 2.6.

2.1 Bayesian parameter inference

Bayesian inference aims to learn about some unknown parameters when data becomes available. Throughout this thesis a model for the data is assumed with an associated likelihood function which gives the density of the data conditional on the unknown parameters. In the Bayesian framework (Bernardo and Smith, 1994; O'Hagan and Forster, 2004) all unknown parameters are considered to be random variables. The beliefs about these variables are summarised through distributions and are updated when data is collected. The update procedure involves taking a prior distribution, $\pi(\theta)$, (which summarises the beliefs about parameters θ before any data is collected) and combining this with data y observed at design τ via the likelihood function $f(y|\theta;\tau)$ (a function that relates the parameters to the data) to obtain a posterior distribution $\pi(\theta|y;\tau)$ representing the updated beliefs. Throughout this thesis the unknown model parameters are considered to be continuous random variables and hence $\pi(\cdot)$ will be a probability density function.

Bayes theorem allows the posterior distribution to be expressed as

$$\pi(\theta|y;\tau) = \frac{\pi(\theta, y;\tau)}{\pi(y;\tau)} = \frac{\pi(\theta)f(y|\theta;\tau)}{\pi(y;\tau)}, \tag{2.1}$$

where $\pi(y;\tau) = \int_\Theta \pi(\theta) f(y|\theta;\tau)\ d\theta$ is the marginal likelihood or normalising constant, ensuring the posterior is a proper distribution ($\int_\Theta \pi(\theta|y;\tau)\ d\theta = 1$). Since the denominator in Equation 2.1 is constant with respect to θ the posterior is proportional to the prior times the likelihood,

$$\pi(\theta|y;\tau) \propto \pi(\theta) f(y|\theta;\tau). \tag{2.2}$$

Typically, the marginal likelihood is rarely tractable in the sense that it is not available in a closed form. Methods such as Markov chain Monte Carlo and importance sampling are often used to approximate the posterior however they are often costly in terms of time and computing resource. See Gelman et al. (2013) for more details of these and other related methods.

2.1.1 Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) methods allow sampling from a distribution that is not available in closed-form. This methodology has been well studied especially in the field of Bayesian statistics (see, for example, Brooks et al., 2011, Robert and Casella, 2013 and Kruschke, 2014). MCMC methods work by constructing a Markov chain with stationary distribution $\pi(\cdot)$ corresponding to the target distribution of interest. In the Bayesian setting this is chosen to be the posterior $\pi(\theta|y;\tau)$. Given any initial starting state the chain should converge to the stationary distribution if run long enough. The chains are liable to converging to local modes of the target distribution and so multiple chains are often run to see if they converge to the same region. Once converged the chain can be run for more iterations to give draws from the target distribution. These draws will be correlated and so independent draws are produced by thinning the output in accordance with the autocorrelation of the sample.

Metropolis-Hastings algorithm

The Metropolis-Hastings (MH) algorithm is a widely used MCMC scheme. It was first proposed by Metropolis et al. (1953) and later generalised by Hastings (1970). This algorithm works by proposing a new state θ^* from the current state θ according to a proposal density $q(\theta^*|\theta)$. The proposed θ^* is then accepted according to an acceptance probability which depends on the (proportional) target density and the proposal density. Otherwise, the chain stays at the current θ. The sequence $\theta^{(1)}, \theta^{(2)}, \ldots$ form a Markov chain where the target is the stationary distribution. Algorithm 2.1 describes the MH algorithm.

The choice of proposal density is arbitrary however it does impact the efficiency of the chain so is an important tuning consideration. A popular choice is to select a symmetric

Algorithm 2.1 Metropolis-Hastings Markov chain Monte Carlo algorithm targetting the posterior for θ given data y observed at design τ.

1: Initialise $\theta^{(0)}$.
2: **for** $i = 1, 2, \ldots, N$ **do**
3: Propose $\theta' \sim q(\theta|\theta^{(i-1)})$.
4: Calculate the acceptance probability, $\alpha = \min(1, A)$ where

$$A = \frac{\pi(\theta')f(y|\theta';\tau)q(\theta^{(i-1)}|\theta')}{\pi(\theta^{i-1})f(y|\theta^{(i-1)};\tau)q(\theta'|\theta^{(i-1)})}$$

5: With probability α, set $\theta^{(i)} = \theta'$. Otherwise, set $\theta^{(i)} = \theta^{(i-1)}$.
6: **return** $\theta^{(0)}, \theta^{(1)}, \ldots, \theta^{(N)}$

proposal density as this simplifies calculation of the acceptance probability. Often a Normal random walk is used, i.e. $\theta^*|\theta \sim N(\theta, \Sigma)$. In this case Σ is selected to give a sufficiently high acceptance rate.

2.1.2 Importance sampling

Importance sampling is a technique which allows estimation of properties of a distribution, in particular a posterior distribution. This is achieved by sampling points from a proposal distribution and computing importance weights. It is particularly useful in finding $\mathbb{E}_{\theta\sim\pi(\theta|y;\tau)}[g(\theta)]$ for some function $g(\cdot)$ when $\pi(\theta|y;\tau)$ is difficult to sample from. Consider draws of θ from a proposal density q. The quantity of interest can then be expressed as

$$\mathbb{E}_{\theta\sim\pi(\theta|y;\tau)}[g(\theta)] = \int_\Theta g(\theta)\pi(\theta|y;\tau)d\theta, \tag{2.3}$$

$$= \int_\Theta g(\theta)\frac{\pi(\theta|y;\tau)}{q(\theta)}q(\theta)d\theta, \tag{2.4}$$

$$= \mathbb{E}_{\theta\sim q(\theta)}\left[\frac{g(\theta)\pi(\theta|y;\tau)}{q(\theta)}\right]. \tag{2.5}$$

Since samples from q are obtainable the expectation in Equation 2.5 can be estimated using Monte Carlo integration,

$$\mathbb{E}_{\theta\sim\pi(\theta|y;\tau)}[g(\theta)] \approx \frac{\sum_{i=1}^N g\big(\theta^{(i)}\big)w^{(i)}}{\sum_{i=1}^N w^{(i)}}, \tag{2.6}$$

where each $\theta^{(i)}$ is a draw from the proposal with associated importance weight $w^{(i)} = \pi\big(\theta^{(i)}|y;\tau\big)/q\big(\theta^{(i)}\big)$ (Monahan, 2011). Note that since the importance weights appear in both the numerator and denominator in Equation 2.6 they can be calculated up to a

Algorithm 2.2 Importance sampling

1: From a proposal, q, obtain N draws, $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(N)}$.
2: Calculate $\tilde{w}^{(i)} = \frac{\pi(\theta^{(i)}) f(y|\theta^{(i)};\tau)}{q(\theta^{(i)})}$ for $i = 1, 2, \ldots, N$.
3: **return** $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(N)}$, $\tilde{w}^{(1)}, \tilde{w}^{(2)}, \ldots, \tilde{w}^{(N)}$

constant of proportionality, i.e. using Equation 2.2, the weights $\tilde{w} = \pi(\theta) f(y|\theta;\tau)/q(\theta)$ give an equivalent value for the expectation in Equation 2.6.

The output of the importance sampling routine (Algorithm 2.2) depends upon the choice of proposal density through $\tilde{w}$. Importance sampling is not useful if these importance weights vary substantially (Gelman et al., 2013). If many of the weights are small and a few large then the approximation in Equation 2.6 will have a high variance. A widely used diagnostic measure is the effective sample size (ESS) (Martino et al., 2017), given by

$$\text{ESS} = \frac{\left(\sum_{i=1}^{N} \tilde{w}^{(i)}\right)^2}{\sum_{i=1}^{N} (\tilde{w}^{(i)})^2}, \tag{2.7}$$

which takes values in the range $[0, N]$. Larger values of the ESS are preferred.

In order to achieve a large effective sample size the proposal distribution should be a good approximation of the posterior, i.e. $\pi(\theta|y;\tau)/q(\theta) \approx 1$. If the proposal density differs significantly from that of the posterior then the importance sampling routine will not be efficient and the results untrustworthy. A low effective sample size can be achieved through choice of an unsuitable proposal.

Often an initial choice of proposal could be the prior density, $q(\theta) = \pi(\theta)$. In this case the weights are simple to calculate and are equal to the likelihood, $\tilde{w} = f(y|\theta;\tau)$. If the posterior does not differ substantially from the prior distribution then the ESS should be sufficiently high.

2.1.3 Bayesian asymptotics

The mode and curvature of the posterior distribution can be approximated analytically using a multivariate normal distribution (Berger, 2013) under easily satisfied assumptions (Chaloner and Larntz, 1989), namely, the likelihood is a continuous function of θ and the mode $\hat{\theta}$ does not lie on the boundary of the parameter space Θ (Davison, 2003). Often this is referred to as the Laplace approximation as it follows from work in Laplace (1810).

Consider a Taylor expansion of the logged posterior about the parameters which maximise the likelihood function $\hat{\theta}$,

$$\log\left(\pi(\theta|y;\tau)\right) \approx \pi(\hat{\theta}|y;\tau) + (\theta - \hat{\theta})\nabla_\theta \log \pi(\hat{\theta}|y;\tau) + \frac{1}{2}(\theta - \hat{\theta})^T H(\hat{\theta})(\theta - \hat{\theta}), \tag{2.8}$$

where $\nabla_\theta = (\frac{\partial}{\partial\theta_1}, \frac{\partial}{\partial\theta_2}, \ldots, \frac{\partial}{\partial\theta_p})$ takes partial derivatives with respect to each parameter and $H(\hat{\theta}) = \nabla_\theta \nabla_\theta^T \log\big(\pi(\hat{\theta}|y;\tau)\big)$ is the Hessian matrix of second derivatives of the log posterior evaluated at $\theta = \hat{\theta}$. Note that the mode is invariant under multiplicative and log transformations of $\pi(\theta|y;\tau)$ so $\hat{\theta}$ is easily found by maximising the logged prior distribution multiplied by the likelihood, $\log(\pi(\theta)f(y|\theta;\tau))$. Since $\hat{\theta}$ is a maximum, $\nabla\pi(\theta|y;\tau) = 0$ meaning Equation 2.8 can be expressed as

$$\log\big(\pi(\theta|y;\tau)\big) \approx \log(k) - \frac{1}{2}(\theta - \hat{\theta})^T\big[-H(\hat{\theta})\big](\theta - \hat{\theta}), \tag{2.9}$$

where k is constant with respect to θ.

Upon exponentiating Equation 2.9 becomes

$$\pi(\theta|y;\tau) \approx k e^{-\frac{1}{2}(\theta-\hat{\theta})^T\left[-H(\hat{\theta})\right](\theta-\hat{\theta})}, \tag{2.10}$$

showing that the posterior distribution is approximately a multivariate normal distribution with mean $\hat{\theta}$ and covariance matrix $-H(\hat{\theta})^{-1}$.

For a large number of samples $y_1, y_2, \ldots, y_d$, it can be shown that the posterior distribution concentrates mass in smaller and smaller neighbourhoods of θ_0 and that $|\hat{\theta} - \theta_0| \to 0$ as $d \to \infty$ (Gelman et al., 2013). In this case the posterior distribution is dominated by the likelihood and the Hessian matrix becomes

$$\Big[H(\theta)\Big]_{\theta=\hat{\theta}} = \Big[\nabla_\theta\nabla_\theta^T \log\big(\pi(\theta)\big)\Big]_{\theta=\hat{\theta}} + \sum_{i=1}^{d}\Big[\nabla_\theta\nabla_\theta^T \log\big(f(y_i|\theta;\tau)\big)\Big]_{\theta=\hat{\theta}}. \tag{2.11}$$

This is dominated by the summation term when d is large. For $\hat{\theta}$ close to θ_0 the curvature of the posterior can be approximated by the Fisher information matrix (see Section 2.2) evaluated at $\hat{\theta}$ (Gelman et al., 2013).

Asymptotically, a multivariate normal distribution centred at $\hat{\theta}$ with covariance matrix equal to the inverse of the Fisher information matrix evaluated at the $\hat{\theta}$, $\mathcal{I}(\hat{\theta};\tau)$, can be used as an approximation of the posterior, i.e. $\theta|y;\tau \sim N_p\big(\hat{\theta}, \mathcal{I}(\hat{\theta};\tau)^{-1}\big)$.

2.2 Fisher information

The Fisher information, defined by Fisher (1925), provides a measure of how much information data y carries about parameters θ. The Fisher score is defined as the first derivative of the log likelihood function with respect to the parameters (Lindgren, 1993),

$$u(\theta, y;\tau) = \nabla_\theta \log f(y|\theta;\tau). \tag{2.12}$$

The Fisher information matrix is then defined in terms of the Fisher score,

$$\mathcal{I}_\theta(\theta;\tau) = \mathbb{E}_{y\sim f(y|\theta;\tau)}[u(\theta,y;\tau)u(\theta,y;\tau)^T]. \tag{2.13}$$

For certain models the information matrix is available in a closed form making it computationally convenient. The Fisher information matrix can equivalently be expressed in terms of the second derivatives of the log likelihood under some regularity conditions,

$$\mathcal{I}_\theta(\theta;\tau) = -\mathbb{E}_{y\sim f(y|\theta;\tau)}[\nabla_\theta \nabla_\theta^T \log f(y|\theta;\tau)]. \tag{2.14}$$

The following argument summarises the relationship between Equation 2.13 and Equation 2.14. First note that for $f = f(y|\theta;\tau)$

$$\begin{aligned}
\mathbb{E}_{y\sim f(y|\theta;\tau)}\left[\frac{1}{f}\nabla_\theta\nabla_\theta^T f\right] &= \int_{\mathcal{Y}} \frac{1}{f}\{\nabla_\theta\nabla_\theta^T f\} f dy, && (2.15)\\
&= \nabla_\theta\nabla_\theta^T \int_{\mathcal{Y}} f\ dy, && (2.16)\\
&= \nabla_\theta\nabla_\theta^T 1, && (2.17)\\
&= 0. && (2.18)
\end{aligned}$$

The swapping of the order of the differentiation and integration requires some regularity conditions (Klenke, 2013, Theorem 6.28). Lee (2012) argues these conditions are satisfied in any reasonable case. From Equation 2.14,

$$\begin{aligned}
\mathcal{I}_\theta(\theta;\tau) &= -\mathbb{E}_{y\sim f(y|\theta;\tau)}\left[\nabla_\theta\nabla_\theta^T \log f\right], && (2.19)\\
&= -\mathbb{E}_{y\sim f(y|\theta;\tau)}\left[\nabla_\theta\left(\frac{1}{f}\nabla_\theta^T f\right)\right], && (2.20)\\
&= -\mathbb{E}_{y\sim f(y|\theta;\tau)}\left[\frac{1}{f}\nabla_\theta\nabla_\theta^T f - \frac{1}{f^2}\nabla_\theta f\nabla_\theta^T f\right], && (2.21)\\
&= -\mathbb{E}_{y\sim f(y|\theta;\tau)}\left[\frac{1}{f}\nabla_\theta\nabla_\theta^T f\right] + \mathbb{E}_{y\sim f(y|\theta;\tau)}\left[\left(\frac{1}{f}\nabla_\theta f\right)\left(\frac{1}{f}\nabla_\theta^T f\right)\right], && (2.22)\\
&= \mathbb{E}_{y\sim f(y|\theta;\tau)}\left[(\nabla_\theta \log f)(\nabla_\theta^T \log f)\right], && (2.23)\\
&= \mathbb{E}_{y\sim f(y|\theta;\tau)}[u(\theta,y;\tau)u(\theta,y;\tau)^T], && (2.24)
\end{aligned}$$

thus showing the equivalence to Equation 2.13. Note that the definition of the Fisher information matrix using the Fisher score is preferred as it does not require any regularity conditions to be met or that the second derivatives exist.

Consider the case where a model with parameters θ is subject to a reparameterisation $\xi(\theta)$. The alternative parameterisation may be of the same or different length to the

original parameters. Under this transformation element j of the Fisher score (Equation 2.12) becomes

$$u_j(\theta, y; \tau) = \frac{\partial}{\partial \theta_j} \log f(y|\theta; \tau), \tag{2.25}$$

$$= \sum_i \frac{d\xi_i}{d\theta_j} \frac{\partial \log f}{\partial \xi_i}, \tag{2.26}$$

and hence

$$\nabla_\theta \log f(y|\theta; \tau) = J(\xi) \nabla_\xi \log f(y|\xi; \tau), \tag{2.27}$$

where $J(\xi)$ is the Jacobian matrix of the transformation with ij^{th} element equal to $\frac{\partial \xi_i}{\partial \theta_j}$. Using Equation 2.13 the Fisher information matrix can then be expressed as

$$\mathcal{I}_\theta(\theta; \tau) = \mathbb{E}_{y \sim f(y|\theta;\tau)}[\nabla_\theta \log f (\nabla_\theta \log f)^T], \tag{2.28}$$

$$= \mathbb{E}_{y \sim f(y|\theta;\tau)}[J(\xi) \nabla_\xi \log f (J(\xi) \nabla_\xi \log f)^T], \tag{2.29}$$

$$= J(\xi) \mathbb{E}_{y \sim f(y|\theta;\tau)}[\nabla_\xi \log f (\nabla_\xi \log f)^T] J(\xi)^T, \tag{2.30}$$

$$= J(\xi) \mathcal{I}_\xi(\xi; \tau) J(\xi)^T. \tag{2.31}$$

2.2.1 Binomially distributed random variables

Consider a random variable $Y = (Y_1, Y_2, \ldots, Y_d)$ observed at times $\tau = (\tau_1, \tau_2, \ldots, \tau_d)$. The observations are independent and identically distributed according to a binomial distribution,

$$Y_i \sim Bin(n, \alpha_i), \tag{2.32}$$

where n is the initial population size and the probability of an event occurring $\alpha_i = \alpha(\tau_i, \theta)$ depends on the design τ_i and a parameter θ. The log-likelihood of this distribution is given by

$$\ell = \sum_{i=1}^{d} \log \binom{n}{y_i} + y_i \log(\alpha_i) + (n - y_i) \log(1 - \alpha_i), \tag{2.33}$$

thus the score is given by

$$u(\theta, y; \tau) = \frac{d\ell}{d\theta}, \tag{2.34}$$

$$= \sum_{i=1}^{d} \frac{d\ell_i}{d\theta}, \tag{2.35}$$

$$= \sum_{i=1}^{d} \sum_{j=1}^{d} \frac{d\alpha_j}{d\theta} \frac{\partial \ell_i}{\partial \alpha_j}. \tag{2.36}$$

$$\tag{2.37}$$

Since the observations are independent $\frac{\partial \ell_i}{\partial \alpha_j} = 0$ for $i \neq j$ and so

$$u(\theta, y; \tau) = \sum_{i=1}^{d} \frac{d\alpha_i}{d\theta} \frac{\partial \ell_i}{\partial \alpha_i}, \tag{2.38}$$

$$= \sum_{i=1}^{d} \frac{d\alpha_i}{d\theta} \left(\frac{y_i - n\alpha_i}{\alpha_i(1-\alpha_i)} \right). \tag{2.39}$$

For this example the score is a scalar value hence

$$\mathcal{I}(\theta; \tau) = \mathbb{E}_{y \sim f(y|\theta;\tau)}[u(\theta, y; \tau)^2]. \tag{2.40}$$

2.2.2 Normally distributed random variables

This section considers models where observations have Normal error structure,

$$Y \sim N_d\big(\mu(\theta, \tau), \Sigma(\theta, \tau)\big). \tag{2.41}$$

Both the mean vector μ and covariance matrix Σ depend on parameters θ and design τ. This allows errors to be correlated across observations so that those in close proximity will be similar.

Under this model, an analytic solution for the expectation over the data is obtainable, with the ij^{th} entry for the Fisher information matrix given by (Malagò and Pistone, 2015)

$$\mathcal{I}_{ij} = \frac{\partial \mu^T}{\partial \theta_i} \Sigma^{-1} \frac{\partial \mu}{\partial \theta_j} + \frac{1}{2} \text{tr} \left(\Sigma^{-1} \frac{\partial \Sigma}{\partial \theta_i} \Sigma^{-1} \frac{\partial \Sigma}{\partial \theta_j} \right), \tag{2.42}$$

where $\frac{\partial \Sigma}{\partial \theta_i}$ is a matrix of partial derivatives of Σ with respect to a parameter θ_i. Equation 2.42 gives the form of the Fisher information matrix however for various covariance structures this can be expressed in a simpler form.

Constant covariance with independent observations, $\Sigma = \sigma^2 \mathbb{I}_d$

In this simple case, the covariance matrix is proportional to the identity matrix $\mathbb{I}_d$. Therefore Σ is not dependent on the parameters θ and thus $\frac{\partial \Sigma}{\partial \theta_k} = 0$ for all k. The ij^{th} element of the Fisher information matrix is given by

$$\mathcal{I}_{ij} = \frac{\partial \mu^T}{\partial \theta_i} \Sigma^{-1} \frac{\partial \mu}{\partial \theta_j}. \tag{2.43}$$

The inverse of $\Sigma = \sigma^2 \mathbb{I}_d$ is given by $\Sigma^{-1} = \sigma^{-2}\mathbb{I}_d$ and so the ij^{th} element of the information matrix can be expressed as

$$\mathcal{I}_{ij} = \frac{1}{\sigma^2} \frac{\partial \mu^T}{\partial \theta_i} \frac{\partial \mu}{\partial \theta_j}. \tag{2.44}$$

Errors dependent upon mean function with independence across observations, $\Sigma = \sigma_1^2 \mathbb{I}_d + \sigma_2^2 diag(f(\mu))$

Consider a covariance matrix of the form $\Sigma = \sigma_1^2 \mathbb{I}_d + \sigma_2^2 diag(f(\mu))$ where errors are independent across designs but the magnitude of variation depends on some function of the mean value, $f(\mu) = (f(\mu_1), f(\mu_2), \ldots, f(\mu_d)) = (f_1, f_2, \ldots, f_d)$. The matrix of partial derivatives of Σ with respect to each element of θ is required in Equation 2.42. The derivative of Σ with respect to θ_k is given by

$$\frac{\partial \Sigma}{\partial \theta_k} = \sigma_2^2 \zeta_k, \tag{2.45}$$

where ζ_k is a matrix of partial derivatives of f with respect to θ_k,

$$(\zeta_k)_{ij} = \frac{\partial f_i}{\partial \mu_j} \frac{d\mu_j}{d\theta_k}. \tag{2.46}$$

Here ζ_k is a diagonal matrix as $\frac{\partial f_i}{\partial \mu_j} = 0$ for $i \neq j$ due to independence between observations.

As both Σ and ζ are diagonal matrices, the second term in Equation 2.42 can be expressed as follows.

$$\begin{aligned}
\text{tr}\left(\Sigma^{-1} \frac{\partial \Sigma}{\partial \theta_i} \Sigma^{-1} \frac{\partial \Sigma}{\partial \theta_j} \right) &= \text{tr}(\Sigma^{-1} \sigma_2^2 \zeta_i \Sigma^{-1} \sigma_2^2 \zeta_j), && (2.47) \\
&= \sigma_2^4 \sum_{k=1}^{d} \Sigma_{kk}^{-1} \left(\frac{\partial f_k}{\partial \mu_k} \frac{d\mu_k}{d\theta_i} \right) \Sigma_{kk}^{-1} \left(\frac{\partial f_k}{\partial \mu_k} \frac{d\mu_k}{d\theta_j} \right), && (2.48) \\
&= \sigma_2^4 \sum_{k=1}^{d} \Sigma_{kk}^{-2} \left(\frac{\partial f_k}{\partial \mu_k} \right)^2 \frac{d\mu_k}{d\theta_i} \frac{d\mu_k}{d\theta_j}. && (2.49)
\end{aligned}$$

The ij^{th} entry for the Fisher information matrix is hence given by

$$\mathcal{I}_{ij} = \frac{\partial \mu^T}{\partial \theta_i} \Sigma^{-1} \frac{\partial \mu}{\partial \theta_j} + \frac{\sigma_2^4}{2} \sum_{k=1}^{d} \Sigma_{kk}^{-2} \left(\frac{\partial f(\mu_k)}{\partial \mu_k} \right)^2 \frac{d\mu_k}{d\theta_i} \frac{d\mu_k}{d\theta_j}. \tag{2.50}$$

Correlated errors across observations, $\Sigma = \Sigma(\tau)$

Consider a covariance matrix which depends only on the design τ and not the parameters θ. The covariance matrix considered here has the form

$$\Sigma_{ij} = \begin{cases} \sigma_1^2 + \sigma_2^2, & i = j \\ \sigma_2^2 \, \rho(\tau_i, \tau_j), & i \neq j \end{cases} \tag{2.51}$$

where ρ is a correlation function. Examples of suitable correlation functions would be the Matérn family or squared exponential correlation. See, for example, Gaetan and Guyon (2010) for a more detailed description of correlation functions. The covariance is constant with respect to the model parameters and so the Fisher information matrix for this example takes the form of Equation 2.43. Here Σ is not a diagonal matrix and thus requires matrix inverse computations.

2.3 Experimental design

Many experiments are costly and expensive to conduct. Collection of data may involve an invasive procedure on subjects or require a substantial amount of resources often limiting the number of observations that can be made. For these reasons, practitioners are interested in maximising the value of the data with respect to the experimental aims. Optimal experimental design is a framework which aims to identify the optimal locations, over space and/or time, to observe data. It can also be used to make other design decisions such as treatment allocations.

2.3.1 Classical experimental design

In the classical setting, experimental design focuses on minimising the variance of the estimators of the unknown parameters. This corresponds to maximising the information present in the data. The quantitative summary of the worth of a design is often based on optimality criteria involving the Fisher information matrix (see Section 2.2) and are referred to as alphabet criteria due to the naming conventions as introduced by Kiefer (1959).

Classical alphabet optimality

Alphabet criteria have been well studied in the classical literature (Box, 1982; Hinkelmann and Maman, 2005; Atkinson et al., 2007). Consider the Fisher information matrix, $\mathcal{I}$, with non-zero eigenvalues, $\lambda_1, \lambda_2, \ldots, \lambda_p$. The following optimality criteria can then be defined.

***D* optimality** is when the determinant criterion is minimised. This can be expressed as minimising the product of non-zero eigenvalues,

$$\min \prod_{i=1}^{p} \frac{1}{\lambda_i}. \tag{2.52}$$

***A* optimality** corresponds to minimising the average variance criterion. The value of this criterion is equal to the inverse of the average or sum of the inverse eigenvalues of $\mathcal{I}$,

$$\min \sum_{i=1}^{p} \frac{1}{\lambda_i}. \tag{2.53}$$

The optimal design under A optimality corresponds to minimising the average variance of the parameter estimates.

***T* optimality** aims to maximise the trace of the information matrix (using the definition used by Sun and Sun, 2015 and Walker, 2016) , equivalent to the sum of its eigenvalues,

$$\max \sum_{i=1}^{p} \lambda_i. \tag{2.54}$$

Sun and Sun (2015) describe T optimality as an alternative to A optimality that is easier to compute. Note that this naming convention is sometimes ambiguous as most other authors (for example Atkinson and Fedorov, 1975, Dette et al., 2012 and Duarte et al., 2015) define T optimality as a criterion related to a test statistic, aiming to discriminate between models.

2.3.2 Bayesian experimental design

The Bayesian experimental design framework incorporates any prior knowledge and uncertainties by taking the expectation over the prior distribution for the model parameters $\pi(\theta)$ and the assumed model for the observations $f(y|\theta;\tau)$. The prior distribution can incorporate information from previous studies as well as any subjective beliefs or expertise elicited from the practitioner.

The aim of Bayesian experimental design is to make an optimal decision whilst accounting for the uncertainties about the model parameters and data which could be observed. The worth of observations taken at a particular design τ given data y is observed based on sampled parameters θ is quantified through a utility function which is to be optimised. In practice, a utility function $\mathcal{U}(\tau, \theta, y)$ quantifying the worth of an experiment should first

be elicited or chosen based on the aims of the experiment. Wolfson et al. (1996) present a discussion on the mechanics of eliciting a utility function however this is difficult to do in practice and so alternative generic utilities are often used. See Section 2.4 for further discussion. The prior distribution for the unknown parameters should be defined as well as a model describing how they relate to observations in the experiment. The Bayesian optimal design τ^* maximises the expected utility $\mathcal{J}(\tau)$ which is the prior predictive utility function,

$$\tau^* = \max_{\tau \in \mathcal{T}} \mathcal{J}(\tau), \tag{2.55}$$

where

$$\mathcal{J}(\tau) = \int_{\Theta} \int_{\mathcal{Y}} \mathcal{U}(\tau, \theta, y) \pi(\theta, y; \tau) dy d\theta. \tag{2.56}$$

Usually the solution to Equation 2.56 does not have a closed form hence τ^* is found using numerical optimisation, typically of an approximation of the expected utility. See Section 2.6.

2.4 Utility functions

A utility function $\mathcal{U}(\tau, \theta, y)$ is a quantitative summary of an outcome which reflects the aims of the experiment. It should reward desirable results whilst penalising any costs involved. The worth of a particular design is given by the expected utility $\mathcal{J}(\tau)$, the average utility weighted by the prior beliefs for parameters $\pi(\theta)$ and the assumed model $f(y|\theta; \tau)$ as defined in Equation 2.56. A design τ_1 provides more knowledge than design τ_2 if the expected utility is higher for τ_1 than it is for τ_2, i.e. $\mathcal{J}(\tau_1) > \mathcal{J}(\tau_2)$.

It is often difficult or infeasible to elicit a specific utility function from the practitioner and so more general functions concerning estimation of the model parameters or predictions from the model are often used. In the classical design setting utilities are usually based on alphabet optimality conditions based on the Fisher information matrix. Section 2.3.1 defines some of these criteria. Often these have analogues in the Bayesian setting where the classic alphabet criteria are averaged over Θ and $\mathcal{Y}$. Such utilities are termed pseudo-Bayesian by Ryan et al. (2016). To be a fully-Bayesian utility function Ryan et al. (2016) proposed that the design criterion should be a function of the posterior distribution. Section 2.5.1 describes a decision theoretic view of optimal design where certain pseudo-Bayesian utilities can be shown to be functions of the posterior and hence considered fully-Bayesian under the preceding definitions. Chapter 4 gives further details and also describes some commonly used Bayesian utility functions.

2.5 Scoring rules

Scoring rules provide a quantitative measure of the accuracy of probabilistic predictions. They originated as an approach to making and evaluating probabilistic predictions (Dawid, 2014). A score can be thought of as calibration measure, ensuring that the forecaster is honest with his predictions, not being under or over confident in their beliefs. Alternatively a score can be interpreted as a loss function where the minimum expected loss corresponds to reporting the true set of probabilities or distribution.

Consider a random variable θ with true distribution given by $P \in \mathcal{P}$ where $\mathcal{P}$ is a set of suitable distributions. In many cases P will be an unknown distribution which is estimated by a quoted distribution, $Q \in \mathcal{P}$. The expected score is the expected penalty between P and Q given by

$$S(P, Q) = \mathbb{E}_{\theta \sim p(\theta)}[s(\theta, Q)], \tag{2.57}$$

where s is a scoring rule which quantifies the penalty. This provides a method of quantifying differences between the distributions.

For different probabilistic forecasts $Q_1, Q_2 \in \mathcal{P}$, $S(P, Q_1) < S(P, Q_2)$ means Q_1 is a more accurate probabilistic forecast than Q_2 because Q_1 is closer to P. An expected score is called strictly proper if $S(P, Q) > S(P, P)$ for $Q \neq P$. A proper scoring rule is beneficial as it rewards choices of Q which are closer to the true distribution P.

Entropy and divergence are concepts which are closely related to proper scoring rules. Entropy is defined as the value of the expected score when the quoted distribution is the truth, $\mathcal{E}(P) = S(P, P)$. The difference in entropy between the true and quoted distributions, denoted $\mathcal{D}(P, Q) = S(P, Q) - \mathcal{E}(P)$, is referred to as the divergence.

The scoring rules used in this thesis, the logarithmic score (Good, 1992) and Hyvärinen score (Hyvärinen, 2005), are defined in Table 2.1. The results relating to the Hyvärinen score rely on the following regularity conditions.

1. $p(\theta)$ and $q(\theta)$ are twice differentiable with respect to θ.
2. $\mathbb{E}_{\theta \sim p(\theta)}\left[||\nabla \log p(\theta)||^2\right], \mathbb{E}_{\theta \sim q(\theta)}\left[||\nabla \log q(\theta)||\right]^2$ are finite.
3. $\nabla_\theta p(\theta) \to 0, \nabla_\theta q(\theta) \to 0$ for $||\theta|| \to \infty$.

There are many other scoring rules defined in the literature, for example Tsallis (1988); Brier (1950); Bregman (1967). Gneiting and Raftery (2007), Parry et al. (2012) and Dawid and Musio (2014) all present further discussion of scoring rules.

	Logarithmic score	Hyvärinen score
Scoring rule $s(\theta, Q)$	$-\log q(\theta)$	$2\Delta \log q(\theta) + \|\|\nabla \log q(\theta)\|\|^2$
Entropy	$-\mathbb{E}_{\theta\sim p(\theta)}[\log p(\theta)]$	$-\mathbb{E}_{\theta\sim p(\theta)}[\|\|\nabla \log p(\theta)\|\|^2]$
Divergence	$\mathbb{E}_{\theta\sim p(\theta)}[\log p(\theta) - \log q(\theta)]$	$\mathbb{E}_{\theta\sim p(\theta)}[\|\|\nabla \log p(\theta) - \nabla \log q(\theta)\|\|^2]$

Table 2.1: *Quantities associated with the logarithmic and Hyvärinen scores.*

2.5.1 Bayesian decision theoretic approach

Following from Minton et al. (1962), Lindley (1972) considered a decision theoretic approach to Bayesian experimental design. As in Section 2.3.2, a design $\tau \in \mathcal{T}$ will be selected and observations sampled given some model parameters drawn from the prior, $y \in \mathcal{Y}$ for $\theta \in \Theta$. Based solely on observing y (without knowledge of θ) the practitioner must choose an action $a \in \mathcal{A}$ where $\mathcal{A}$ is some set of possible actions. The benefit of taking this action is quantified by a general utility $\mathcal{V}(a, \tau, \theta, y)$. The utility $\mathcal{U}(\tau, \theta, y)$ required in Bayesian optimal design is then defined as taking the best action given y,

$$\mathcal{U}(\tau, \theta, y) = \mathcal{V}(\hat{a}, \tau, \theta, y), \tag{2.58}$$

where

$$\hat{a}(y; \tau) = \max_{a\in\mathcal{A}} \int_{\Theta} \mathcal{V}(a, \tau, \theta, y)\pi(\theta|y; \tau)d\theta. \tag{2.59}$$

The general utility could be elicited from the practitioner. However in practice this is difficult and instead a more generic function is often used. The action a could be a point estimate where $\mathcal{U}$ is quadratic loss i.e. $(a-\theta)^2$ for $\theta \in \Theta$ (Chaloner and Verdinelli, 1995). Bernardo (1979) considered the case where the action a is to estimate a density. The practitioner must quote a distribution for the model parameters after observing data in the experiment. For the practitioner to be honest the expected general utility, $\mathbb{E}_{\theta\sim\pi(\theta|y;\tau)}[\mathcal{V}(a, \tau, \theta, y)]$, must be maximised at the true posterior. This corresponds to the framework of proper scoring rules. Proper scoring rules (Section 2.5) are a class of functions which assess the quality of density estimates which the generalised utility can be based on.

The utility for the experiment $\mathcal{U}(\tau, y)$ (as defined in Equation 2.58) can be expressed as the difference between the score under prior uncertainty about θ and the score of the posterior distribution once data y has been observed,

$$\begin{aligned}
\mathcal{U}(\tau, y) &= \mathbb{E}_{\theta\sim\pi(\theta|y;\tau)}[s(\theta, \pi(\theta))] - \mathbb{E}_{\theta\sim\pi(\theta|y;\tau)}[s(\theta, \pi(\theta|y; \tau))] && (2.60)\\
&= S(\pi(\theta|y; \tau), \pi(\theta)) - S(\pi(\theta|y; \tau), \pi(\theta|y; \tau)) && (2.61)\\
&= \mathcal{D}(\pi(\theta|y; \tau), \pi(\theta)). && (2.62)
\end{aligned}$$

This comparison of the knowledge before and after the experiment makes it possible to discuss the information gained from the experiment (Lindley et al., 1956). The expected utility is thus given by

$$\mathcal{J}(\tau) = \mathbb{E}_{y\sim f(y|\theta;\tau)}\big[\mathcal{D}(\pi(\theta|y;\tau),\pi(\theta))\big]. \tag{2.63}$$

2.6 Existing methods for Bayesian optimal design

This section provides an overview of some existing algorithms used for Bayesian optimal design problems. These vary in complexity and efficiency thus provide a benchmark for assessing the performance of new methods of finding the optimal design. This chapter describes the Müller (Müller, 1999) and ACE (Overstall and Woods, 2017) algorithms in detail. In addition, brief details of other methods for finding the Bayesian optimal design are given.

2.6.1 Müller algorithm

The Müller algorithm (Müller, 1999) is a simulation based method of finding the optimal design based on Markov Chain Monte Carlo methods. Samples are drawn from the target density,

$$h(\tau,\theta,y) \propto \mathcal{U}(\tau,\theta,y)\pi(\theta,y;\tau), \tag{2.64}$$

using a Metropolis Hastings MCMC scheme (Algorithm 2.3, $M = 1$). The target distribution is defined such that the marginal density for τ is proportional to the expected utility $\mathcal{J}(\tau)$. The optimal design τ^* is then chosen to be the design which returns the mode for $h(\tau)$, the marginal density of $h(\tau,\theta,y)$ for τ. The mode can be approximated by fitting a suitable function to the MCMC output, for example a multivariate normal distribution, whose mode can be expressed analytically.

In order to make the mode more easily identifiable, Müller suggests making the target more peaked by powering up the distribution using a batch of M utility evaluations. The new target density is

$$h_M(\tau,\theta_1,\ldots,\theta_M,y_1,\ldots,y_M) \propto \prod_{m=1}^{M} \mathcal{U}(\tau,\theta_m,y_m)\pi(\theta_m,y_m;\tau). \tag{2.65}$$

The marginal density for τ is now proportional to $\mathcal{J}^M(\tau)$ for $M \in \mathbb{N}$. As the batch size, M, is increased the utility surface will become more peaked, thus allowing easier identification of the modal value. Algorithm 2.3 details this method.

Algorithm 2.3 Müller algorithm using MCMC to target the optimal design. The batch size, M, is the number of utility evaluations per iteration of the scheme.

1: Initialise $\tau^{(0)}$. Simulate M independent realisations, $\theta_1^{(0)}, \ldots, \theta_M^{(0)}$, with corresponding $y_1^{(0)}, \ldots, y_M^{(0)}$, from $\pi(\theta, y; \tau^{(0)})$.
2: Compute $H^{(0)} = \prod_{m=1}^{M} \mathcal{U}(\tau^{(0)}, \theta_m^{(0)}, y_m^{(0)})$.
3: **for** $i = 1, \ldots, N$ **do**
4: Propose $\tau' \sim q(\tau|\tau^{(i-1)})$ and simulate $(\theta_1', y_1'), \ldots, (\theta_M', y_M')$ from $\pi(\theta, y; \tau')$.
5: Compute $H' = \prod_{m=1}^{M} \mathcal{U}(\tau', \theta_m', y_m')$
6: Calculate the acceptance probability, $\alpha = \min(1, A)$ where

$$A = \frac{H' q(\tau^{(i-1)}|\tau')}{H^{(i-1)} q(\tau'|\tau^{(i-1)})}$$

7: With probability α, set $\tau^{(i)} = \tau'$ and $H^{(i)} = H'$. Otherwise, set $\tau^{(i)} = \tau^{(i-1)}$ and $H^{(i)} = H^{(i-1)}$.
8: **end**

Müller et al. (2004) suggest adapting Algorithm 2.3 by gradually incrementing M so that the target distribution changes over iterations, similar to simulated annealing (Van Laarhoven and Aarts, 1987). Thus the target distribution becomes more peaked throughout the iterations, helping to avoid becoming stuck in a local mode. In practice this requires more simulations of the utility in later iterations of the MCMC scheme. Theoretically, the marginal of τ of the powered up target $h_M(\tau)$ should converge uniformly to a Dirac delta function at the optimal design τ^* as $M \to \infty$ in the MCMC scheme in Algorithm 2.3.

The Müller algorithm has proven to be a popular choice when selecting a method for Bayesian optimal design. Many authors have employed similar strategies to solve design problems (Bielza et al., 1999; Müller et al., 2004; Cook et al., 2008). However, in practice, the algorithm has been found to be slow to converge and is particularly inefficient if either the design or parameter space are large (Stroud et al., 2001; Amzal et al., 2006).

Low dimension parametrisation of the design

Ryan et al. (2015) aimed to find a high dimensional optimal design using a dimension reduction scheme within the Müller algorithm. Such schemes typically characterise a design by a small number of parameters ν. Examples of such schemes include the Beta scheme and geometric scheme (Ryan et al., 2014). Parametrising the design has the benefit of reducing the dimension of the optimisation problem and hence the computational burden of finding the optimal design. However Overstall and Woods (2017) showed that the designs returned typically do not optimise the expected utility. The trade off between computational cost and design quality has to be considered when using low dimensional parametrisations.

2.6.2 Approximate co-ordinate exchange algorithm

The approximate co-ordinate exchange (ACE) algorithm (Overstall and Woods, 2017) is a more recently developed method aiming to find the optimal design. According to Overstall and McGree (2018), the ACE algorithm is "currently the state of the art in computing Bayesian designs for realistic-sized design spaces". In Overstall and Woods (2017), the authors demonstrated that in practice ACE returned better designs than Müller when using a dimension reduction scheme, with further improvements seen when searching over the original design space. The ACE algorithm is implemented in an `R` package, `acebayes` (Overstall et al., 2017), which calls functions in `C++` for increased speed. This makes the ACE algorithm easy to use. ACE can be applied to a wide range of design problems with flexibility in the choice of model and utility function. The design space can be continuous or discrete with the option of adding constraints.

The ACE algorithm uses a two stage method; co-ordinate exchange followed by point exchange. The first phase of the algorithm iteratively updates each component in the design in turn. This reduces the problem to a one dimensional optimisation for each element of the design. The update procedure fits a surrogate model, typically a Gaussian process, to the expected utility given that component of the design. The value which maximises this is then estimated and is accepted or rejected according to a Bayesian test of equality to see if the expected utility is improved. This uses a large number of simulations in the Monte Carlo approximation of the expected utility under the current and proposed designs. Phase 1 performs this process iteratively looping over each component of the design N_1 times.

Phase 2 of the ACE algorithm examines clusters of design points through use of a point exchange algorithm with candidate points formed from the components of the design output from phase 1. This tests if replication of some points is more beneficial than the current design. The component of the design whose replication yields the highest expected utility (according to a Monte Carlo estimate) is identified. This is then added to the design, resulting in a design containing too many points. A search for the design point whose deletion would lead to the lowest reduction in the estimated expected utility is conducted and removed to form the candidate design. The replication and deletion are jointly accepted or rejected, as in phase 1, according to a Bayesian test. This is repeated N_2 times, returning an estimated optimal design.

The ACE algorithm can be subject to converging to local optima and hence the authors suggest running the algorithm multiple times from different starting designs and selecting the design which has the highest expected utility, established using a Monte Carlo estimate with many simulations. This can be performed in parallel, reducing computation time if multiple cores are available. A further drawback of ACE is that it is inefficient for highly correlated designs as only one dimension is searched at a time.

Algorithm 2.4 The approximate co-ordinate exchange algorithm. The tuning parameters are: N_1 and N_2, the number of iterations of phase 1 and phase 2 respectively; b_1 and b_2, the number of simulations used to establish the candidate design and compute the associated acceptance probability; Q, the number points at which to evaluate the expected utility to fit the model in phase 1.

Choose an initial design $\tau^{(0)} = \left(\tau_1^{(0)}, \ldots, \tau_d^{(0)}\right) \in \mathcal{T}$. Set $\tau^{(C)} = \tau^{(0)}$.

Phase 1: *Co-ordinate exchange*

for $n = 1, \ldots, N_1$ **do**

  for $i = 1, \ldots, d$ **do**

    Select Q design points from the i^{th} margin of the design space, $\xi_i = \{\xi_{i,1}, \xi_{i,2}, \ldots, \xi_{i,Q}\}$.

    Approximate the expected utility via Monte Carlo using b_1 simulations, $\hat{\mathcal{J}}(\xi_{i,1}|\tau_{-i}^{(C)}), \ldots, \hat{\mathcal{J}}(\xi_{i,Q}|\tau_{-i}^{(C)})$, where $\tau_{-i}^{(C)} = (\ldots, \tau_{i-1}^{(C)}, \tau_{i+1}^{(C)}, \ldots)$ is the current design vector with the i^{th} component removed.

    Fit a model to $\left\{\xi_{i,q}, \hat{\mathcal{J}}(\xi_{i,q}|\tau_{-i}^{(C)})\right\}_{q=1}^{Q}$ and find the scalar value ξ_i^* which maximises it.

    Conduct a Bayesian test using b_2 simulated utilities to obtain a probability p_1 of accepting the proposed design $\tau' = (\tau_1, \ldots, \tau_{i-1}^{(C)}, \xi_i^*, \tau_{i+1}^{(C)}, \ldots, \tau_d)$.

    Set $\tau^{(C)} = \tau'$ with probability p_1.

Phase 2: *Point exchange*

for $m = 1, \ldots, N_2$ **do**

  for $j = 1, \ldots, d$ **do**

    Propose a point to be replicated

$$\tau_\kappa = \arg\max_{\tau_k} \hat{\mathcal{J}}(\tau_k|\tau^{(C)})$$

    for $\tau_k \in \tau^{(C)}$. Concatenate this to the current design $\tilde{\tau} = (\tau^{(C)}, \tau_\kappa)$. Note that $\tilde{\tau} \notin \mathcal{T}$.

    Find the element of $\tilde{\tau}$ whose removal yields the highest expected utility based on a Monte Carlo approximation using b_1 simulations,

$$\nu = \arg\max_{v} \hat{\mathcal{J}}(\tilde{\tau}_{-v}).$$

    Conduct a Bayesian test using b_2 simulated utilities to get probability p_2 of accepting proposed design $\tau' = \tilde{\tau}_{-\nu}$.

    Set $\tau^{(C)} = \tau'$ with probability p_2.

return $\tau^{(C)}$

2.6.3 Other optimal design algorithms

There are many more methods which target the optimal design. Chaloner and Verdinelli (1995) and more recently Ryan et al. (2016) give a detailed review of Bayesian experimental design in their respective papers. Some of these algorithms are described briefly below.

Amzal (Amzal et al., 2006) is a simulation based method to finding the optimal design. The algorithm extends the ideas of the Müller algorithm (Section 2.6.1) but uses a particle based approach. Like Müller it targets $h_M(\tau)$ (Equation 2.65). Amzal uses a sequential Monte Carlo approach. It targets $h_M(\tau)$ with M increasing during the algorithm. As M increases the distribution becomes more peaked around the mode τ^*.

The Amzal algorithm has two main advantages. The target distribution, $h_M(\tau)$, is sampled at each iteration therefore the design which returns the best utility can be estimated from each iteration as opposed to waiting for asymptotic behaviour. Furthermore, the algorithm adaptively targets the optimal design. That is, as the algorithm progresses designs are sampled more densely around the modes of the powered utility surface.

Gillespie Boys (Gillespie and Boys, 2019) is another particle based approach to the problem of Bayesian experimental design. This method considers designs on a discrete grid over the design space $\mathcal{T}$. Near continuous designs can be obtained using a fine grid. The algorithm simulates utility values at these designs, concentrating on those which give estimated expected utilities in the top $100\alpha\%$ of their distribution. The tuning parameter α is a sequence which decreases as the iterations increase causing the algorithm to focus on near optimal designs. Throughout the algorithm previous approximations of the expected utility are stored so that they can be combined with any new computed utilities via a weighted average.

The Gillespie Boys algorithm benefits from being more efficient under a fixed computational budget than other algorithms when the dimension of the design is small. The algorithm also performs well when the expected utility surface is multi-modal. Similarly to the Amzal method, the Gillespie Boys algorithm adaptively targets the optimal design through the iterations of the algorithm by concentrating on those which give a high estimated expected utility. In a high dimensional design space the algorithm does not perform as well. To explore this space for the optimal design a large number of points are required and thus the computational cost of the algorithm increases due to the number of utility realisations required. The computational burden of this algorithm is also dependent upon the coarseness of the discrete grid used; the finer the grid, the higher the computational cost and vice versa.

Induced natural selection heuristic (INSH) (Price et al., 2018) is a type of evolutionary algorithm (Goldberg, 1994) which allows the best designs to "survive" through evolutions/iterations. The algorithm is initialised by sampling a coarse grid of designs from the design space. At each iteration the expected utility is estimated at each design and the "best" are identified to become the set of accepted designs. The authors take the (user specified) r designs which give the highest expected utilities from the current and previous iterations. A new set of designs for the next iteration are proposed from a perturbation function. This is a probability distribution which adds noise to the accepted design. The algorithm then continues until it has exhausted the computational budget or has converged.

Chapter 3

Stochastic gradient descent for optimal design

Existing methods for optimal design problems typically require many evaluations of the utility function, usually scaling with the dimension of the design. Typically in Bayesian optimal design the utility is a function of the posterior distribution which can be computationally expensive to estimate. Due to this, efficient methods of finding the optimal design using as few utility evaluations as possible are desired.

Optimal design is an optimisation problem of the expected utility. This is a deterministic function but typically only stochastic estimates can be obtained. Stochastic optimisation methods could be employed as they offer a computationally efficient alternative to the existing methods. Used widely in the field of machine learning, stochastic optimisation algorithms can be used to search high dimensional spaces for values which optimise a function. In this chapter some stochastic gradient descent (SGD) algorithms are described which can be used to search for a design which maximises the expected utility function.

The stochastic gradient descent (SGD) algorithm aims to find $\phi^* \in \mathbb{R}^d$, which minimises a function, $g(\phi)$. SGD is particularly useful when $g(\cdot)$ cannot be evaluated but can be estimated. A selection of adaptations to the general framework are presented in Section 3.1.3 and Section 3.1.4 which aim to improve some characteristics of the vanilla SGD algorithm described in Section 3.1.2. Often constraints are required on ϕ. Section 3.2 outlines how these are applied to SGD. Finally, Section 3.3 describes methods of gradient computation.

3.1 Stochastic gradient descent

Stochastic gradient descent (SGD) methods aim to find the vector which minimises the objective function $g(\cdot)$,

$$\phi^* = \arg\min_{\phi} g(\phi).$$

Algorithm 3.1 General algorithm for SGD for n iterations.

1: Initialise ϕ_0 and set $i = 1$.
2: **while** $i \leqslant n$ **do**
3: $\quad \phi_i = \phi_{i-1} - \alpha_i h(\phi_{i-1})$
4: $\quad i := i + 1$
5: **return** ϕ_n

If the aim is to maximise a function then SGD can be applied to the negative of the objective function. This is known as stochastic gradient ascent. For convenience, throughout this thesis any stochastic gradient optimisation will be referred to as SGD.

SGD is an iterative procedure which uses gradient estimates of $g(\cdot)$ to find ϕ^*. The generic SGD algorithm is defined in Algorithm 3.1. After providing an initial state ϕ_0 it iteratively updates the current state ϕ_i by moving in the direction of $-\alpha_i h(\phi_{i-1})$, where α_i is called the learning rate and $h(\cdot)$ is a function of the gradient, or an estimate, at the previous state.

3.1.1 Gradient descent

First, consider a special case of SGD where the gradient of the objective function, ∇g, can be evaluated. The gradient descent algorithm (Curry, 1944) is applicable in this scenario and can be thought of as traversing the surface of the function downhill until a state where a gradient of zero is reached, i.e. a minimum. It is recovered when $h(\phi_{i-1}) = \nabla g(\phi_{i-1})$ in Algorithm 3.1, giving an update rule

$$\phi_i = \phi_{i-1} - \alpha_i \nabla g(\phi_{i-1}). \tag{3.1}$$

The objective function is subject to some regularity conditions that are typically met (Karimi et al., 2016). A sequence of learning rates α_i must be set by the practitioner. A typical choice is $1/i$. It can be shown that ϕ_i will eventually converge to a local minimum of the loss function (Bottou, 2010), if

$$\sum_{i=1}^{\infty} \alpha_i = \infty, \qquad \sum_{i=1}^{\infty} \alpha_i^2 < \infty, \tag{3.2}$$

based on the theory of Robbins and Monro (1951). In specific cases, such as when $g(\cdot)$ is a convex function, a function where a line drawn between any two points on the curve is wholly greater than or equal to the function, the algorithm will converge to a global minimum.

3.1.2 VANILLA SGD

In the case where only noisy estimates of the gradient of the objective function, ∇g, can be made, gradient descent (Section 3.1.1) is not applicable. Since ∇g cannot be calculated directly, stochastic gradient descent is an adaptation of gradient descent which uses an unbiased estimate of the gradient function $\hat{\nabla} g$. Specifically, $h(\phi_{i-1}) = \hat{\nabla} g(\phi_{i-1})$ in Algorithm 3.1, leading to a state update rule of

$$\phi_i = \phi_{i-1} - \alpha_i \hat{\nabla} g(\phi_{i-1}). \tag{3.3}$$

It can be shown that convergence holds subject to conditions outlined by Bottou et al. (2018).

DISCUSSION

Here disadvantages of the gradient descent and vanilla SGD algorithms are discussed before Section 3.1.3 and Section 3.1.4 present adaptive algorithms which aim to address some of the following issues.

Firstly, both converge to local minima rather than the desired global minimum (Chollet, 2018). This problem can be partially solved by initialising the algorithm at a variety of initial conditions. Comparing the value of $g(\cdot)$ at the final states will indicate which (locally minimal) final state is best however there is no guarantee of converging to the global minimum. Although this offers a solution to the issue of converging to a bad local minimum, it increases the computation burden and can become unfeasible dependent upon the dimension of the search space.

The second major disadvantage of these algorithms is that the practitioner has to select a suitable sequence of learning rates. Within the conditions for convergence (Equation 3.2) there still remains a wide variety of choice for α_i. A popular sequence chosen by practitioners is $1/i$. The choice of sequence affects the speed and stability of the convergence and so this is not suitable for all applications. Hence the practitioner is required to tune α_i which can be a difficult task. Bengio (2012) and Goodfellow et al. (2016) both describe the learning rate as the most important parameter which should always be tuned. Adaptations to SGD which claim to require less tuning are discussed in Section 3.1.4. Alternatively, online adaptation of the learning rate has been suggested (Baydin et al., 2017). This involves updating the learning rate alongside the state within the SGD algorithm. Wu et al. (2018) present a similar approach where the learning rate is initialised at a high value before decreasing its magnitude according to gradient observation made throughout the algorithm. Although a sequence is required to have convergence guarantees, in practice, α_i is often chosen to be fixed with favourable results.

A further disadvantage is that if the gradient estimates oscillate around zero, causing changes in the direction of the updates, the computational cost is increased as the algorithm requires longer to converge. This effect is exaggerated in cases where the objective function acts like a valley with steep sides. SGD using momentum (see Section 3.1.3) addresses this issue by using information from past gradient estimates to make updates.

3.1.3 SGD with momentum

Modifying the vanilla SGD algorithm to include *momentum* aims to address the issue of oscillatory behaviour. Momentum in the case of SGD incorporates a running average of past gradient estimates into the update step. If consecutive updates move in the same direction then the algorithm accumulates momentum and makes larger updates in that direction. If the direction of the gradients contradict each other then the updates will slow down by decreasing the momentum. For example this should happen close to a minimum of the objective function.

The update step for SGD with momentum (Qian, 1999) is given by

$$\phi_i = \phi_{i-1} - \alpha_i h(\phi_{i-1}), \tag{3.4}$$

where

$$h(\phi_{i-1}) = \beta h(\phi_{i-2}) + (1-\beta)\hat{\nabla} g(\phi_{i-1}). \tag{3.5}$$

Similarly to the vanilla SGD, using momentum also requires a sequence of learning rates, α_i, to be selected by the practitioner. This is typically chosen to be a constant value $\alpha_i = \alpha$. The weighting parameter, β, controls the decaying dependence on past gradient estimates. A typical choice for β is around 0.9 (Ruder, 2016).

Using momentum introduces the benefit of limiting the oscillatory behaviour exhibited by the state when updates are based on the direction of the estimated gradient, $\hat{\nabla} g$. This can contribute towards faster convergence (Rumelhart et al., 1986) which reduces the computational cost of optimisation. The benefits of including momentum are achieved when suitable hyper-parameters are selected. If bad values are chosen for β the updates may be too dependent on previous gradient estimates, destabilising convergence to the global optimum.

3.1.4 Adaptive moment estimation

The adaptive moment estimation (Adam) algorithm (Kingma and Ba, 2014) aims to build upon the idea of using momentum within the SGD algorithm by updating exponential

moving averages of the estimated gradient and squared gradient. This algorithm is widely used in the field of machine learning with well documented improvements in speed of convergence and loss obtained over alternative methods (Kingma and Ba, 2014; Ruder, 2016).

In Algorithm 3.1, the update step is defined for the Adam algorithm as

$$\phi_i = \phi_{i-1} - \alpha_i h(\phi_{i-1}), \tag{3.6}$$

where the gradient function $h(\cdot)$ is defined as

$$h(\phi_{i-1}) = \hat{m}_i / (\sqrt{\hat{v}_i} + \varepsilon), \tag{3.7}$$

with

$$m_i = \beta_1 m_{i-1} + (1 - \beta_1)\hat{\nabla} g(\phi_{i-1}), \tag{3.8}$$

$$\hat{m}_i = \frac{m_i}{1 - \beta_1^i}, \tag{3.9}$$

$$v_i = \beta_2 v_{i-1} + (1 - \beta_2)[\hat{\nabla} g(\phi_{i-1})]^2, \tag{3.10}$$

$$\hat{v}_i = \frac{v_i}{1 - \beta_2^i}, \tag{3.11}$$

where all operations are element-wise, α_i is again the learning rate and parameters β_1, β_2 and ε are constants. As in the other SGD algorithms, convergence is guaranteed for a decreasing sequence of learning rates, α_i. The authors use a fixed α which is shown to work well in practice (Kingma and Ba, 2014).

The gradient function of the Adam algorithm uses momentum for the first and second moments of the estimated gradient, $\hat{\nabla} g$. These are denoted m and v respectively with both initialised at a value of zero, i.e. $m_0 = v_0 = 0$. Both moments are updated by using a weighted average between the past moment estimates and the observed moment at the current state (see Equations 3.8 and 3.10). The dependence upon the past moment estimates is controlled by parameters β_1 and β_2 for the first and second moment respectively. The initialisation of the moments introduces a bias towards zero in both m_i and v_i which is corrected by Equation 3.9 and Equation 3.11. The bias corrected moments along with a further parameter, ε, used to ensure numerical stability, give the formulation of $h(\cdot)$ in Equation 3.7.

The parameters in the Adam algorithm are described with default values given in Table 3.1. The authors of the Adam algorithm show that the default values of the parameters are suitable for many situations and do not require a lot of tuning (Kingma and Ba, 2014).

Parameter	Description	Default value
α	Learning rate, a constant	0.001
β_1	Controls dependance on past m_i's	0.9
β_2	Controls dependance on past v_i's	0.999
ε	Ensures numerical stability in $h(\cdot)$	10^{-8}

Table 3.1: *Description and default values of the parameters in the Adam algorithm (Section 3.1.4).*

3.1.5 Other SGD algorithms

SGD algorithms are widely used in the field of machine learning and many further adaptations to the vanilla algorithm exist. Some of these algorithms project accumulated gradients forward before making a correction to obtain the next state as in the case of Nesterov accelerated gradient (Nesterov, 1983). Others, namely Adagrad (Duchi et al., 2011) and Adadelta (Zeiler, 2012) use adaptive procedures to update the learning rates based on all or some of the past squared gradients similarly to Adam. An overview of SGD methods is given by Ruder (2016).

3.1.6 Illustration of SGD algorithms

This section illustrates the dynamics of the vanilla SGD (Section 3.1.2), SGD with momentum (Section 3.1.3) and Adam (Section 3.1.4) algorithms when the objective function has a steep valley. The Rosenbrock function (Rosenbrock, 1960) is an example of such a function and is widely used to test the performance of optimisation algorithms. It is defined as

$$g(x, y) = (a - x)^2 + b(y - x^2)^2. \tag{3.12}$$

Here it is used to demonstrate the behaviour of the aforementioned algorithms.

Figure 3.1 shows that the vanilla SGD algorithm has highly oscillatory behaviour over the initial iterations. The valley in the objective function gives rise to this behaviour as consecutive gradient estimates have opposing signs in the y direction. Over the iterations the magnitude of the gradients decrease and so the algorithm finds the bottom of the valley and traverses this until it finds the minimum of the function.

For SGD with momentum a similar behaviour is observed however the oscillations are damped and so the algorithm finds the bottom of the valley quickly and moves along it towards the minimum at $(1, 1)$. The Adam algorithm again improves upon the oscillatory behaviour of both the vanilla SGD and SGD with momentum algorithms. It quickly finds the minimum of the valley and moves towards the minimum of the objective function.

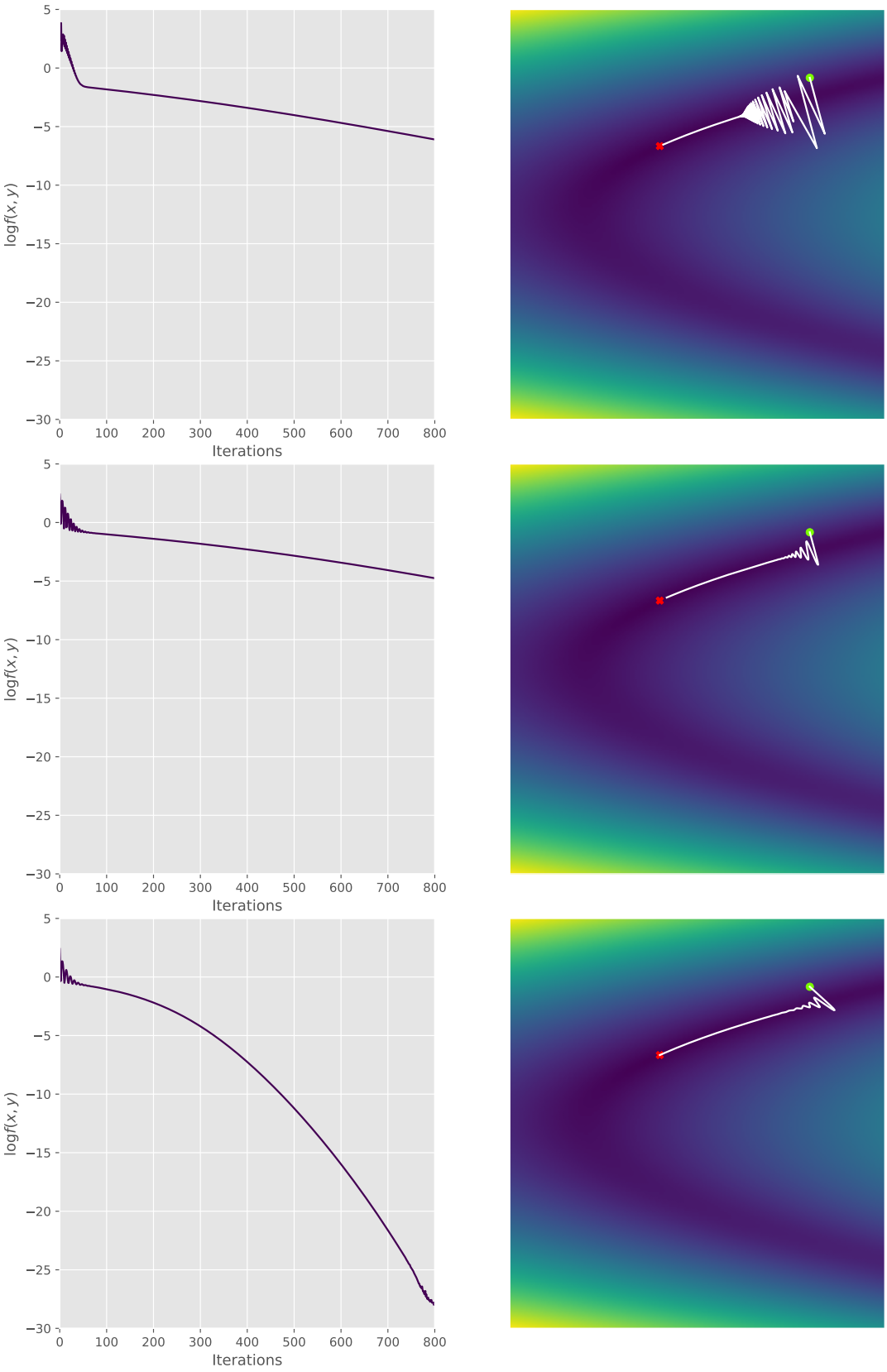

Figure 3.1: *The value of the objective function (left) and path (right) through the iterations of the vanilla SGD (top), SGD with momentum (middle) and Adam (bottom) algorithms where the objective function is the Rosenbrock function (Equation 3.12) with* $a = 1$ *and* $b = 10$*. All algorithms share the same initial state indicated by a green circle. The minimum of the function is at* $(1, 1)$*, represented by a red cross.*

All of the algorithms reduce the objective rapidly for the first 50 iterations, then more slowly thereafter. The trace plots of the objective function in Figure 3.1 show that the Adam algorithm is most efficient during this slower stage.

3.2 Constrained SGD

It is common for optimisation problems to be constrained in some way, for example $\phi_i \geqslant 0$ or, more generally, $\phi \in \mathcal{T} \subseteq \mathbb{R}^d$. In the context of optimal design, this could be due to restrictions on the design space or proximity of sampling times to each other. A simple solution to ensure the states of the algorithm are constrained to the permissible space is to include a penalty in the objective function. This aims to ensure the gradients move the states back to the feasible space, $\mathcal{T}$. Penalties can be incorporated additively, $\tilde{g}(\phi) = g(\phi) + \mathcal{P}(\mathcal{T}, \phi)$ or multiplicatively, $\tilde{g}(\phi) = g(\phi)\big(1 + \mathcal{P}(\mathcal{T}, \phi)\big)$, where $\tilde{g}$ is a penalised objective function which is to be minimised and $\mathcal{P}$ is a positive penalty function.

Throughout this thesis constraints on any optimisation are implemented by adding a large penalty to the loss function for any $\phi \notin \mathcal{T}$ where $\mathcal{T}$ is rectangular. Penalties can be used to encourage states to stay within the lower and upper boundaries, $\ell = (\ell_1, \ell_2, \ldots, \ell_d)$ and $\upsilon = (\upsilon_1, \upsilon_2, \ldots, \upsilon_d)$ respectively, or to encourage successive observations to be at least distance δ apart.

Lower bound:	$\mathcal{P}(\mathcal{T}, \phi) = k \max(\ell - \phi, 0),$	for $i = 1, 2, \ldots, d$
Upper bound:	$\mathcal{P}(\mathcal{T}, \phi) = k \max(\phi - \upsilon, 0),$	for $i = 1, 2, \ldots, d$
Separation:	$\mathcal{P}(\mathcal{T}, \phi) = k \max(\delta - D, 0),$	for $i = 1, \ldots, d-2, d-1$

where D is a matrix of absolute differences between states, $D_{ij} = |\phi_i - \phi_j|$, for $i, j = 1, 2 \ldots, d$.

Under this penalty, the practitioner has to select a value for k. The magnitude of $g(\cdot)$ should be considered when choosing k, ensuring that the penalty is large enough so that the gradient updates are directed back towards the permissible region, $\mathcal{T}$.

Simple penalties may not be effective in more complex settings and so more sophisticated methods can be used which project the state into the permissible space (Kushner and Yin, 2003).

3.3 Gradient estimation

SGD relies on obtaining an estimate of the gradient at the current state. In this section various methods that can be used to compute gradients are described. Further to this, software capable of implementing gradient computations is discussed.

3.3.1 Gradient computation

Implementing SGD requires estimating ∇g, a vector of derivatives of $g(\cdot)$. There are four main ways in which the derivative of a function can be computed: manual differentiation; symbolic differentiation; numerical differentiation; and automatic differentiation (Baydin and Pearlmutter, 2014).

Manual differentiation requires a practitioner to derive an expression for the derivative function. The advantage of this method is that, given the user makes no errors (a strong assumption for complex calculations), there are no issues with error due to approximation or numerical stability. Manual differentiation for more complex examples is rarely straightforward and so calculation of the exact derivative formula by the user can be infeasible or may be impossible due to intractability of the derivative.

Symbolic differentiation offers a method of obtaining an exact derivative without requiring the user to do any calculations manually. Using a set of programmed calculus rules, symbolic differentiation software takes an algebraic expression of the function and outputs an expression for the gradient. Symbolic differentiation output can produce excessively complex expressions of the gradient function, known as expression swell (Baydin et al., 2018). This leads to the computation becoming memory intensive and slow (Margossian, 2019).

Numeric differentiation computes the derivative through numerical methods such as using the Newton quotient (Adams, 2009), otherwise known as finite differencing.

$$\frac{\mathrm{d}g(x)}{\mathrm{d}x} \approx \frac{g(x+\delta) - g(x)}{\delta}. \tag{3.13}$$

Finite differencing offers fast computation of a simple function however the accuracy of the estimate is dependent upon δ. In the limit as $\delta \to 0$ the true gradient is recovered. In practice, if δ is too large, the estimated derivative will not be a good approximation of the true gradient. On the other hand if too small this can lead to large truncation error and issues with floating point error (Margossian, 2019).

Algorithm 3.2 Automatic differentiation of $g(x_1, x_2, \ldots, x_m)$ using forward propagation of gradients for a computational graph on N nodes, $n_1, n_2, \ldots, n_N$. The independent variables $x_1, x_2, \ldots, x_m$ and dependent variable g correspond to nodes $n_1, n_2, \ldots, n_m$ and n_N respectively. $Ch(n)$ is a function which gives the indices of the children of node n.

1: Fix the independent variable the derivative will be taken with respect to, here denoted x_j.
2: Set $v_1, \ldots, v_{j-1}, v_{j+1}, \ldots, v_m = 0$ and $v_j = 1$.
3: **for** $i = 2, \ldots, N$ **do**
4: Compute the value of the derivative at each node recursively,

$$v_i = \sum_{k \in Ch(n_i)} \frac{\partial n_i}{\partial n_k} v_k \tag{3.14}$$

5: **return** $\frac{\partial n_N}{\partial x_j} = v_N$

Automatic differentiation (AD) does not require analytical derivatives or manual input. Instead it decomposes the function to be differentiated into simple operations and uses standard differentiation rules to compute numerical values of the derivative vector. In contrast to numeric differentiation, derivatives returned by AD are free from truncation error (Griewank and Walther, 2008). Unlike symbolic differentiation, AD outputs a numeric value of the gradient rather than an expression. The code for AD is less restrictive than that of symbolic differentiation as it allows branching, loops, and recursion (Baydin et al., 2018).

The function that is to be differentiated is represented by a directed computational graph comprising of nodes $n_1, n_2, \ldots, n_N$ and edges. Typically the nodes represent variables but are labelled with the operation name. The inputs to the graph are the independent variable values $x_1, x_2, \ldots, x_m$ corresponding to nodes $n_1, n_2, \ldots, n_m$ and the output is the function value g corresponding to node n_N. Edges between nodes show where data is communicated through the graph, e.g. $x \to y$ mean that x is an input in the operation to compute y. The output of one node becomes the input for other nodes (Zaccone et al., 2017). See Figure 3.2 for an example.

Algorithm 3.2 and Algorithm 3.3 describe AD using forwards and backwards propagation of gradients respectively. The direction of the gradient propagation refers to the direction in which gradients are computed. Forwards indicates that the derivatives are taken with respect to an independent variable x_j and are computed recursively from the input nodes $n_1, n_2, \ldots, n_m$ in the direction of the edges in the graph until the derivative of the dependent variable, corresponding to node n_N, has been computed. Backwards propagation of gradients computes the derivative of g with respect to each node recursively starting at n_N and moving in the opposing direction to the edges of the graph. This gives the derivative of g with respect to each of the input nodes. AD using

Algorithm 3.3 Automatic differentiation of $g(x_1, x_2, \ldots, x_m)$ using backwards propagation of gradients for a computational graph on N nodes, $n_1, n_2, \ldots, n_N$. The independent variables $x_1, x_2, \ldots, x_m$ and dependent variable g correspond to nodes $n_1, n_2, \ldots, n_m$ and n_N respectively. $Pa(n)$ is a function which gives the indices of the parents of node n.

1: Fix the dependent variable g.
2: Set $v_N = 1$.
3: **for** $i = N-1, N-2, \ldots, 1$ **do**
4: Compute the value of the derivative at each node recursively,

$$v_i = \sum_{k \in Pa(n_i)} \frac{\partial n_k}{\partial n_i} v_k, \tag{3.15}$$

and store value v_i.
5: **return** $\frac{\partial g}{\partial n_1} = v_1, \frac{\partial g}{\partial n_2} = v_2, \ldots, \frac{\partial g}{\partial n_m} = v_m$

forward propagation of gradients is often subject to substantial memory costs. In contrast, propagating the gradients backwards addresses these issues thus is widely used in machine learning and AD software (Goodfellow et al., 2016).

Illustration

Consider a simple function with two input variables, $g(x, y) = e^x \sin(x + y)$. The computational graph is shown in Figure 3.2. Illustrations of local gradients in both the forwards and backwards cases of propagation of gradients are shown in Figure 3.3 and Figure 3.4 respectively. In this example $\frac{\partial g}{\partial x} = e^x\big(\sin(x + y) + \cos(x + y)\big)$. When using the forward propagation of gradients $\frac{\partial g}{\partial x}$ can be recovered by substitution of the relevant terms in n_7' from Figure 3.3. (Note that when using AD software only the numerical value would be given however the symbolic representation is used here to illustrate the example.) If $\frac{\partial g}{\partial y}$ was required then similar calculations would be required with $n_1' = 0$ and $n_2' = 1$. In contrast, propagating the gradients backwards is more efficient due to the output being the seed variable. As there is only one output of the graph the gradients with respect to each input variable can be recovered in a single pass through the graph. In Figure 3.4, substitution of the symbolic gradient expressions into n_1' and n_2' recover $\frac{\partial g}{\partial x}$ (as above) and $\frac{\partial g}{\partial y} = e^x \cos(x + y)$.

Summary

In summary, for complex functions, manual differentiation is infeasible. Numeric differentiation requires tuning of δ and can return estimates that are subject to

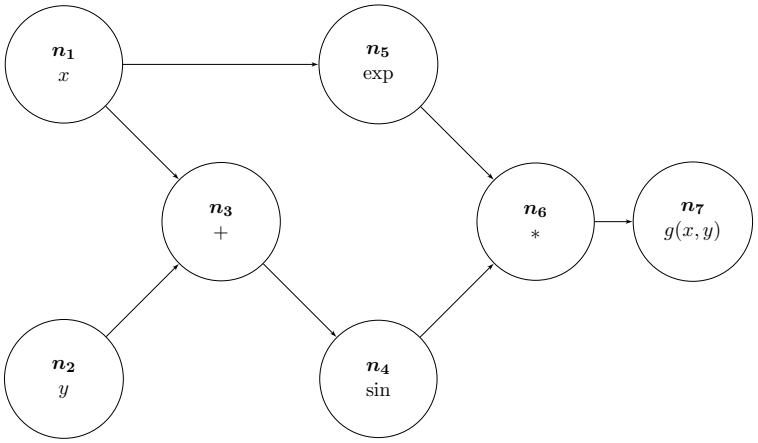

Figure 3.2: *Example of a computational graph used in automatic differentiation of the function* $g(x, y) = e^x \sin(x + y)$.

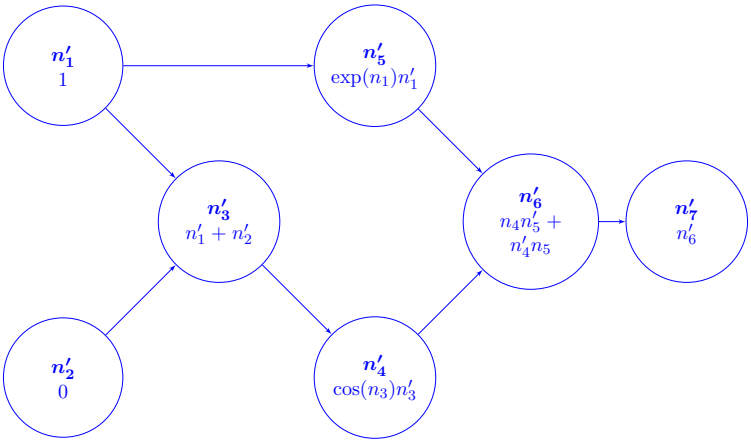

Figure 3.3: *Forward propagation of gradients with respect to* x. *If the partial derivative of* $g(\cdot)$ *with respect to* y *was required the seed variables would be set as* $n'_1 = 0$ *and* $n'_2 = 1$.

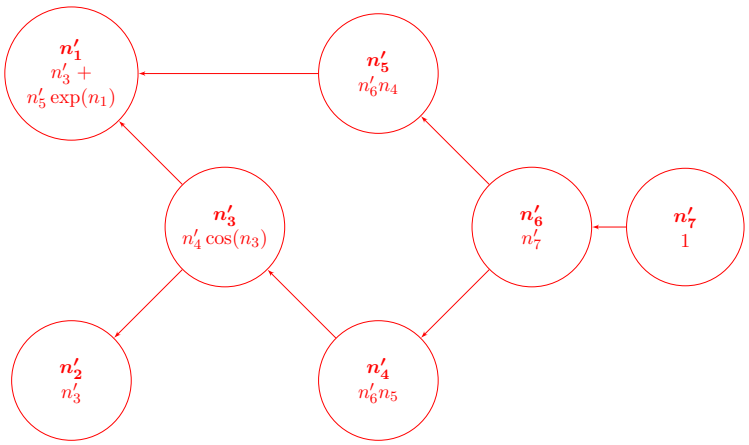

Figure 3.4: *Backward propagation of gradients for the function* $g(x, y) = e^x \sin(x + y)$.

Algorithm 3.4 Stochastic gradient optimisation of expected utility $\tilde{\mathcal{J}}(\tau) = \mathcal{J}(\tau) + \mathcal{P}(\tau)$.

1: Initialise design τ_0, number of iterations n, batch size K (used in gradient estimation).
2: **for** $t = 0, 1, 2, \ldots, n-1$ or until a convergence condition is reached **do**
3: Calculate g_t, an estimate of $\nabla\tilde{\mathcal{J}}(\tau)$ at $\tau = \tau_t$ using Equation 3.18.
4: Get τ_{t+1} by calling the update rule of a stochastic gradient optimisation algorithm with current state τ_t and gradient estimate g_t. The update rule of the Adam algorithm (Section 3.1.4) is used throughout this thesis.
5: **return** τ_n

truncation error. Although symbolic differentiation gives accurate expressions for the gradient, it can incur a high computation and memory overhead in comparison to automatic differentiation. Due to the ease of implementation and accuracy of the calculation, automatic differentiation with backwards propagation of gradients is the preferred method of calculating the gradient, ∇g, for SGD throughout this thesis.

3.3.2 Software choice

Automatic differentiation software will be utilised henceforth negating the need to carry out the gradient computations for implementing SGD manually. Many programmes are capable of automatic differentiation including Theano (Team, 2016), Torch (Collobert et al., 2011) and Tensorflow (Abadi et al., 2016).

Throughout this thesis SGD methods will be implemented using Tensorflow. Tensorflow is primarily a machine learning framework hence conveniently has many SGD algorithms, including Adam (Section 3.1.4), already implemented.

3.4 SGD applied to optimal design

Stochastic gradient descent methods can be applied to optimal design problems. The optimal design τ^* is an optimisation problem where only estimates of the expected utility function can be found. If $\tau \in \mathcal{T}$ where $\mathcal{T}$ is finite then a penalty term is added to the expected utility (see Section 3.2) and so the optimal design is found by maximising $\tilde{\mathcal{J}}(\tau) = \mathcal{J}(\tau) + \mathcal{P}(\tau)$.

If gradients of the expected utility can be estimated then SGD is applicable and will return a design corresponding to a local maxima of the expected utility. If the expected utility surface is convex then the global maximum will be found and τ^* will be returned. Algorithm 3.4 describes the process of applying SGD to the optimal design setting.

The dimension of the integration in the calculation of the expectation is often large and so the numerical approximation is commonly found using a Monte Carlo estimate,

$$\hat{\mathcal{J}}(\tau) = \frac{1}{K}\sum_{i=1}^{K}\mathcal{U}(\tau, \theta_i, y_i), \tag{3.16}$$

where (θ_i, y_i) is a draw from the prior $\pi(\theta)$ and simulated data from the model with parameters θ_i. Similarly, a Monte Carlo estimate is used for the gradients of the expected utility

$$\hat{\nabla}\mathcal{J}(\tau) = \frac{1}{K}\sum_{i=1}^{K}\nabla\mathcal{U}(\tau, \theta_i, y_i), \tag{3.17}$$

where $\nabla\mathcal{U}$ is found using automatic differentiation software as discussed in Section 3.3.2. Equation 3.17 is then used to compute the gradient of the expected utility including any penalisation,

$$\hat{\nabla}\tilde{\mathcal{J}}(\tau) = \nabla\mathcal{P}(\tau) + \hat{\nabla}\mathcal{J}(\tau), \tag{3.18}$$

where $\mathcal{P}(\tau)$ is a function defining the penalty at design τ.

This is a simple method to implement however it can become increasingly expensive computationally when an accurate estimate of the expected utility is required. The cost of this approximation also scales with respect to the dimensionality of the parameter space Θ and sample space $\mathcal{Y}$ however in practice the SGD algorithm copes well with more variable estimates of the expected utility gradient (see Section 6.2.2).

Chapter 4

Utility functions

This chapter considers utility functions used in Bayesian optimal design. First, Section 4.1 defines some commonly used utility functions. Next Section 4.2 derives the Fisher information gain utility from a decision theoretical viewpoint. Finally, a utility function relating to a classical alphabet optimality criterion is described in Section 4.3.

4.1 Common Bayesian utility functions

This section defines some of the most commonly used utilities in Bayesian experimental design, namely the posterior precision (for example Cook et al., 2008; Drovandi et al., 2013; Gillespie and Boys, 2019) and the Shannon information gain (for example Bernardo, 1979; Ryan, 2003; Overstall et al., 2019).

Posterior precision is a utility function that involves the posterior covariance matrix, defined as

$$\mathcal{U}_P(\tau, \theta, y) = |\Sigma|^{-1}, \tag{4.1}$$

where $\Sigma = \text{Var}(\theta|y;\tau)$ and $|\cdot|$ takes the determinant of a matrix. The posterior precision summarises how concentrated the posterior is around its mean. This utility may not be suitable if the posterior distribution is multi-modal or has considerable skew.

Shannon information gain (SIG) is based on the concept of Shannon entropy (Shannon, 1948) which quantifies the average information content of a particular quantity of interest. SIG looks at the change in entropy from the prior distribution to the posterior. In terms of a utility function for optimal design, the SIG utility is given by

$$\mathcal{U}_{SIG}(\tau, \theta, y) = \log \pi(\theta|y;\tau) - \log \pi(\theta). \tag{4.2}$$

SIG has the following interpretation: when $\mathcal{U}_{SIG}$ is positive then the true parameter value has higher density under the posterior than it did under the prior; if it is negative then it has lower density.

Through application of Bayes theorem to Equation 4.2, the SIG utility can be expressed as a function of likelihoods

$$\mathcal{U}_{SIG}(\tau, \theta, y) = \log f(y|\theta;\tau) - \log f(y;\tau), \tag{4.3}$$

hence removing the need to calculate the posterior for θ, reducing the computational burden. This now requires $\log f(y;\tau)$ to be computed. Often this is estimated with a Monte Carlo method which can become expensive depending on the dimension of $\mathcal{Y}$ and the precision of the estimate required.

SIG is closely associated with the Kullback-Leibler divergence (KLD) (Kullback and Leibler, 1951). The expected utility under the KLD is

$$\begin{aligned}
\mathcal{J}_{KLD}(\tau) &= \int_{\mathcal{Y}} \int_{\Theta} \log \frac{\pi(\theta|y;\tau)}{\pi(\theta)} \pi(\theta|y;\tau) d\theta \; \pi(y;\tau) dy, && (4.4)\\
&= \int_{\mathcal{Y}} \int_{\Theta} \log \frac{\pi(\theta|y;\tau)}{\pi(\theta)} \pi(\theta, y;\tau) d\theta dy, && (4.5)\\
&= \mathbb{E}_{\theta,y\sim\pi(\theta,y;\tau)}[\log \pi(\theta|y;\tau) - \log \pi(\theta)], && (4.6)\\
&= \mathcal{J}_{SIG}(\tau). && (4.7)
\end{aligned}$$

Hence the expected KLD to the prior from the posterior is equivalent to the expected SIG utility.

The SIG utility function also has a derivation from a decision theoretic approach (Bernardo, 1979). Using the expression for the divergence under the logarithmic score from Table 2.1 in Equation 2.63 the expected SIG utility function is recovered.

4.2 Fisher information gain

This section focuses on the Fisher information gain (FIG) utility function to quantify the information about model parameters which is gained by making observations from a model. The FIG utility is computationally convenient as it is based on the Fisher information matrix which is available in closed-form for some commonly used distributions.

4.2.1 Derivation of the Fisher information gain

The FIG utility function can be derived from a decision theoretic viewpoint using the results from Section 2.5.1 and the divergence of the Hyvärinen score (see Table 2.1). The regularity conditions of the Hyvärinen score are required for both the prior and the posterior given any y, τ (see Section 2.5). Under the Hyvärinen score the divergence between distributions P and Q satisfying the regularity conditions is

$$\mathcal{D}(P, Q) = \mathbb{E}_{\theta \sim p(\theta)}\left[||\nabla \log p(\theta) - \nabla \log q(\theta)||^2\right]. \tag{4.8}$$

The expected utility function can then be expressed as the divergence between the posterior and prior,

$$\begin{aligned}
\mathcal{J}(\tau) &= \mathbb{E}_{y \sim f(y|\theta;\tau)}\left[\mathcal{D}(\pi(\theta|y;\tau), \pi(\theta))\right] && (4.9)\\
&= \int_{\mathcal{Y}} \int_{\Theta} ||\nabla_\theta \log \pi(\theta|y;\tau) - \nabla_\theta \log \pi(\theta)||^2 \pi(\theta|y;\tau)\pi(y;\tau) d\theta dy && (4.10)\\
&= \int_{\Theta} \int_{\mathcal{Y}} ||\nabla_\theta \log f(y|\theta;\tau)||^2 f(y|\theta;\tau)\pi(\theta) dy d\theta && (4.11)\\
&= \mathbb{E}_{\theta \sim \pi(\theta)}\left[\mathbb{E}_{y \sim f(y|\theta;\tau)}\left[||\nabla_\theta \log f(y|\theta;\tau)||^2\right]\right] && (4.12)\\
&= \mathbb{E}_{\theta \sim \pi(\theta)}\left[\mathrm{tr}\mathcal{I}(\theta;\tau)\right] && (4.13)
\end{aligned}$$

where $\mathcal{I}(\theta;\tau)$ is the Fisher information matrix (defined in Section 2.2). The swapping of the order of integration is subject to the conditions of Fubini's theorem (Walker, 2012), namely, $\int_x \int_y f(x,y)dydx = \int_y \int_x f(x,y)dxdy$ if $\int_x \int_y |f(x,y)|dydx$ is finite. This requirement follows from the regularity conditions of the Hyvärinen score.

The FIG utility is defined as the trace of the Fisher information matrix with expected utility defined in Equation 4.9,

$$\mathcal{U}_{FIG}(\tau, \theta, y) = \mathrm{tr}\mathcal{I}(\theta;\tau). \tag{4.14}$$

This can be considered a summary of asymptotic posterior precision (see Section 2.1.3). It is equivalent to the classical T optimality condition (as defined in Section 2.3.1) averaged over the joint distribution of parameters θ and data y. Under the definitions of Chaloner and Verdinelli (1995) the FIG utility can be considered pseudo-Bayesian as it appears to be a functional of the approximate posterior rather than the exact posterior. However, Equation 4.9 shows that FIG can be derived from a decision theoretic viewpoint and thus can be considered fully-Bayesian. Further to this, FIG gives the same expected utility as a functional of the posterior: the Hyvärinen divergence. Walker (2016) came to a similar conclusion that FIG has a Bayesian interpretation.

4.2.2 FIG for data models

This section shows the derivation of the FIG for some data models used in the main examples of this thesis. Several cases of normally distributed variables will be considered as well as those which have a Binomial distribution.

Binomial distribution

Consider a vector of observations $y = (y_1, y_2, \ldots, y_d)$ at times $\tau = (\tau_1, \tau_2, \ldots, \tau_d)$ where each $Y_i|\theta; \tau_i \sim Bin(n, \alpha_i)$ independently with $\alpha_i = \alpha(\tau_i, \theta)$. The parameter in this model is a scalar quantity θ. From Section 2.2.1, the Fisher information is defined as

$$\mathcal{I}(\theta; \tau) = \mathbb{E}_{y \sim f(y|\theta;\tau)}\big[u(\theta, y; \tau)^2\big], \tag{4.15}$$

where

$$u(\theta, y; \tau) = \sum_{i=1}^{d} \frac{d\alpha_i}{d\theta}\left(\frac{y_i - n\alpha_i}{\alpha_i(1-\alpha_i)}\right), \tag{4.16}$$

as derived in Equation 2.39. In this model the Fisher information is a scalar meaning $\mathrm{tr}\mathcal{I} = \mathcal{I}$ and hence

$$\mathcal{U}_{FIG}(\tau, \theta) = \mathcal{I}(\theta; \tau). \tag{4.17}$$

Normal distribution

Consider observations distributed according to a multivariate normal distribution,

$$Y \sim N_d(\mu(\theta, \tau), \Sigma(\theta, \tau)), \tag{4.18}$$

for some mean function $\mu(\cdot)$ and covariance function $\Sigma(\cdot)$. This section derives the FIG utility function for various choices of covariance structure using the equation of the Fisher information matrix (Equation 2.42).

Constant covariance with independent observations. In this simple case, the covariance matrix is proportional to the identity matrix, $\Sigma = \sigma^2 \mathbb{I}_d$. Σ is therefore not dependent on the parameters $\theta = (\theta_1, \theta_2, \ldots, \theta_p)$ and thus $\frac{\partial \Sigma}{\partial \theta_i} = 0$ for all $i = 1, 2 \ldots, p$. The FIG utility is then

$$\mathcal{U}_{FIG}(\tau, \theta) = \mathrm{tr}(\mathcal{I}(\theta; \tau)) = \sum_{i=1}^{p} \frac{\partial \mu^T}{\partial \theta_i} \Sigma^{-1} \frac{\partial \mu}{\partial \theta_i}. \tag{4.19}$$

Due to the covariance matrix being diagonal, the FIG can also be expressed as a double summation,

$$\mathcal{U}_{FIG}(\tau,\theta) = \text{tr}(\mathcal{I}(\theta;\tau)) = \sigma^{-2}\sum_{i=1}^{p}\sum_{k=1}^{d}\left(\frac{\partial\mu_k}{\partial\theta_i}\right)^2. \tag{4.20}$$

Since the scale of the variance, σ^2, is constant the maximum FIG is obtained by maximising $\sum_{i=1}^{p}\sum_{k=1}^{d}\left(\frac{\partial\mu_k}{\partial\theta_i}\right)^2$. Therefore the optimal design does not depend on the magnitude of the variance.

Errors dependent upon mean function with independence across observations. Here the covariance matrix takes the form $\Sigma = \sigma_1^2\mathbb{I}_d + \sigma_2^2\text{diag}(f(\mu))$. The Fisher information matrix for this model is defined in Equation 2.50. Using this the FIG utility is defined as follows,

$$\mathcal{U}_{FIG}(\tau,\theta) = \text{tr}(\mathcal{I}(\theta;\tau)) = \sum_{i=1}^{p}\left\{\frac{\partial\mu^T}{\partial\theta_i}\Sigma^{-1}\frac{\partial\mu}{\partial\theta_i} + \frac{\sigma_2^4}{2}\sum_{k=1}^{d}\Sigma_{kk}^{-2}\left(\frac{\partial f_k}{\partial\mu_k}\frac{d\mu_k}{d\theta_i}\right)^2\right\}, \tag{4.21}$$

where $f_k = f_k(\mu)$. The covariance matrix Σ is diagonal and thus the utility can be expressed as

$$\mathcal{U}_{FIG}(\tau,\theta) = \sum_{i=1}^{p}\sum_{k=1}^{d}\left(\frac{\partial\mu_k}{\partial\theta_i}\right)^2\Sigma_{kk}^{-1} + \frac{\sigma_2^4}{2}\sum_{k=1}^{d}\Sigma_{kk}^{-2}\left(\frac{\partial f_k}{\partial\mu_k}\frac{d\mu_k}{d\theta_i}\right)^2, \tag{4.22}$$

$$= \sum_{i=1}^{p}\sum_{k=1}^{d}\left(\frac{\partial\mu_k}{\partial\theta_i}\right)^2\Sigma_{kk}^{-1}\left\{1 + \frac{\sigma_2^4}{2}\Sigma_{kk}^{-1}\left(\frac{\partial f_k}{\partial\mu_k}\right)^2\right\}, \tag{4.23}$$

where $\Sigma_{kk} = \sigma_1^2 + \sigma_2^2 f_k$.

Correlated errors across observations If the covariance matrix is correlated across designs but does not depend on any other parameters, i.e. $\Sigma = \Sigma(\tau)$, then the Fisher information matrix is defined as in Equation 2.43 since $\frac{\partial\Sigma}{\partial\theta_k} = 0$ for all k. The corresponding utility function is given by Equation 4.19,

$$\mathcal{U}_{FIG}(\tau,\theta) = \text{tr}(\mathcal{I}(\theta;\tau)) = \sum_{i=1}^{d}\frac{\partial\mu^T}{\partial\theta_i}\Sigma^{-1}\frac{\partial\mu}{\partial\theta_i}. \tag{4.24}$$

Note that for this specification of the variance structure errors are correlated across the design so Σ is not diagonal and so computation of $\mathcal{U}_{FIG}(\tau,\theta)$ requires matrix calculations.

4.2.3 Scaled FIG

The FIG utility is not scale invariant meaning that the value it takes depends on the scale of the parameters θ. In certain models a lot of information about one of the parameters can be easily gained whilst the others are negligible in terms of their contribution to the FIG. In these cases the design which maximises the FIG will focus on optimising the contribution from the dominant parameter. In practice this is undesirable as all parameters should be learned equally if there is no prior motivation for weighting their contributions towards the utility. To address this issue, each element of the trace of the information matrix can be scaled by a constant, representing the typical value for that contribution, resulting in the contribution relating to each parameter being roughly equal to unity. Weighting the contributions is equivalent to scaling the parameters in the model and hence the optimal design ideally should be invariant under this operation.

Weighted FIG considers a reparameterisation $\xi(\theta) = (w_1\theta_1, w_2\theta_2, \ldots, w_p\theta_p)$. Using Equation 2.31, the weighted FIG utility is given by

$$\begin{aligned}
\mathcal{U}_{FIG}(\tau, \theta; \tilde{w}) &= \mathrm{tr}\mathcal{I}_{\xi}(\theta; \tau, \tilde{w}), && (4.25)\\
&= \mathrm{diag}(\mathcal{I}(\theta; \tau))^T \tilde{w}^{-1}, && (4.26)\\
&= \sum_{i=1}^{p} \mathcal{I}_{ii}/\tilde{w}_i, && (4.27)\\
&= \sum_{i=1}^{p} \mathbb{E}_{y\sim f(y|\theta;\tau)}\left[\frac{u_i(\theta, y; \tau)^2}{w_i^2}\right], && (4.28)
\end{aligned}$$

where diag maps a matrix to its leading diagonal, u_i is the i^{th} element of the Fisher score (Equation 2.12) and $\tilde{w} = (w_1^2, w_2^2, \ldots, w_p^2)$ is the vector of squared weights, i.e. $\tilde{w}_i = w_i^2$.

When using weighted FIG, the goal is to find suitable weights and the optimal design, i.e. the aim is to find $(\tau^*, \tilde{w}^*)$ such that

$$\tau^* = \max_{\tau\in\mathcal{T}} \mathcal{J}_{FIG}(\tau; \tilde{w}^*), \tag{4.29}$$

where

$$\tilde{w}^* = \max_{\tilde{w}} \mathcal{K}(\tilde{w}; \tau^*), \tag{4.30}$$

with

$$\mathcal{K}(\tilde{w}; \tau) = -\frac{1}{2}|| \ \tilde{w} - \mathbb{E}_{\theta\sim\pi(\theta)}[\mathrm{diag}(\mathcal{I}(\theta; \tau))] \ ||^2. \tag{4.31}$$

The optimal weights are chosen according to $\mathcal{K}(\tilde{w}; \tau)$ as this tries to make each of the contributions to the sum in Equation 4.28 equal.

Algorithm 4.1 outlines a stochastic optimisation routine for optimal design with adaptive weights. This requires estimates of the gradient of $\mathcal{J}_{FIG}(\tau; \tilde{w})$ and $\mathcal{K}(\tilde{w}; \tau)$. Both are found

Algorithm 4.1 Stochastic gradient optimisation of expected Fisher information gain with adaptive weights.

1: Initialise design τ_0, number of iterations n, batch size K (used in gradient estimation). Set initial squared weights $\tilde{w}_0$.
2: **for** $t = 0, 1, 2, \ldots, n-1$ or until a convergence condition is reached **do**
3: Calculate $g_{\tau,t}$, an estimate of $\nabla\mathcal{J}(\tau)$ at $\tau = \tau_t$ using Equation 3.17 and $g_{\tilde{w},t}$ an estimated gradient for $\tilde{w}$.
4: Get $\tau_{t+1}, \tilde{w}_{t+1}$ by calling the update rule of a stochastic gradient optimisation algorithm with current state $\tau_t, \tilde{w}_t$ and gradient estimates $g_{\tau,t}$ and $g_{\tilde{w},t}$. The update rule of the Adam algorithm (Section 3.1.4) is used for both updates.
5: **return** $\tilde{w}_n$

using unbiased Monte Carlo estimates:

$$\widehat{\nabla}\mathcal{J}_{FIG}(\tau;\tilde{w}) = \frac{1}{K}\sum_{k=1}^{K}\nabla[\text{diag}(\mathcal{I}(\theta^{(k)};\tau))^T\tilde{w}^{-1}], \tag{4.32}$$

and

$$\hat{\nabla}\mathcal{K}(\tilde{w};\tau) = \frac{1}{K}\sum_{k=1}^{K}\text{diag}(\mathcal{I}(\theta^{(k)};\tau)) - \tilde{w}, \tag{4.33}$$

where $\theta^{(k)}$ are independent draws from the prior.

Convergence of Algorithm 4.1 is difficult to guarantee (see Appendix B of Harbisher et al., 2019). The optimisation function becoming stuck in a cycle is also a possibility with simultaneous gradient optimisation of the objective functions (Balduzzi et al., 2018). However in practice convergence was achieved (see Section 6.2.1). The practical approach to applying the adaptive algorithm is to conduct a pilot run for a fixed number of iterations before fixing the weights and switching to the standard algorithm (Algorithm 3.4).

4.2.4 Adversarial FIG

Overstall (2020) discussed how the FIG is prone to returning designs which correspond to an ill conditioned posterior. When maximising the expected Fisher information the diagonal elements of the information matrix are not considered and as such the returned design can correspond to a singular information matrix. The example considered in Chapter 6 demonstrates this behaviour, even when using the adaptive weighting procedure described in Section 4.2.3. This motivates the need for a modification to FIG to attempt to offset this behaviour. One approach is to use an adversarial approach (see, for example, Banks et al., 2015). This will aim to find the best design taking into account that an adversary will select a linear transformation of the variables aiming to minimise

the resulting utility. The transformation of variables introduces some dependence from the off-diagonal elements of the information matrix into the utility function. This dependence should address some of the issues of the FIG utility therefore the adversarial variant of FIG should avoid returning designs corresponding to singular information matrices.

The adversarial FIG (AFIG) utility considers a transformation of variables, $\xi(\theta) = A\theta$ where A is an invertible matrix with inverse $B = A^{-1}$ and dimension $p \times p$. From Equation 2.31,

$$\mathcal{I}_\xi(\xi;\tau) = B\mathcal{I}_\theta(\theta;\tau)B^T. \tag{4.34}$$

Hence the AFIG utility is given by

$$\mathcal{U}_{AFIG}(\tau,\theta,B) = \mathrm{tr}(B\mathcal{I}_\theta B^T), \tag{4.35}$$

$$= \mathrm{tr}(B^T B\mathcal{I}_\theta), \tag{4.36}$$

due to the cyclic property of the trace. Note that when $B = \mathrm{diag}(1/w_1, 1/w_2, \ldots, 1/w_p)$ the scaled FIG (as described in Section 4.2.3) is recovered.

The matrix B can be considered the Cholesky factor of $B^T B$ and hence can be a lower triangular matrix without loss of generality. Note that the magnitude of B scales the utility by an arbitrary positive constant. This has no effect on the optimal design and so the condition that $|B| = 1$ is recommended. The enforcement that B is lower triangular with determinant equal to one reduces the number of parameters needed to define B. The ij^{th} element of B is given by

$$B_{ij} = \begin{cases} 0, & i < j \\ b_{ij}, & i > j \\ \exp(b_{ij}), & i = j < p \\ \exp(-\sum_{k=1}^{p-1} b_{kk}), & i = j = p \end{cases} \tag{4.37}$$

meaning B is parametrised by $(b_{11}, b_{21}, b_{22}, b_{31}, \ldots, b_{p\ p-1})$.

For the AFIG utility the objective function now relies on design τ and the matrix B,

$$\mathcal{K}(\tau, B) = \mathbb{E}_{\theta\sim\pi(\theta)}[\mathcal{U}_{AFIG}(\tau,\theta,B)]. \tag{4.38}$$

Due to the adversarial nature of this utility function the aim is to find the optimal design under the worst case scenario for the parametrisation of B, i.e.

$$\tau^* = \max_\tau \mathcal{K}(\tau, B^*), \tag{4.39}$$

$$B^* = \min_B \mathcal{K}(\tau^*, B). \tag{4.40}$$

Algorithm 4.2 Stochastic gradient optimisation of expected adversarial Fisher information gain $\mathcal{K}(\tau, B)$.

1: Initialise design τ_0, number of iterations n, batch size K (used in gradient estimation). Set initial weights matrix B_0.
2: **for** $t = 0, 1, 2, \ldots, n-1$ or until a convergence condition is reached **do**
3: Calculate $g_{\tau,t}$, an estimate of $\nabla_\tau \mathcal{K}(\tau, B)$, and $g_{B,t}$, an estimate of $-\nabla_B \mathcal{K}(\tau, B)$, at $\tau = \tau_t$, $B = B_t$ using a Monte Carlo estimate.
4: Get τ_{t+1}, B_{t+1} by calling the update rule of a stochastic gradient optimisation algorithm with current state τ_t, B_t and gradient estimates $g_{\tau,t}$ and $g_{B,t}$. The update rule of the Adam algorithm (Section 3.1.4) is used for both updates.
5: **return** τ_n

The procedure to find τ^* and B^* via simultaneous gradient descent (Nagarajan and Kolter, 2017) is described in Algorithm 4.2.

The convergence of algorithm Algorithm 4.2 is not guaranteed and cyclic behaviour could be observed. In practice, for the example considered in Section 6.4, convergence was reached without any adaptations to the algorithm. If this was not the case then techniques from the machine learning community could be employed, i.e. a two time-scale update rule (Heusel et al., 2017). This is where the learning rates of the respective optimisation steps are different. Asymptotically, one of the learning rates should approach zero faster than the other aiding convergence.

4.2.5 Limitations

The Fisher information is computationally convenient when available in closed form. In the case where nuisance parameters or latent variables are present in the model then these need to be integrated out of the likelihood. This may mean that the score function $\nabla_\theta \log f(y|\theta; \tau)$ is not available in a closed form leading to the Fisher information being intractable. It is possible in these circumstances to estimate the Fisher information using samples from the posterior distribution of the nuisance parameters however this increases the computational burden of obtaining the utility therefore limiting the computational benefits of the FIG utility.

4.3 Utility based on classical D-optimality criterion

This section considers a utility function closely related to classical D-optimality and the SIG utility functions. Section 4.3.1 defines and describes some properties of the D-optimality utility function. Examples of finding the design under this utility are shown in Chapter 5 and Chapter 6.

4.3.1 D-optimality utility function

A utility function that is often used in optimal design is the log determinant of the Fisher information matrix,

$$\mathcal{U}_D(\tau, \theta, y) = \mathcal{U}_D(\tau, \theta) = \log|\mathcal{I}(\theta; \tau)|. \tag{4.41}$$

This corresponds to classical D-optimality. Note that D-optimality can be generalised to the Bayesian setting in several ways, usually as a function of the determinant of $\mathcal{I}$, i.e. $\mathcal{U}_D(\tau, \theta, y) = g(|\mathcal{I}(\theta; \tau)|)$ for some function g.

This utility does not depend on y as the information matrix takes the expectation over the data. In Bayesian optimal design this is averaged over the prior predictive distribution to obtain the expected utility. This utility is considered pseudo-Bayesian under the definition of Chaloner and Verdinelli (1995). Using asymptotic results, the optimal design under this utility can be thought of as maximising the approximate generalised precision.

Unlike the FIG utility, the $\log|\mathcal{I}(\theta; \tau)|$ utility does not have a derivation from a decision theoretic viewpoint. It can however be considered as an approximation to the SIG utility (see Section 2.1.3). Lohr (1995) using a result from Clarke and Barron (1990) showed that maximising $\log|\mathcal{I}(\theta; \tau)|$ is equivalent to maximising SIG asymptotically.

A desirable property of this utility function is that $\log|\mathcal{I}(\theta; \tau)|$ is invariant to linear reparametrisations. Consider a reparameterisation $\xi(\theta) = A\theta + b$ where A is a constant square matrix and b is a constant vector. Then using Equation 2.31 it is possible to show

$$\log|\mathcal{I}_\theta(\theta; \tau)| = \log|A\mathcal{I}_\xi(\xi; \tau)A^T|, \tag{4.42}$$

$$= 2\log|A| + \log|\mathcal{I}_\xi(\xi; \tau)|, \tag{4.43}$$

and so the transformation of variables only changes the utility by a constant with respect to the design τ. The optimal design τ^* is the same under both parameterisations. Due to this, Firth and Hinde (1997) defined $\mathcal{U}_D$ as *parameter neutral* and claimed it is the most natural formulation of D-optimality for Bayesian optimal design.

A utility function based on the information matrix may be liable to rewarding a large value for one diagonal element at the expense of not learning about the other model parameters. This was discussed in Section 4.2.3. Ideally, the utility should give diminishing returns so that the other model parameters can be learned. One way to do this is to incorporate risk aversion, for example, including a transformation according to a concave function (Parmigiani and Inoue, 2009). The $\log|\mathcal{I}(\theta; \tau)|$ utility can be viewed as a more risk averse than the FIG utility when expressed in terms of the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_p$

of the information matrix $\mathcal{I}(\theta;\tau)$. Equation 4.44 demonstrates that the $\log|\mathcal{I}|$ and FIG can be expressed as $\sum_{i=1}^{p} g(\lambda_i)$ where g is a function of the eigenvalues λ.

$$\log|\mathcal{I}(\theta;\tau)| = \sum_{i=1}^{p} \log \lambda_i \qquad \text{tr}\mathcal{I}(\theta;\tau) = \sum_{i=1}^{p} \lambda_i \tag{4.44}$$

The $\log|\mathcal{I}(\theta;\tau)|$ utility uses a concave function $g(x) = \log(x)$ and so introduces some risk aversion whereas FIG sets g as the identity function, $g(x) = x$.

This utility function is also well suited to optimisation via SGD type methods as the gradients are easy to compute, especially when using automatic differentiation software. Similarly to the FIG utility, the $\log|\mathcal{I}|$ utility benefits from $\mathcal{I}$ often being available in closed-form. However $\mathcal{U}_D$ incurs a higher computational cost as the determinant of a matrix is a more costly operation than finding the trace of a matrix.

Chapter 5

Examples

This chapter considers using stochastic gradient descent (SGD) to find the optimal design in two different examples using the utility functions as described in Chapter 4. The first example in Section 5.1 uses a simple death model to demonstrate that the SGD approach does indeed return the correct optimal design. Next, a geostatistical model is used in Section 5.2. This example requires a large number of observations that are correlated hence should demonstrate an algorithms ability to scale to high dimensional problems.

5.1 Simple death model

The simple death model measures the size of a population over time where only deaths can occur. The model assumes that there is no immigration or births into the system, resulting in a population size that is strictly decreasing from its initial size. Other model assumptions are that deaths happen independently and the rate of death remains constant. This means that individuals within the population are not affected by their surroundings or neighbours. In this case the design τ are the times at which observations of the population size should be made in order to learn about the death rate θ.

The simple death model is widely used to demonstrate optimal design methodology as certain utility functions have an analytically tractable solution to the design problem. Many authors (Cook et al., 2008; Drovandi and Pettitt, 2013; Gillespie and Boys, 2019) have used this model as an example to demonstrate that their method identifies the correct design.

Renshaw (1993) considers that deaths in the population occur with constant rate $\theta > 0$ according to a stochastic process. First the probabilities of an individual being alive or

dead at time τ are defined as

$$\alpha(\tau) = Pr(\text{Individual alive at time } \tau), \tag{5.1}$$
$$\beta(\tau) = Pr(\text{Individual dead at time } \tau) = 1 - \alpha(\tau). \tag{5.2}$$

For a small interval of time, $[\tau, \tau + h)$,

$$\alpha(\tau + h) = Pr(\text{Alive at time } \tau \text{ and does not die in } [\tau, \tau + h)) \tag{5.3}$$
$$= \alpha(\tau) \times Pr(\text{Does not die in } [\tau, \tau + h)) \tag{5.4}$$
$$\simeq \alpha(\tau)(1 - \theta h), \tag{5.5}$$

which can be expressed as follows,

$$\frac{\alpha(\tau + h) - \alpha(\tau)}{h} = -\theta\alpha(\tau). \tag{5.6}$$

Taking the limit as $h \to 0$ yields

$$\frac{d\alpha(\tau)}{d\tau} = -\theta\alpha(\tau), \tag{5.7}$$

which is solved to obtain

$$\alpha(\tau) = e^{-\theta\tau}, \tag{5.8}$$

and thus

$$\beta(\tau) = 1 - e^{-\theta\tau}. \tag{5.9}$$

The population at time τ is then a sum of Bernoulli random trials each with probability of surviving $\alpha(\tau)$ and thus the population at can then be modelled as a Binomial random variable. In this example the initial population size is considered fixed at n giving

$$P(\tau) \sim Bin(n,\ \alpha(\tau)) \ \equiv \ Bin(n,\ e^{-\theta\tau}), \tag{5.10}$$

which has mean $ne^{-\theta\tau}$ and variance $ne^{-\theta\tau}(1 - e^{-\theta\tau})$. Simulations from this model can be made using the following relation

$$P(\tau + \delta)|P(\tau) \sim Bin(P(\tau), e^{-\theta\delta}). \tag{5.11}$$

where $\delta > 0$ is an increment in time.

Since the death rate is constrained to be positive, a suitable choice of prior for θ is a log-Normal. Following the example used by Cook et al. (2008); Drovandi and Pettitt (2013) and Gillespie and Boys (2019), the prior distribution for the death rate is defined as

$$\log(\theta) \sim N(-0.005, 0.01),$$

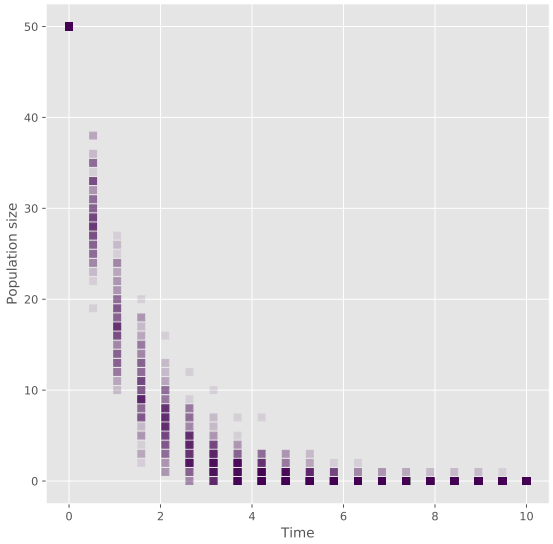

Figure 5.1: *Simulated prior predictive population sizes for the simple death model with initial population size of* 50.

with the initial population size fixed at $n = 50$.

The prior predictive populations displayed in Figure 5.1 show the exponential decay that characterises the simple death model. The rate of this decay is controlled solely by the parameter θ. For this choice of prior distribution the population appears to usually be extinct by time $\tau = 7$. The figure also shows that the variability of the population size is larger at $\tau \in [1.0, 2.5]$, suggesting that there may be more information about the true value of θ in this range of times rather than at times where the population size is less variable.

5.1.1 FIG utility

In this example a single observation y is made at the scalar design τ. Using Equation 2.40 and $Var(Y) = \mathbb{E}[(Y - n\alpha)^2] = n\alpha(1-\alpha)$ the following is obtained.

$$\mathcal{I}(\theta; \tau) = \left(\frac{d\alpha}{d\theta}\right)^2 \frac{n}{\alpha(1-\alpha)}. \tag{5.12}$$

Hence, the FIG for this model (using Equation 4.17) is given by

$$\mathcal{U}_{FIG}(\tau, \theta) = \mathcal{I}(\theta; \tau) = \frac{\tau^2 n e^{-\theta\tau}}{1 - e^{-\theta\tau}}, \tag{5.13}$$

since $\frac{d\alpha}{d\theta} = -\tau e^{-\theta\tau}$.

The utility surface for this model when a one observation design is required is shown in Figure 5.2. The expected FIG,

$$\mathcal{J}_{FIG}(\tau) = \mathbb{E}_{\theta \sim \pi(\theta)}\big[\mathcal{U}_{FIG}(\tau, \theta)\big], \tag{5.14}$$

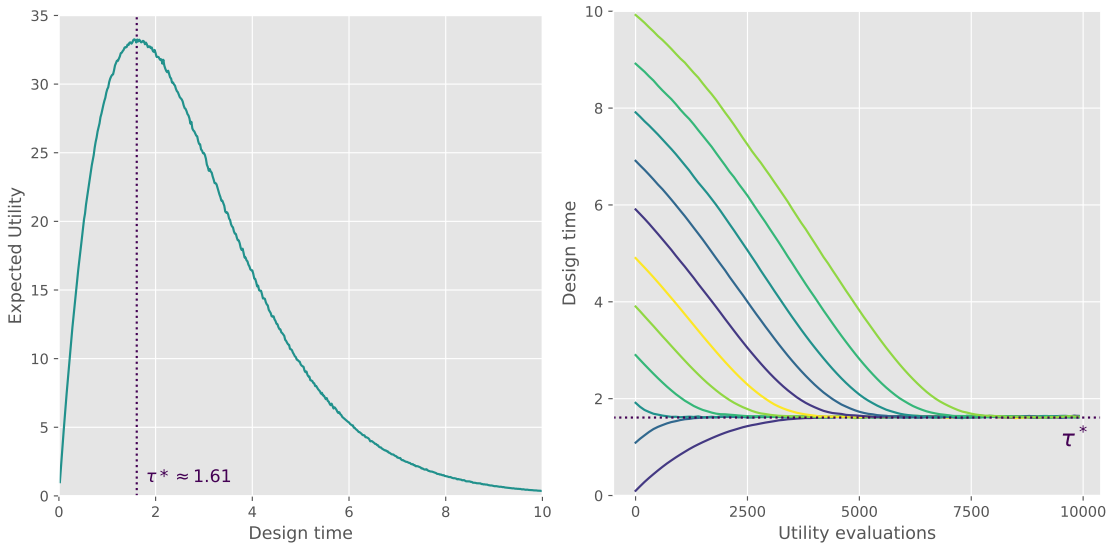

Figure 5.2: *The expected FIG utility surface of the simple death model (left) and associated trace plots of observation time τ against computation cost, measured in utility gradient evaluations, for 11 independent runs of the SGD algorithm (right). The optimal design τ^* is shown by the dotted line.*

can feasibly be computed via numerical methods. Here the expected utility is maximised at $\tau^* \approx 1.61$.

For this example, the SGD algorithm (Algorithm 3.4) is implemented using default tuning of the Adam update rule. Multiple independent runs are considered from 11 different initial states equally spaced over $\mathcal{T} = [0, 10]$. Each run has a fixed computational budget of $10,000$ utility evaluations.

Figure 5.2 (right) shows a trace plot of the SGD design over the computational cost, measured in terms of utility gradient evaluations. It clearly converges to a design around the optimal design value of 1.61 demonstrating that the SGD algorithm targets the correct design to optimise the expected FIG utility. This is true for each of the 11 independent runs of the SGD algorithm, each starting at an initial states evenly spaced over the interval $(0, 10)$. Note that the further the initial state is from the optimal design the more iterations it requires to converge. Overall, this demonstrates that in this example of a unimodal utility function the SGD algorithm converges to the correct design from a variety of initial conditions.

5.1.2 AFIG utility

Consider the AFIG utility as described in Section 4.2.4. Here the model has a single parameter θ. For the reparameterisation used in the AFIG utility $\xi(\theta) = a\theta$ where $a > 0$ is a scalar. This scales the utility by an arbitrary magnitude. Under the suggested constraints

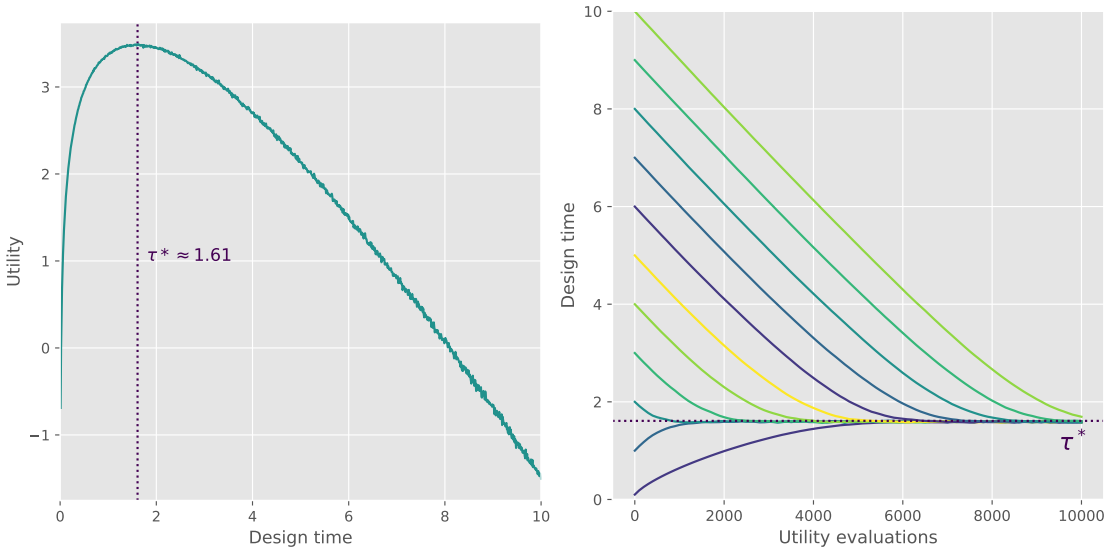

Figure 5.3: *The expected* $\log|\mathcal{I}|$ *utility surface of the simple death model (left) and associated trace plots of observation time* τ *against computation cost ,measured in utility gradient evaluations, for* 11 *independent runs of the SGD algorithm (right). The optimal design* τ^* *is shown by the dotted line.*

on A the magnitude of the transformation should be equal to 1, hence here $a = 1$. This means the transformation is the identity function therefore the AFIG utility is equivalent to the FIG utility function in this example.

5.1.3 D-OPTIMALITY UTILITY

This section considers the $\mathcal{U}_D$ utility function rather than $\mathcal{U}_{FIG}$ as used in Section 5.1.1. Note that these utilities differ only by the logarithmic term as the information matrix (given by Equation 2.40) becomes a scalar in this example thus the trace and determinant give equivalent values. The utility in this example is

$$\mathcal{U}_D(\tau, \theta) = \log\left(\frac{\tau^2 n e^{-\theta\tau}}{1 - e^{-\theta\tau}}\right). \tag{5.15}$$

The expected utility curve under the prior for θ is shown in Figure 5.3. This was found using numerical methods. Similarly to Section 5.1 the expected utility is maximised at $\tau^* = 1.61$.

Figure 5.3 displays the trace over computational cost of 11 independent runs of the SGD algorithm from initial states $0, 1, 2, \ldots, 10$. All of the runs converge to around 1.61 and thus demonstrate that the SGD algorithm is targeting designs which maximise $\mathcal{J}_D(\tau) = \mathbb{E}_{\theta\sim\pi(\theta)}[\mathcal{U}_D(\tau, \theta)]$ with those starting at initial states further from τ^* taking longer to converge. Compared to the traces when using the utility $\mathcal{U}_{FIG}$, convergence is slower for the $\mathcal{U}_D$ utility.

5.2 Geospatial model

Geospatial models are used extensively to model processes over space. There are many applications such as in environmental sciences where Sampson and Guttorp (1992) used them to model solar radiation in south western British Columbia; crime mapping (Leitner, 2013); social science, modelling interactions between agents (Ward and Gleditsch, 2018) and health research in spatial epidemiology (Bergquist and Rinaldi, 2010).

Gaussian processes (GPs) are often used for geospatial models (Diggle, 2007). These are characterised by a mean function and a Gaussian error structure that is correlated over space,

$$Y \sim N_d(\mu(\theta, \tau), \Sigma(\theta, \tau)). \tag{5.16}$$

The covariance matrix quantifies how inputs are related. It assumes that locations that are close to each other should give similar outputs. Commonly used covariance functions belong to the squared exponential and Matérn classes of functions. These are defined and discussed in Rasmussen and Williams (2006). The covariance matrix can also include some additive noise that is independent of location known as a nugget effect and thus Σ is

$$\Sigma(\tau) = \sigma_1^2 \mathbb{I}_d + R(\theta, \tau), \tag{5.17}$$

where σ_1^2 controls the magnitude of the observational variance component, $\mathbb{I}_d$ is the d dimensional identity matrix corresponding to the number of observations in τ and $R(\theta, \tau)$ is some variance matrix.

The geospatial model considered in this section has a simple linear trend in two dimensions. The design τ is a $d \times 2$ matrix where each row $\tau_i \in [-0.5, 0.5]^2$ is a location in a unit square centred at the origin. Observations y are assumed to be correlated according to a squared exponential covariance function with a nugget effect, giving:

$$Y \sim N(\mu(\theta, \tau), \Sigma(\tau)), \tag{5.18}$$

$$\mu_i = \theta_1 \tau_{i,1} + \theta_2 \tau_{i,2}, \tag{5.19}$$

$$\Sigma(\tau) = \sigma_1^2 \mathbb{I}_2 + \sigma_2^2 \rho(\tau), \tag{5.20}$$

$$\rho_{i,j} = \exp\left[-\sum_{k=1}^{2} (\tau_{i,k} - \tau_{j,k})^2 / \ell^2 \right], \tag{5.21}$$

where $\theta = (\theta_1, \theta_2)$ is the unknown parameter vector of interest that controls the trend, σ_1^2 and σ_2^2 are the observational variance components, ℓ is the covariance length scale and $\mathbb{I}_2$ is the 2-dimensional identity matrix. The form of $\Sigma(\tau)$ gives $\text{Var}(Y_i) = \text{Var}(Y_j)$ for all i, j i.e. all diagonal entries of Σ are the same. The parameters which control the behaviour of the covariance (σ_1, σ_2 and ℓ) are nuisance parameters. For simplicity, these parameters

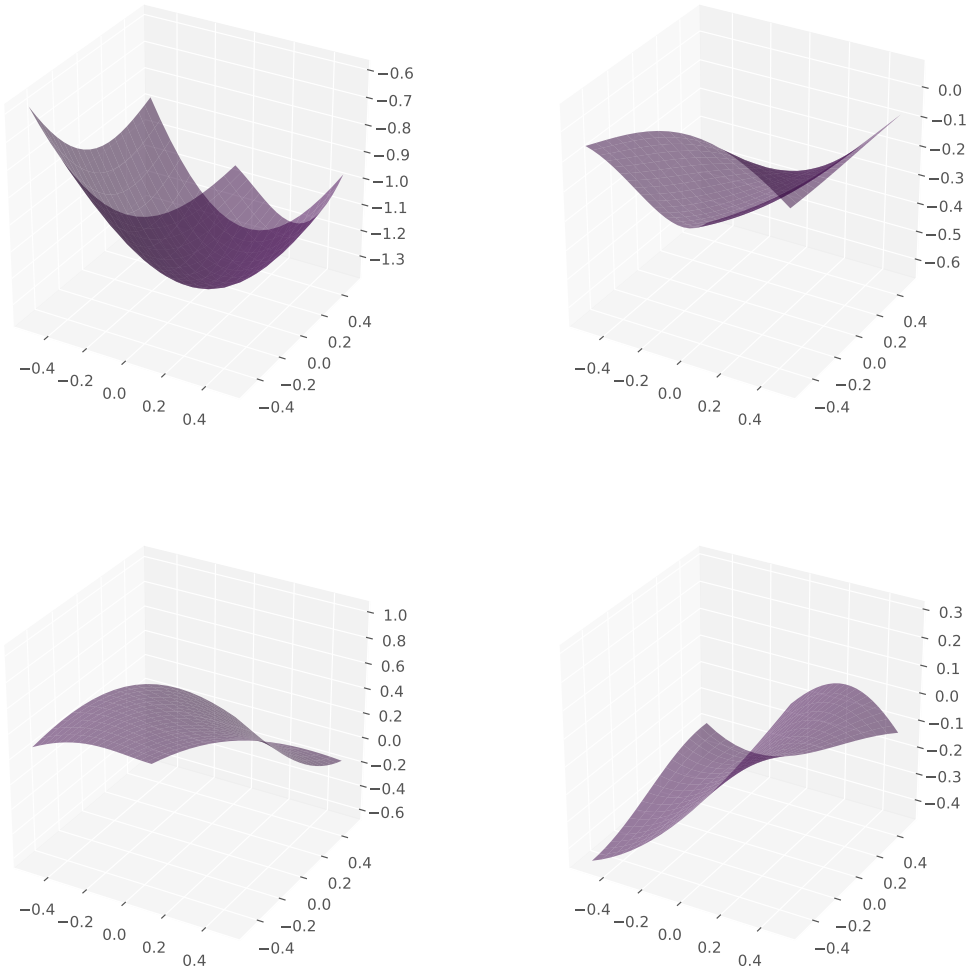

Figure 5.4: *Example fields drawn from the geospatial model as defined in Equation 5.18 for* $\theta_1 = 0.1$, $\theta_2 = -0.3$, $\sigma_1 = 0$, $\sigma_2 = \sqrt{0.5}$ *and* $\ell = 1$. *The* z *axis shows the simulated function value over the unit square (*x *and* y *axis)*

.

will be assumed to be fixed and known. Figure 5.4 shows example outputs from this model for some fixed parameter values.

5.2.1 FIG utility

The FIG for the geospatial model is given by Equation 4.24. The derivative of the mean function with respect to each parameter is given by $\frac{\partial \mu_j}{\partial \theta_i} = \tau_{ij}$, yielding

$$\mathcal{U}_{FIG}(\tau, \theta) = \sum_{j=1}^{2} \tau_{\cdot j}^T \Sigma^{-1} \tau_{\cdot j}, \tag{5.22}$$

where $\tau_{\cdot j}$ is a column of τ.

Note here the utility does not depend on the model parameters θ. Furthermore, if the length scale ℓ and both σ_1^2 and σ_2^2 are fixed the utility function is deterministic. This reduces the computational time required to find the optimal design as the integral over the parameter space is no longer required.

When setting the values of σ_1^2 and σ_2^2 to be constant, the covariance matrix in Equation 5.20 can be written as

$$\Sigma = k[\gamma\mathbb{I} + (1-\gamma)R] \tag{5.23}$$

where $k = \sigma_1^2 + \sigma_2^2$, $\gamma = \sigma_1^2/(\sigma_1^2 + \sigma_2^2)$ and R is the matrix of correlations between designs with entries $R_{ij} = \rho(\tau_i., \tau_j.)$. In this parametrisation of the covariance matrix, γ controls the weighting between the nugget term and the smooth spatial variance. Note that the constant k scales the utility function by an arbitrary amount therefore does not change the optimal design and thus can be neglected when searching for the optimal design.

Search for optimal design using SGD and ACE

This example will search for the optimal 100 observation design. To visualise the effect changing γ and ℓ have on the estimated optimal design returned by SGD see Table 5.1. Note that the designed returned under the following scenarios appear very similar: (a) large ℓ and $\gamma \to 1$ leading to $\Sigma \to k\gamma\mathbb{I}$; (b) $\ell \to 0$ and $\gamma \to 0$ leading to $\Sigma \to k\mathbb{I}$; (c) $\ell \to 0$ and $\gamma \to 1$ leading to $\Sigma \to k\gamma\mathbb{I}$. In all of these scenarios the covariance matrix Σ tends towards a scalar multiple of the identity matrix, $\Sigma \to k'_{\ell,\gamma}\mathbb{I}$, for some constant $k'_{\ell,\gamma}$. Since the optimal design is invariant to the scaling of the covariance matrix (see Section 4.2.2) scenarios (a)-(c) should lead to the same optimal design.

In this example the length scale ℓ^2 is set to a constant value of 0.001 and the weighting between $\mathbb{I}$ and R is set to be $\gamma = 0.1$. This corresponds to a situation where the variation is heavily weighted towards the correlation. Also note that for this choice of length scale there appears to be no replication of observations in the returned designs.

The results from the SGD algorithm (see Section 3.4) will be compared to those returned from the ACE algorithm (Algorithm 2.4), both starting from the same initial states sampled uniformly from the unit square centred at the origin. Note that phase 2 of the ACE algorithm is omitted in this example as no replication of locations were observed. This example discourages replication of design points at a particular location through the local variation component suppressing the information gained by making multiple observations close together.

Since the utility function is deterministic the number of iterations needed for convergence of the SGD algorithm could be reduced compared to when a optimising a stochastic function.

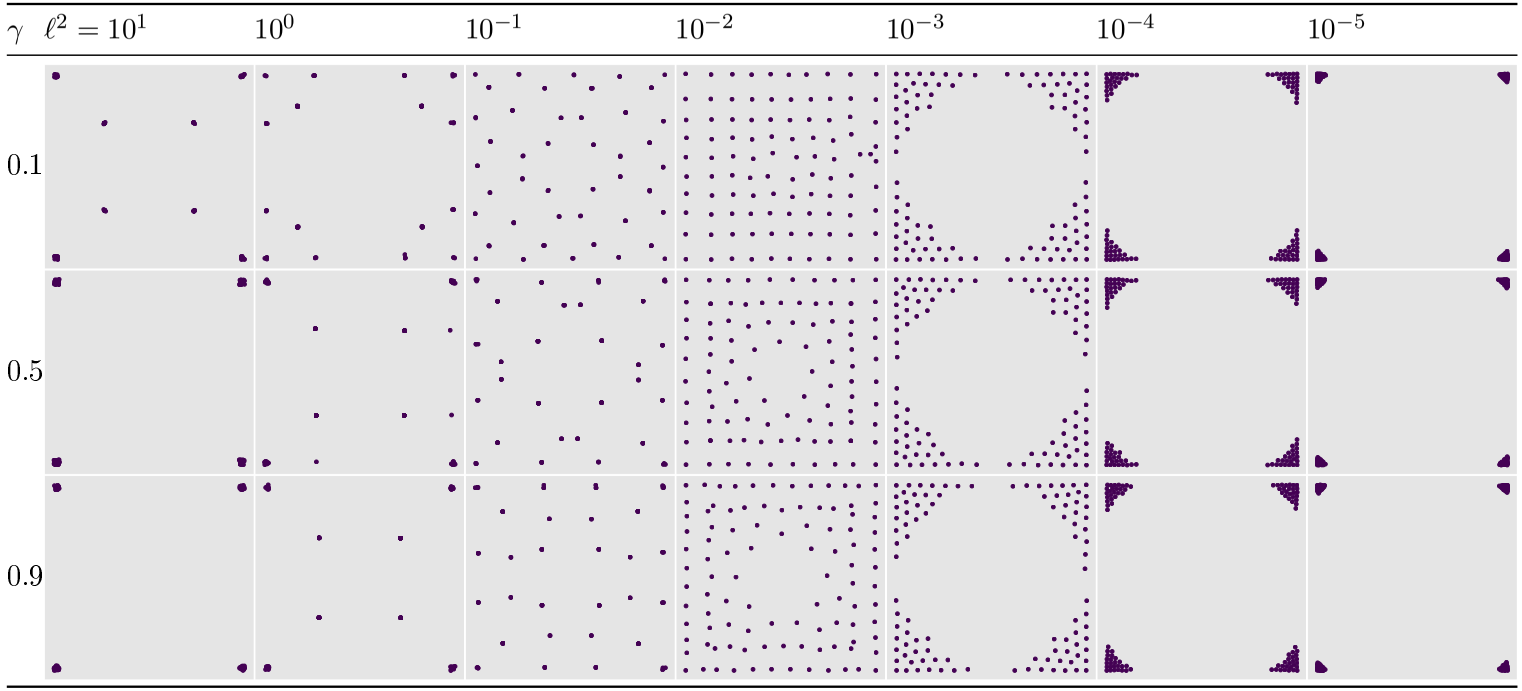

Table 5.1: *The optimal designs output by the SGD algorithm for the geospatial example under the FIG utility shown for various choices of γ and ℓ. The design space is a unit square centred at the origin.*

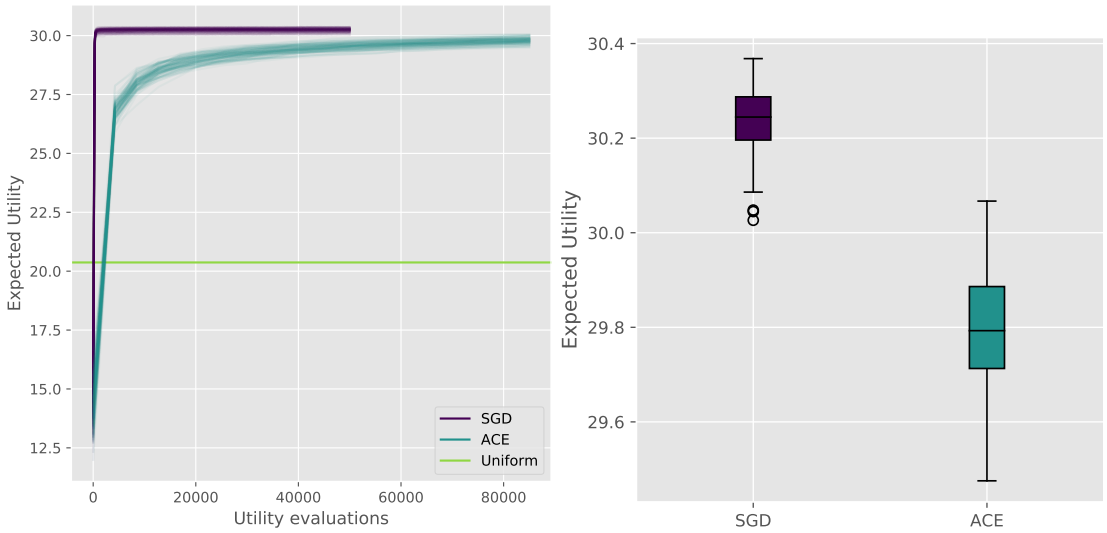

Figure 5.5: *The expected FIG utility for the designs returned by the SGD and ACE algorithms (right) and trace plots of the utility over the computation of the algorithms (left) for the geospatial example. For this example the expected utility is deterministic and hence no estimation is required. The horizontal line indicates the utility for a uniformly spaced grid over the design space. The results shown are from* 100 *independent runs starting from different initial conditions consistent between methods.*

Due to this the SGD algorithm was run for $50,000$ iterations with computational cost of $50,000$ utility evaluations. The ACE algorithm was used with the default settings and hence required its default computational cost ($\approx 85,000$ utility evaluation).

Figure 5.5 shows that for this example the ACE algorithm is outperformed by the SGD algorithm both in terms of utility returned and also the computational overhead required. The ACE algorithm may struggle to converge to the optimal design in this example due to the high correlation between the observations, making it difficult to identify the global maximum utility by updating one τ_{ij} value at a time. This has a lesser effect on SGD which updates all design points simultaneously. SGD appears to have converged after the budgeted number of utility evaluations whereas ACE may need some further iterations before convergence is reached. Note that for SGD a utility evaluation is actually an evaluation of a utility gradient. The SGD algorithm converges after very few iterations, indicating that the computational cost could be lowered significantly if a condition to terminate at convergence was implemented.

This example has shown that in some scenarios, such as when the design points are highly correlated, ACE takes longer to identify the best design. As well as returning designs with better utilities, SGD converges within very few iterations representing an improvement in computational cost required compared to the ACE algorithm.

5.2.2 AFIG utility

In this example the AFIG utility is not required as the parameters are learned equally due to the symmetry of the model. When the AFIG utility is implemented for this model the weighting matrix quickly converges to values which are close to the identity matrix,

$$B_{conv} = \begin{pmatrix} 0.999 & 0.000 \\ -0.016 & 1.001 \end{pmatrix}, \tag{5.24}$$

where B_{conv} is the value of B once the converge has been achieved in Algorithm 4.2. Hence $B^T B$ is also very close to the identity matrix. In this case the AFIG utility is almost the same as the FIG utility function, confirming that the parameters are learned equally in this model.

5.2.3 D-optimality utility

Here the optimal design under the $\mathcal{U}_D$ utility is sought, again drawing comparisons between the SGD and ACE algorithms. Only phase 1 of the ACE algorithm is used throughout this example. There were no replicated locations observed in the design returned and so phase 2 was omitted in this example. As in Section 5.2.1, the variation will be heavily influenced by the correlation component with the weighting between the additive error and the correlation, γ, set to be 0.1 and length scale ℓ^2 is a constant value of 0.001. When constant, the parameter k only scales the utility and thus the optimal design is unchanged. Here k is fixed at a value of 1.

As in the example of Section 5.2.1 the utility function is deterministic and so the number of utility evaluations required by the algorithms should be reduced compared to when a stochastic utility is used. For this example SGD was run for $50,000$ iterations which corresponds to a computational cost of $50,000$ utility evaluations. ACE was run under its default setting which incurred a computational cost of $\approx 85,000$ utility evaluations.

Figure 5.6 shows the trace plots and boxplots of the utility at the returned designs from the SGD and ACE algorithms. The results are very similar to those displayed in Figure 5.5. SGD converges very quickly to designs which yield high utility values. ACE has slower convergence and returns designs which have a smaller utility than those returned by SGD. The trace plot for ACE appears to have a small positive gradient after $80,000$ utility evaluations indicating that the algorithm may not have converged within the default computational budget. Similarly to when $\mathcal{U}_{FIG}$ was used in Section 5.2.1, the ACE algorithm suffers due to the high correlation between designs and so the marginal updates are not as effective as the updates used by SGD where all designs are updated simultaneously. This means that ACE takes longer to converge which requires more utility evaluations.

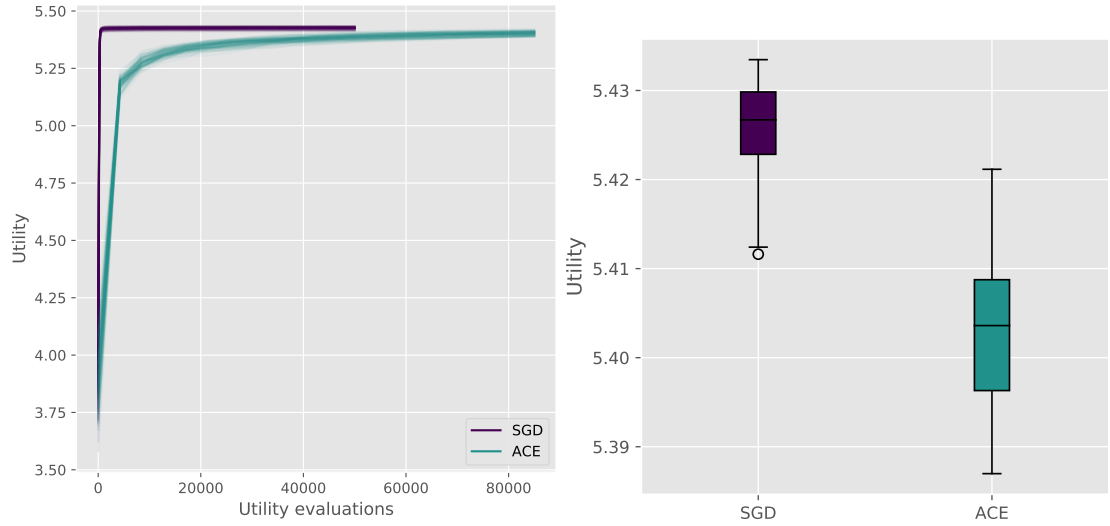

Figure 5.6: *The expected D-optimality utility for the designs returned by the SGD and ACE algorithms (left) and trace plots of the utility over the computation of the algorithms (right) when using the* $\log|\mathcal{I}|$ *utility function for the geospatial example.*

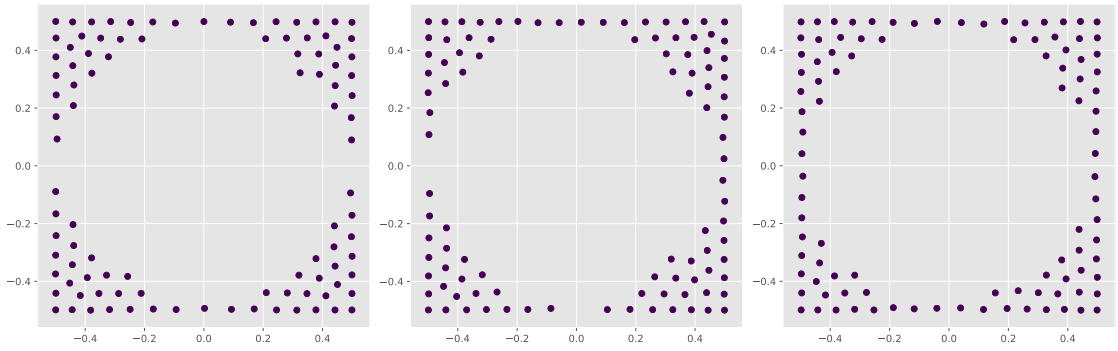

Figure 5.7: *An example design returned from the geospatial model using the FIG (left), AFIG (middle), and* $\log|\mathcal{I}|$ *(right) utility functions.*

Comparison of returned designs

The designs returned under all utility functions for this model are very similar. All favour placing points in the extremes of the design space. The spacing between locations is approximately the same with no replication of points in any of the returned designs under the different utility functions.

5.3 Summary

Section 5.1 demonstrated that SGD targets the correct design for a simple example. This was shown for both the FIG and $\log|\mathcal{I}|$ utility functions. The speed of convergence of the SGD algorithm is dependent upon the initial state with those further from the optimal design taking longer to converge.

Section 5.2 considered an example with many correlated observations. This demonstrated SGD can scale well to find designs with lots of observations. In contrast to ACE, SGD was shown to be well suited to examples where designs are correlated as all observations are updated simultaneously. SGD was efficient in terms of utility evaluations as it converged after very few iterations and also to better designs than those returned by ACE.

Chapter 6

Simulation study

This chapter considers a pharmacokinetic model used to describe the dynamics of the concentration of a drug over time. It is popularly used to compare methods of finding the optimal design. Simulation studies will be conducted to draw comparisons between the SGD, ACE and Müller algorithms. The designs returned under different utility functions are also investigated.

6.1 Compartmental pharmacokinetic model

Pharmacokinetics (PK) is defined as the study of the time course of drug absorption, distribution, metabolism, and excretion (Tozer and Rowland, 2006). The effect of a drug is often related to its concentration. Often, the concentration level has to be observed indirectly via samples of fluid (such as blood, plasma or urine) from the subject.

Models of observations will have some form of error structure to account for the variation in measurements that arise through error inherent in the recording equipment and random variation within the subjects. As continuous observation is infeasible, measurements must be taken at prescribed discrete time points. Due to the sometimes invasive nature of gathering measurements a constraint often enforced is that successive observations have to be a certain time apart. The number of observations are constrained by budget and resources, as well as patient comfort and well being.

Compartmental models are a class of model that are widely used in PK studies (Bonate, 2011). They view the drug as passing through several stages or "compartments" with parameters controlling the rates of movement between compartments. They are popular in optimal design literature, being used to demonstrate varying methodology (Atkinson et al., 1993; Stroud et al., 2001; Gotwalt et al., 2009; Ryan et al., 2014; Overstall and

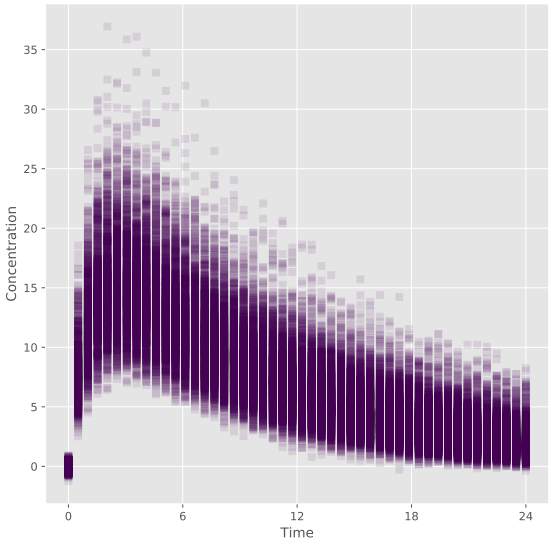

Figure 6.1: *Realisations y from the compartmental PK model using the prior as defined in Equation 6.6 and with σ_1^2 and σ_2^2 fixed at 0.1 and 0.01 respectively.*

Woods, 2017). Here a one compartment model with first order absorption as used by Ryan et al. (2014) and Overstall and Woods (2017) is described.

The model for the mean level of drug can be expressed as an ODE system,

$$\frac{d\mu_1(t)}{dt} = -\theta_1\mu_1(t), \tag{6.1}$$

$$\frac{d\mu_2(t)}{dt} = \frac{\theta_2}{\theta_3}\mu_1(t) - \theta_2\mu_2(t), \tag{6.2}$$

with boundary conditions $\mu_1(0) = D$ and $\mu_2(0) = 0$, where $t \in [0, 24]$ is the time in hours, $\theta = (\theta_1, \theta_2, \theta_3)$ is the unknown parameter vector of interest, μ_1 and μ_2 are the quantity of drug outside (i.e. in a drip) and inside the body (i.e. in the blood plasma) respectively, and $D = 400$ is the known initial dose. This system of equations can be solved to give

$$\mu_1(t) = D\exp(-\theta_1 t), \tag{6.3}$$

$$\mu_2(t) = \frac{D\theta_2}{\theta_3(\theta_2 - \theta_1)}(\exp(-\theta_1 t) - \exp(-\theta_2 t)). \tag{6.4}$$

For this example, interest lies in observing the amount of drug inside the body (Equation 6.4). Observations of the drug concentration y at times $\tau = (\tau_1, \tau_2, \ldots, \tau_d)$ are assumed to be distributed

$$y \sim N_d\left(\mu_2(\tau), \sigma_1^2 + \sigma_2^2 \operatorname{diag}\left(\mu_2(\tau)^2\right)\right), \tag{6.5}$$

where $\mu_2(\tau) = (\mu_2(\tau_1), \mu_2(\tau_2), \ldots, \mu_2(\tau_d))$ is the mean vector, σ_1 and σ_2 are the additive and multiplicative standard deviations respectively and diag$(\cdot)$ maps to a diagonal matrix. Often a condition that successive observations have to be made at least 0.25 hours apart is imposed, i.e. $|\tau_i - \tau_j| > 0.25 \; \forall \; i \neq j$

The errors in this model are independent of patient or any other measurements. The multiplicative term means that there is a relation between the concentration of drug and the error variance with larger variance when the drug concentration is high. Additive error is also present in this model, and is independent of the concentration.

In the Bayesian setting prior information and beliefs about the model parameters are incorporated using prior distributions. Ryan et al. (2014) defined the prior as

$$\log(\theta_j) \sim N(m_j, 0.05), \tag{6.6}$$

independently for $j = 1, 2, 3$ where $m = (\log(0.1), \log(1), \log(20))$. Prior predictive realisations from the model using this prior information are displayed in Figure 6.1. This shows that the concentration y varies over time, increasing from the start of the experiment until times around 3 hours, dependent upon the parameters. The drug is then metabolised causing the concentration to decrease until the end of the observation window of 24 hours.

6.2 FIG utility

This section investigates the performance of SGD under the FIG utility. Comparisons to existing methods of finding the optimal 15 observation design will be made, specifically comparing SGD to the Müller (Section 2.6.1) and ACE (Section 2.6.2) algorithms. For the purposes of the simulation study, simplifications to the PK model (defined in Section 6.1) are used to reduce the computation. Firstly, the multiplicative error term σ_2 is set to be zero and the additive error term σ_1 is assumed to be known and set to a value of 0.1. Further to this, the condition that observations had to be at least 15 minutes apart is also neglected from this simulation study for fairness, as implementation of this restriction would vary between methods. Note that the purpose of this study is to compare methods of finding the optimal design. Simplifying the model and removing its constraints allows the comparison to be made more easily. Due to this the designs which are returned may be unrealistic and lack real world interpretation.

The FIG utility here is given by Equation 4.20,

$$\mathcal{U}_{FIG}(\tau, \theta) = \sigma_1^{-2} \sum_{i=1}^{3} \sum_{k=1}^{15} \left(\frac{\partial \mu_k}{\partial \theta_i} \right)^2, \tag{6.7}$$

where

$$\frac{\partial \mu_k}{\partial \theta_1} = \frac{\mu_k}{\theta_2 - \theta_1} - \frac{D\theta_2}{\theta_3(\theta_2 - \theta_1)} \tau_k e^{-\theta_1 \tau_k}, \tag{6.8}$$

$$\frac{\partial \mu_k}{\partial \theta_2} = \frac{\mu_k}{\theta_2} - \frac{\mu_k}{\theta_2 - \theta_1} + \frac{D\theta_2}{\theta_3(\theta_2 - \theta_1)} \tau_k e^{-\theta_2 \tau_k}, \tag{6.9}$$

$$\frac{\partial \mu_k}{\partial \theta_3} = -\frac{\mu_k}{\theta_3}. \tag{6.10}$$

Note that the σ_1^{-2} term in Equation 6.7 is ignored when implementing the optimal design algorithms since it is a multiplicative constant and as such does not affect the optimal design (see Section 4.2.2).

A consideration of the simulation study is the initial state of the algorithms. Each simulation study will compute 100 independent runs of each algorithm, starting from initial states which are drawn from a uniform distribution with support $[0, 24]^d$, where $d = 15$ is the number of observations in the design. For fairness, each simulation study uses the same initial states.

The algorithms under consideration use differing numbers of utility evaluations at each iteration depending on their tuning choices. To draw fair comparisons, a computation budget will be set in terms of utility evaluations. The computational budget chosen here is 1.8×10^7 utility evaluations, roughly the same number as required by the ACE algorithm under the default tuning choices. Note that for the SGD algorithm a utility gradient evaluation is considered. A computation of a utility gradient incurs a comparable computational cost to a utility evaluation when using automatic differentiation software (see Section 3.3.1). An initial fixed cost will be added for the calculation of the computational graph for the function to return the utility gradient. Thereafter a gradient of the utility can be realised through simple operations which can be compared to the computations required to realise a utility. The timings of running each of the methods will be given to allow comparison of timings despite each algorithm being implemented in different programming languages meaning direct comparison of timings is difficult.

The algorithms used in this simulation study all use various tuning choices meaning that the variability of the estimated utility differs between methods. In order to draw fair comparisons, the computation of the expected utility at the returned designs will be consistent across methods. At each of the returned designs a Monte Carlo estimate of the expected utility will be calculated using $20,000$ realised utility values.

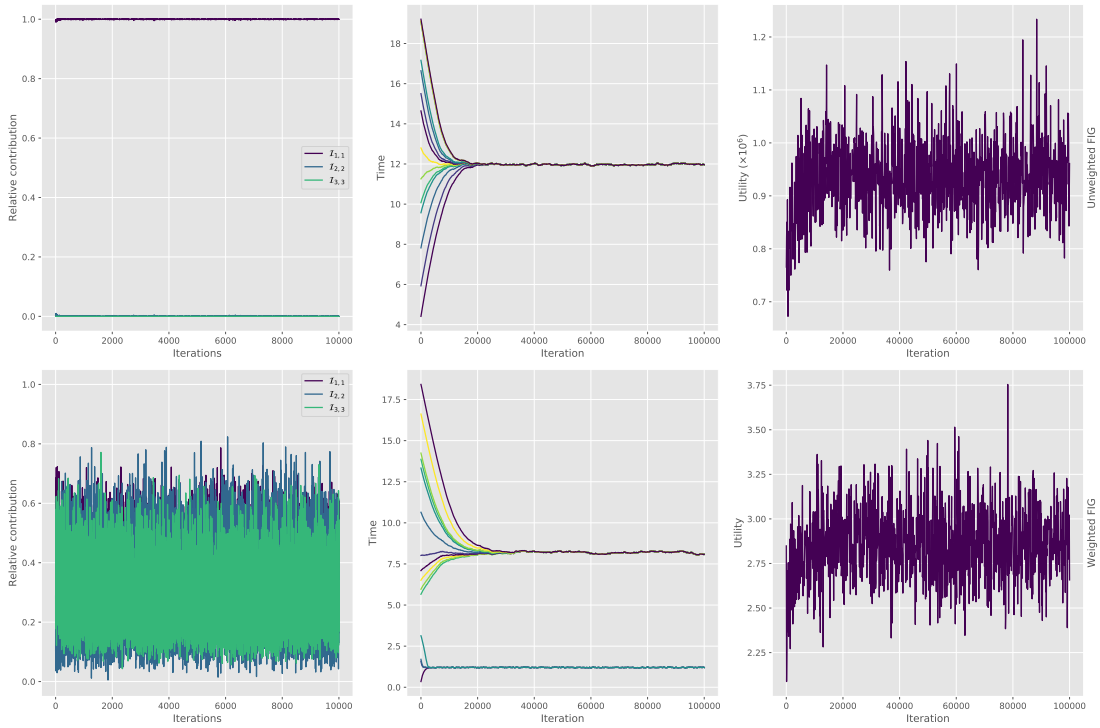

Figure 6.2: *Trace plots of relative contribution to expected FIG (left), each observation time (middle) and utility (right) over the iterations of the SGD algorithm using the unweighted (top) and weighted (bottom) FIG utility for the PK example.*

6.2.1 Scaling FIG contributions

Section 4.2.3 discussed the need for scaling the contributions to the FIG utility. This is particularly important for the current example as the unweighted utility is dominated by the one element of the trace of the information matrix. This can be seen in Figure 6.2 where the first entry in the trace of the information matrix is almost equivalent to the total value of the trace.

To establish suitable weightings for the FIG contribution a pilot run was carried out using adaptive weights. The weights $w = (5 \times 10^5, 3 \times 10^3, 7 \times 10^1)$ were found to be suitable after a pilot run of the adaptive weights algorithm (Algorithm 4.1). These values will be constant throughout the simulation study for the 15 point design. Note that although the weighting changed slightly on different runs using adaptive weighting, the order of magnitude remained stable. These weights will the used in the weighted FIG utility (Equation 4.28) under the reparameterisation as described in Section 4.2.3.

Figure 6.2 shows the effect of weighting on the design and utility over the iterations of the SGD algorithm. Without weighting the observation times all converge to a single time and the utility is of the order of 10^6. Once the FIG is weighted the observations converge to two clusters and the utility is approximately 3, the same as the number of parameters p

Method	Tuning choices
ACE	Phase I only, defaults as given by authors (Overstall and Woods, 2017)
SGD	Iterations $= 1.8 \times 10^7$, $K = 1$
Müller$_1$	$J = 1$, iterations $= 1.8 \times 10^7$, $\sigma^2_{RW} = 1 \times 10^{-2}$
Müller$_2$	$J = 2$, iterations $= \frac{1}{2} \times 1.8 \times 10^7$, $\sigma^2_{RW} = 2 \times 10^{-4}$

Table 6.1: *Settings for the algorithms used in the simulation study under the default computational cost of the ACE algorithm (*1.8×10^7*).*

in this example. The contribution to the utility is also more equally weighted between the elements of the trace of the information matrix.

6.2.2 Tuning choices

This section outlines the tuning choices used for the SGD, ACE and Müller algorithms in this simulation study. Table 6.1 summarises the settings chosen for the simulation.

Tuning choices in SGD

All methods targetting the optimal design require some form of tuning and SGD is no different. The main tuning parameters for SGD are the batch size, K, used to estimate the gradient of the utility function (a larger batch size means a less variable estimate and vice versa), the choice of stochastic optimisation algorithm and the hyper-parameters associated with that choice and the number of iterations the algorithm should be run for, N. The algorithm length could be constrained by computational budget, time or be set at a sufficiently large number of iterations so that convergence can be achieved. An alternative approach could be to set a convergence condition and run the algorithm until this is met.

Throughout this study the Adam optimisation algorithm (see Section 3.1.4) with the default settings will be used. Kingma and Ba (2014) showed these are suitable for a variety of problems. Under a fixed computational budget the only tuning left to consider is the choice of batch size, i.e. the number of realisations used to estimate the gradient. Here batch sizes of 1, 10 and 100 are considered to investigate the effect on the expected utility through the iterations of the SGD algorithm.

Convergence here will be determined as when the expected utility stops improving, determined informally from plots of the utility trace. Figure 6.3 shows that larger batch sizes converge fastest in terms of iterations. In contrast, when considering computational cost, a batch size of 1 appears to achieve convergence fastest. This shows a batch size of 1 is the most efficient computationally and therefore will be used from now on.

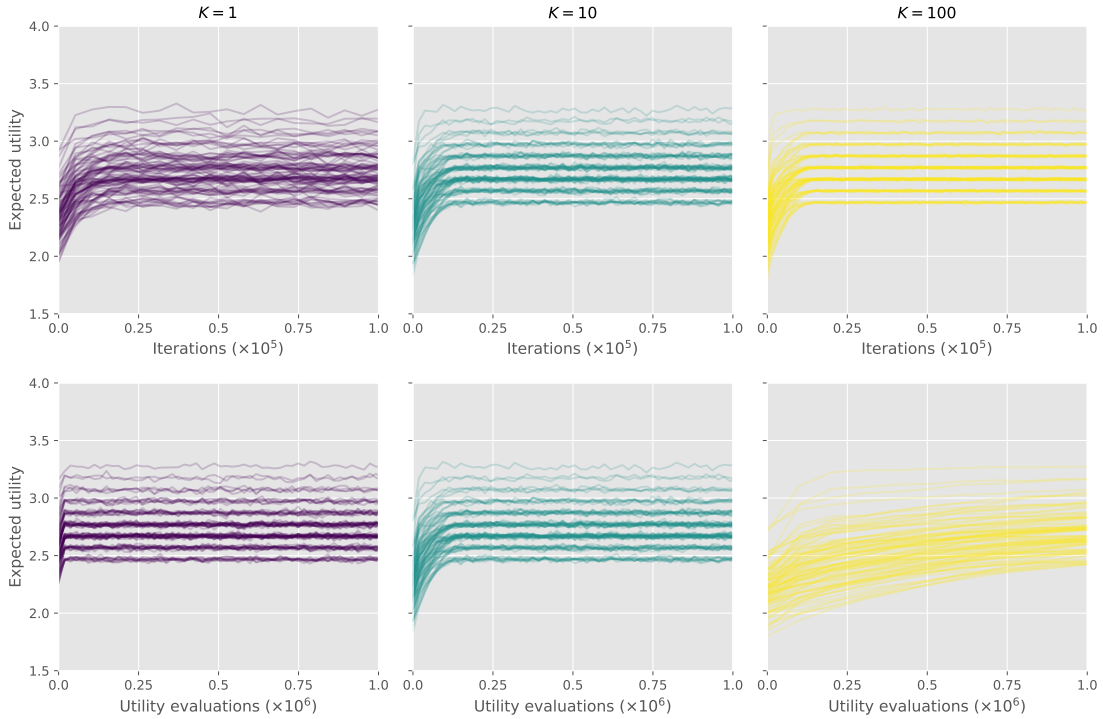

Figure 6.3: *Trace plots of expected FIG utility for the PK example from 100 independent runs of the SGD algorithm against iterations (top) and computational cost (bottom) for various choice of batch size K used in the Monte Carlo estimate of the expected utility (Equation 3.16). The expected utility displayed is computed from the simulated utilities made during the algorithm.*

Tuning choices in ACE and Müller

For this study, the default values for the parameters in the ACE algorithm, as given by the authors, will be used (Overstall and Woods, 2017). The Müller algorithm (Algorithm 2.3) requires tuning the batch size used to power up the utility surface and the MCMC scheme. Batch sizes of $J = 1$ and $J = 2$ will be considered. For this simulation study the MCMC scheme will use a Normal random walk proposal with variance chosen to give a sufficiently high acceptance rate (around 40%). After a short investigation the marginal variance of the random walk σ^2_{RW} was set to be 0.01 and 0.0002 for $J = 1$ and $J = 2$ respectively. The MCMC scheme was run for a fixed number of iterations (computed depending on choice of J) so that the computational budget is exhausted, i.e. to always use the fixed number of utility evaluations.

6.2.3 Comparison between methods

This section looks at the output from the various methods aiming to find the optimal design. The designs and associated utilities will be investigated, allowing comparisons to be drawn. In optimal design it is often the case that the utility is improved when

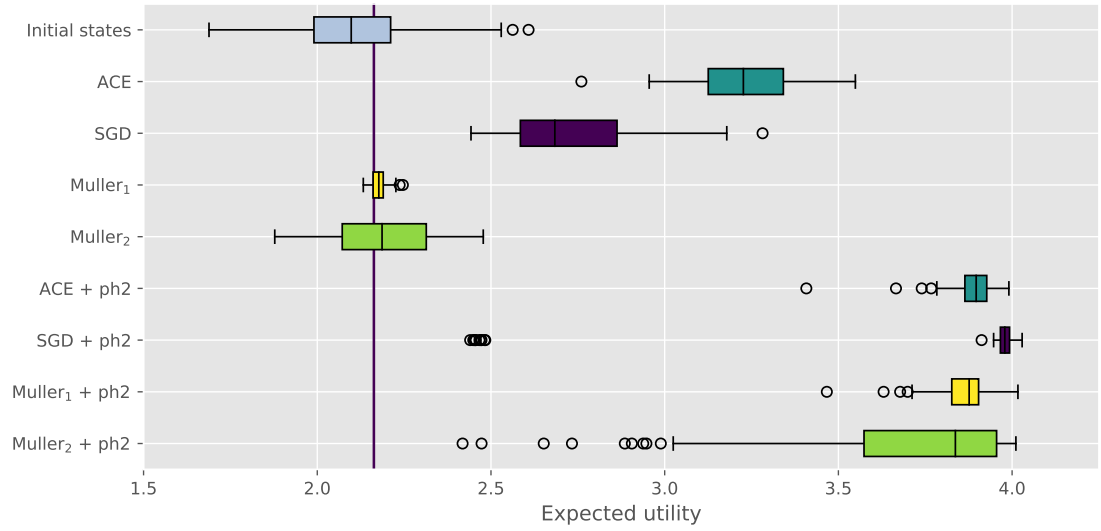

Figure 6.4: *Box plot showing the expected utility of the returned designs from* 100 *independent runs from different initial states. Each expected utility is a Monte Carlo estimate using* $20,000$ *realised utilities at the returned designs. The vertical line is the expected utility of a uniform grid over the search space.*

multiple observations are made at the same time (see, for example, Pronzato and Walter (1985); Binois et al. (2019)). Therefore in this study a point exchange algorithm, namely ACE phase 2 (Section 2.6.2), will be used to investigate how post-processing can affect the returned designs and utilities. The computational cost for the ACE phase 2 using the default settings is fixed and will be applied on the output of all of the methods considered thus fair comparisons can still be drawn.

Figure 6.4 shows the expected utilities at the designs returned by the ACE, SGD, Müller$_1$ (where $J = 1$) and Müller$_2$ (where $J = 2$) methods before and after applying ACE phase 2 as post processing. This shows that all of the algorithms show some improvement over the initial states. Both ACE and SGD show a marked improvement in the expected utility whereas the Müller algorithms only show a slight improvement. Under the fixed computational budget, ACE appears to perform the best as it returns designs with the highest expected utilities. After applying the post processing algorithm, all of the methods of finding the optimal design improve the expected utility of the returned designs. It appears SGD with phase 2 performs the best. Note that there are around 10 initial conditions which return designs with low expected utility (≈ 2.4) after applying ACE phase 2. This suggests that in these cases SGD may have converged to a local maxima rather than the global maximum. Müller$_2$ returns designs that are highly variable in expected utility in comparison to Müller$_1$ so using $J = 1$ appears preferable and will be considered in further investigations.

The different methods return very different designs as shown in Figure 6.5. SGD returns designs with multiple observations at two times, namely ≈ 1.2 and ≈ 8.2. Under the default

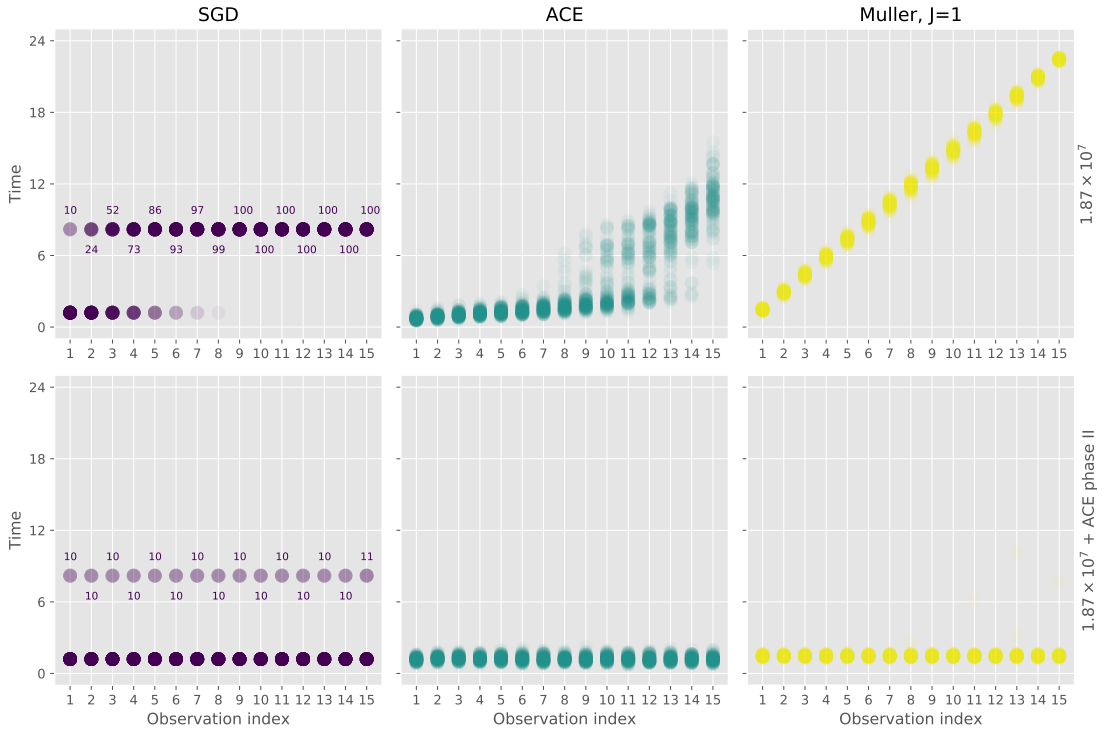

Figure 6.5: *Plots of the returned designs from the various runs of the SGD, ACE and Müller*$_1$ *algorithms when using the FIG utility. Each returned design was sorted by time and indexed by observation. The intensity of the colour represents the degree of repetition over the* 100 *independent runs of the algorithms.*

budget of utility evaluations SGD usually places between one and seven observations at the earlier time with ten of the runs placing all observations at the latter time. The plot of the marginal expected utility in Figure 6.6 indicates that there is a local maxima at ≈ 8 with the global maximum at 1.2 and thus provides evidence that SGD is converging to a local maxima. After point exchange is applied almost all of the observations are placed at time 1.2 with the exception of the 10 runs which do not contain an observation at 1.2 and therefore cannot place any observations at that time. These correspond to the designs which return the cluster of expected utilities at ≈ 2.4.

ACE under the set computational budget returns designs which place some observations at earlier times (between 0 and 4 hours) and up to half placed at later times (between 6 and 10 hours). After applying phase 2, all the returned designs place all observations between 1 and 2 hours. The designs returned from Müller$_1$ after the computational budget was exhausted are almost uniformly spread across the design space. This indicated that the algorithm has not moved far from the initial states which were sampled from a uniform distribution on the design space. Similarly to ACE, after applying the point exchange algorithm all observations are placed at early times.

The returned designs indicate that the optimal design is found when all observations are taken around time 1.2 hours. This repetition of all observations is shown to be optimal in

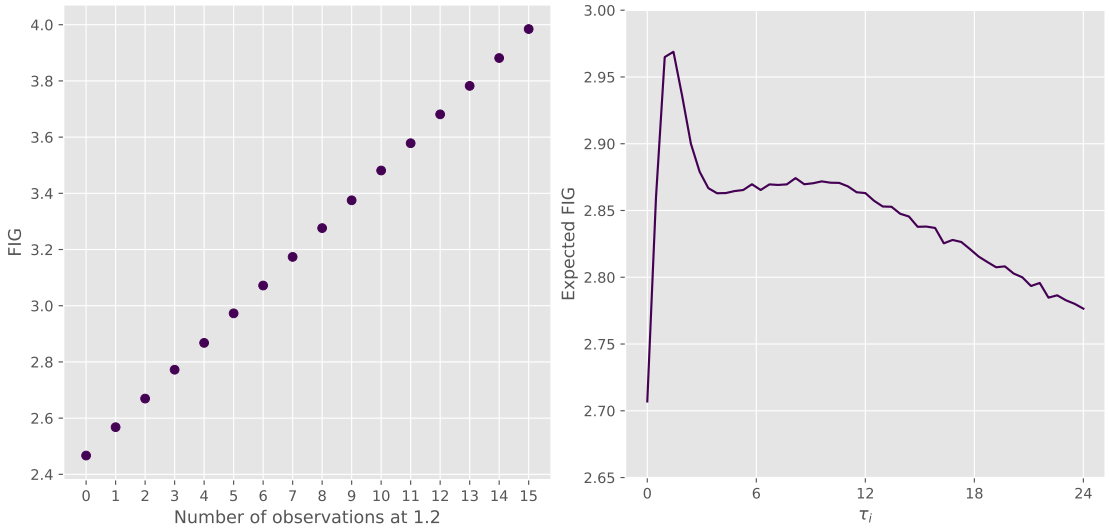

Figure 6.6: *(left) Monte Carlo estimate for the expected utility (using* 2×10^6 *particles) for different proportions of observations at times* ≈ 1.2 *and* ≈ 8.2*. The* x *axis shows the number of observations taken at time* 1.2*. (right) Expected utility as one design point is varied with all others at fixed values: four at time* 1.2*; ten at time* 8.2*.*

Figure 6.6. This intuitively represents an unrealistic design and may be explained by the lack of enforcement of the usual condition in the model that successive observations have to be 15 minutes apart and the behaviour of the utility function. The shortcomings of the FIG utility was discussed in Section 4.2.4 and is investigated further in Section 6.6.

Figure 6.7 displays a traceplot of the expected utility against computation cost for both the ACE and SGD algorithms. SGD appears to converge very quickly, as the expected utility from all 100 runs return stable expected utility values after very few utility evaluations. In contrast, the traceplot relating to ACE shows that the expected utility is still improving, suggesting that more iterations are required before it can be judged to have converged. All of the runs of ACE appear to display very similar trajectories obtaining similar utilities, and so the behaviour does not seem to depend too much on the initial state. SGD displays very different behaviour in that it returns different expected utilities, implying that it converges to different local maxima dependent upon the initial starting point of the algorithm.

6.2.4 Summary

The results of the simulation study show that the optimal design in this situation is all observations should be taken at time 1.2 hours. Other authors have also noted replication of points in the optimal design (Pronzato and Walter, 1985). Boukouvalas et al. (2014) and Binois et al. (2019) argue that repeated observations can be highly informative. Nonetheless this degree of repetition intuitively gives a poor design for this example as the 3 parameters controlling the shape of the concentration curve could not be inferred

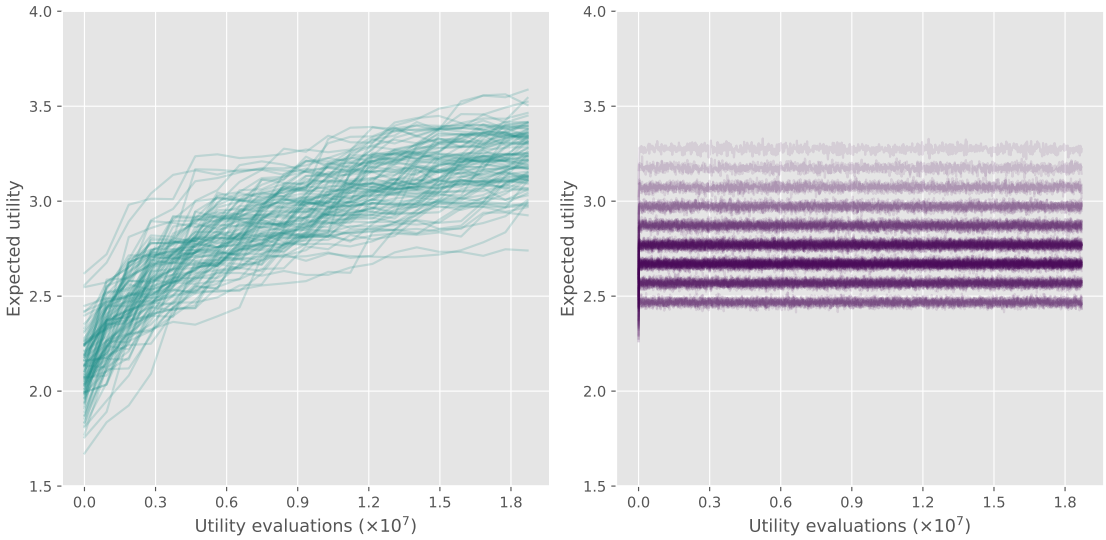

Figure 6.7: *Traceplot of the expected FIG utility through the iterations of the ACE (left) and SGD (right) algorithms for the* 100 *independent runs under the fixed computational budget.*

by making observations at a single time point. Also, in practice, it would be difficult for a practitioner to take multiple samples at the same time. Further investigation into this behaviour related to the choice of FIG utility is discussed in Section 6.6.

In summary, this simulation study has shown that SGD converges after far fewer utility evaluations than ACE representing a better efficiency in terms of computational expense. This is especially beneficial in situations where the utility function is expensive or difficult to compute. Although the convergence is faster (in terms of utility evaluations), SGD is more susceptible to converging to undesirable local maxima rather than the global maximum. This simulation study has shown that this can be resolved by post-processing, at least for this example. A point exchange algorithm can be employed when clusters of observations at a single time are permitted. This was shown to be beneficial in the majority of cases in the simulation study. However there were some cases ($\approx 10\%$) where post processing could not identify the global maximum. Other post processing schemes, such as a perturbation on the returned design, could be employed when observations are subject to restrictions on proximity to each other. Another solution to find the global maximum is to run the algorithm from multiple initial states and select the design which returns the best expected utility. This is also recommended by the authors for ACE.

The simulation study used utility evaluations as a rough proxy of computational cost. For completeness the wall clock timings can also be considered. The approximate timings to run each algorithm were: ACE 130; SGD 7,800; Müller_1 9,000; Müller_2 4,500 (seconds). Note that the implementations used different programming languages and the Müller algorithm timing was dominated by non-utility calculations and hence was heavily influenced by the number of iterations. The computation of SGD would also be subject to some computational overhead related to non-utility calculations at each iteration.

Under a fixed computation budget ACE was the fastest in wall-clock time. Although SGD takes longer than ACE for the full computational budget it takes only 40 seconds to reach 10^4 utility evaluations, at which point it has effectively converged.

Compared to the extensively used SIG utility FIG is faster to compute. Under this pharmacokinetic model (defined in Section 6.1), the FIG utility can be found using Equation 6.7 with Equations 6.8–6.10. The SIG utility, given in Equation 4.3, requires evaluation of the likelihood. When using the programming language `R` for computations, it took an average time of 0.008 seconds for FIG and 0.082 seconds for SIG to produce 1000 simulated utilities. This represents a tenfold improvement in speed of computation for FIG over SIG.

6.3 D-optimality utility

This section carries out the simulation study as described in Section 6.2 but using the $\log|\mathcal{I}|$ utility function defined in Section 4.3.1. Comparisons will again be drawn between the SGD, ACE and Müller algorithms. The Adam algorithm with default tuning parameters will be used for the update step in SGD. The ACE algorithm will be used with its default settings. The Müller algorithm uses the same tuning for the random walk as in Section 6.2: $\sigma^2_{RW} = 0.01$ for $J = 1$; $\sigma^2_{RW} = 0.0002$ for $J = 2$. This gives a sufficiently high acceptance rate. The computational budget will once more be set at the default number of utility evaluations used by ACE.

Figure 6.8 shows the estimated expected utility at the designs returned by the SGD, ACE and Müller algorithms before and after applying ACE phase 2 as post processing. The Müller returns a design with a better expected utility than that at the initial state for the majority of the runs however in some cases the design returned has a worse expected utility than the initial state. ACE and SGD both improve over the initial states with ACE returning the highest utilities before post processing, After applying ACE phase 2 the expected utilities at the designs returned by all algorithms improve. The designs returned by SGD after postprocessing correspond to the highest expected utilities. The designs from 100 independent runs of each algorithm are displayed in Figure 6.9. Both SGD and ACE return designs with observations concentrated around three different times. In contrast, Müller places the designs close to uniformly over the design space. Before postprocessing the number at each cluster is variable however after applying phase 2 of ACE all algorithms place the designs evenly over the clusters with five observations made at each of the three times.

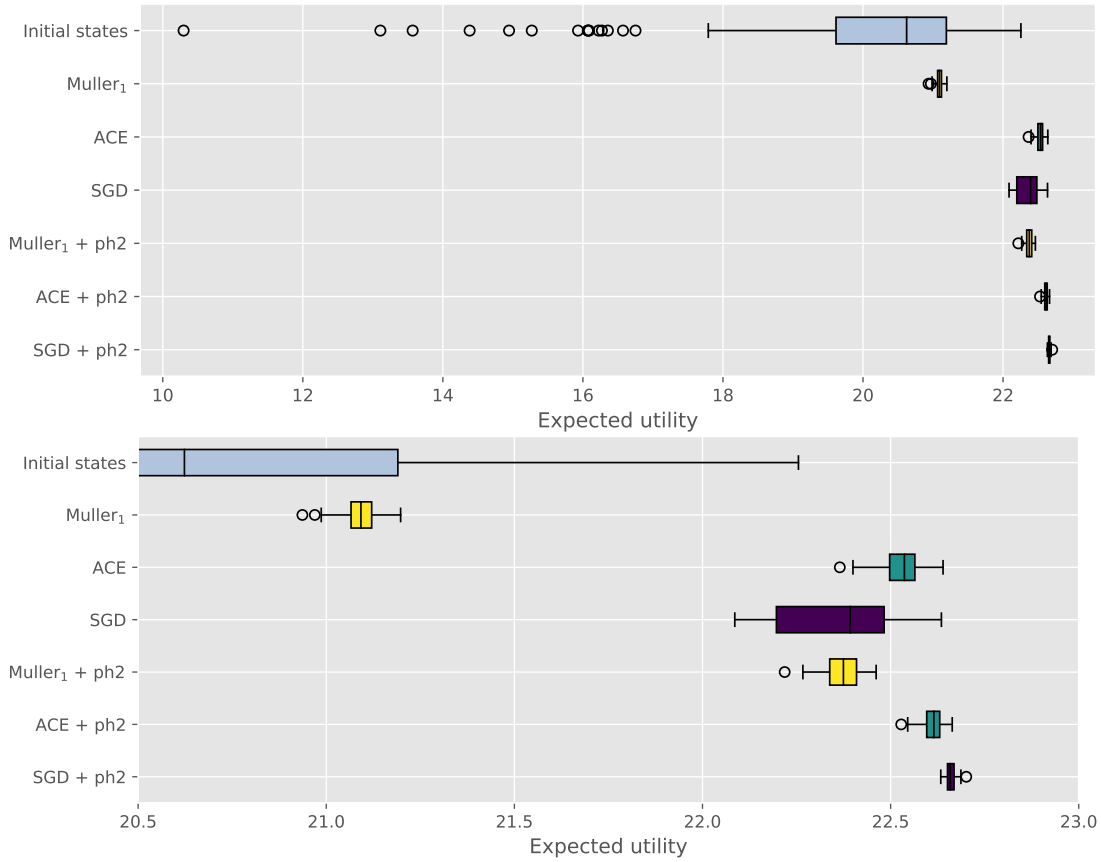

Figure 6.8: *Box plot showing the expected* $\log|\mathcal{I}|$ *utility of the returned designs from* 100 *independent runs from different initial states. The expected utility shown is from a Monte Carlo estimate using* $20,000$ *realised utilities at the returned designs. Both plots are the same but have different x-axis ranges to help inspection of results.*

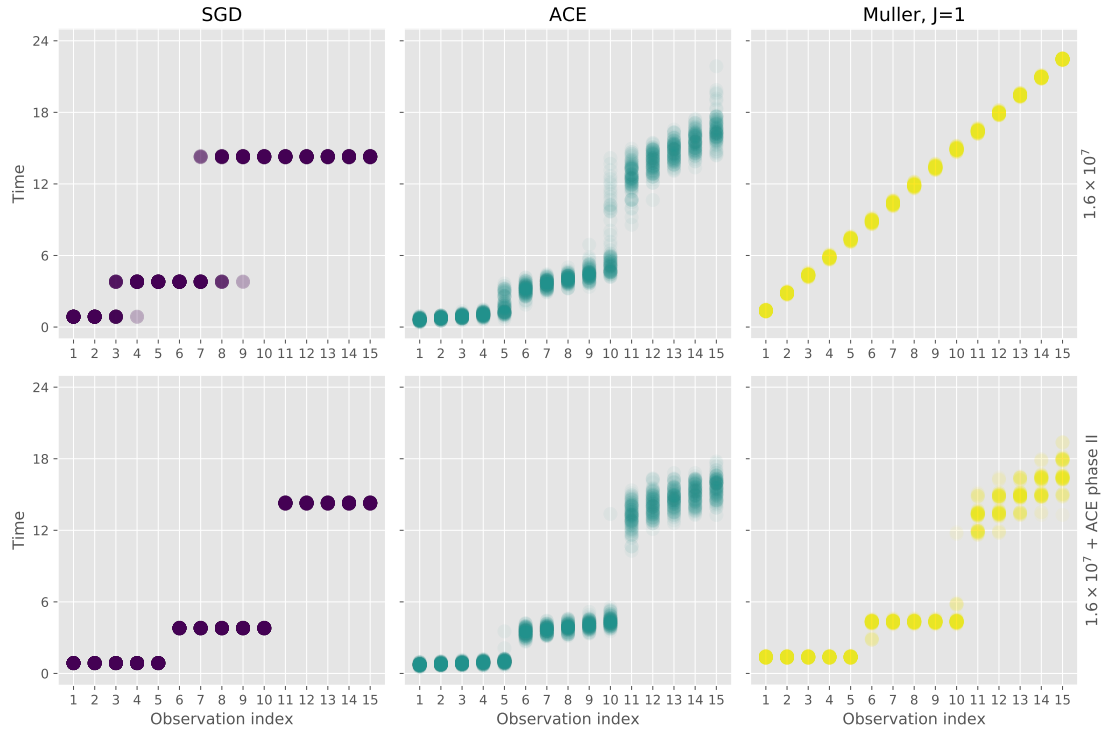

Figure 6.9: *Plots of the returned designs from the various runs of the SGD, ACE and Müller*$_1$ *algorithms when using the* $\log|\mathcal{I}|$ *utility. Each returned design was sorted by time and indexed by the observation. The intensity of the colour represents the degree of repetition over the* 100 *independent runs of the algorithms.*

6.4 AFIG framework

This section considers the optimal design problem under the AFIG utility function. The SGD algorithm will be run independently from 100 different initial states to see how this affects the returned designs. The tuning choices are the same as those used in Section 6.2, i.e. a batch size of 1 for estimating the expected utility and an Adam updates using default parameter choices.

Section 4.2.4 discussed that convergence of Algorithm 4.2 is not guaranteed. To assess if convergence was reached for this example a short pilot run of $100,000$ iterations of this algorithm was carried out. Figure 6.10 shows that both the elements of the weighting matrix B and the observation times of the design have effectively converged after $60,000$ iterations. No cyclic behaviour is shown in the traceplots and so there appears to be no evidence that any modifications to the algorithm are required. The time until convergence is affected by the initial state therefore for the simulation study the computational budget is set to $500,000$ iterations. This ensures that there is enough time for the algorithm to converge, even from an initial state far from a maxima of the expected utility.

Figure 6.11 shows the designs returned when running the Algorithm 4.2 to maximise the expected AFIG utility. Each of the 100 returned designs corresponds to starting the

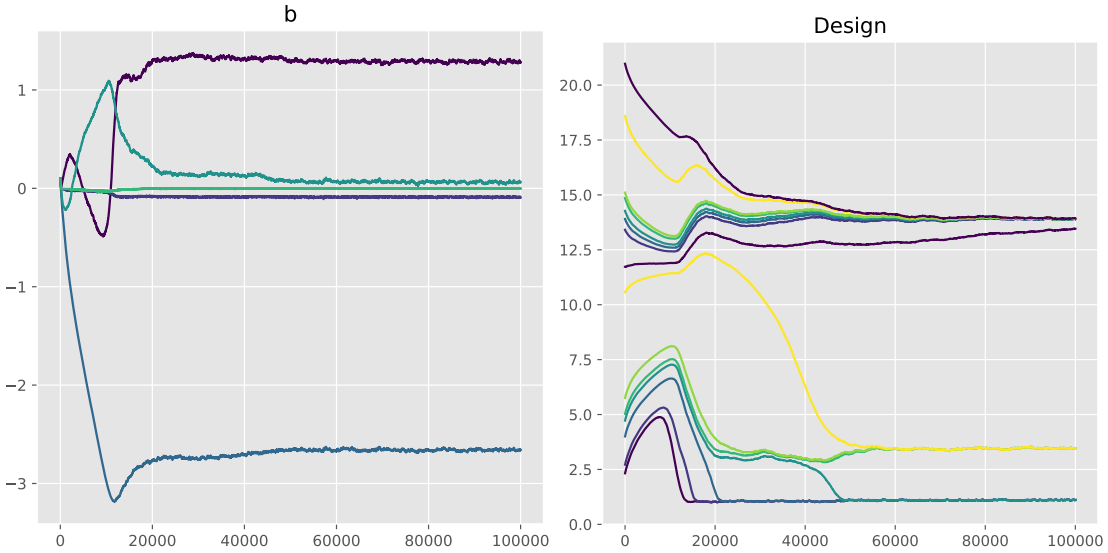

Figure 6.10: *Example traces of the elements of B (left) and the design (right) when using SGD algorithm to maximise the expected AFIG utility.*

algorithm at a different initial state. In all cases the designs returned distribute the observations at three distinct times (approximately times 1, 3 and 14 hours). Intuitively, this should provide more information about the three model parameters than the optimal design found for the expected FIG utility. Section 6.5 explores this in more detail by comparing the posterior distribution at the optimal designs under different utility functions.

In this example the designs returned have repetition of observation times. For the examples using other utility functions (see Section 6.2 and Section 6.3) SGD has converged to local optima of the expected utility and ACE phase 2 is easily applied as a postprocessing step to identify the global optimum. For the AFIG utility, it is unclear how to postprocess the designs as the expected utility surface and objective function are dependent on the weights matrix which should be updated simultaneously with the design. In future work it would be interesting to explore methods trying to identify the global optimum of this objective function.

6.5 Comparison of optimal designs under different utility functions

This section considers the designs returned under various utility functions. The observation times under these different utilities will be considered and an informal comparison of the typical posterior they produce will be conducted.

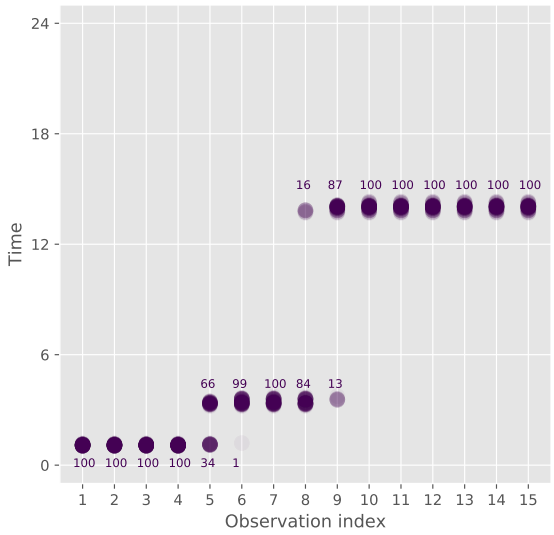

Figure 6.11: *Plots of the returned designs from the various runs of the SGD algorithm using simultaneous updates for the expected AFIG utility. Each run uses a computational budget of* $500,000$ *utility evaluations. Each returned design was sorted by time.*

Optimal design under SIG utility

The ACE algorithm (using both phase 1 and phase 2) was used to find the optimal design under the SIG utility for the pharmacokinetic model. The designs output from 20 independent runs of the algorithm, each starting from a different initial state, are shown in Figure 6.12. All of the designs appear to display a similar pattern with a cluster of around four or five observations around time 1, another cluster around time 4.5 and the remaining observations at later times around time 15. The variability in the time an observation is made is larger for those taken at later times suggesting there is less sensitivity to where these observations taken in relation to the value of the expected utility. For the purposes of the following comparison the returned design which yielded the highest expected SIG value will be considered as the optimal design for this utility.

Comparison

The optimal design under the FIG utility is different from the designs returned by other utility functions as shown in Figure 6.13. It places all observations at a single time point. Under the AFIG, SIG and $\log|\mathcal{I}|$ utilities the designs are clustered together with some observations taken at the same or similar times. In contrast to FIG, the designs under the other utilities converge to multiple clusters of repeated observations at similar times. It is unsurprising that the optimal designs are similar under the $\log|\mathcal{I}|$ and SIG utility functions as the $\log|\mathcal{I}|$ utility is an approximation of the SIG utility (as discussed in Section 4.3.1).

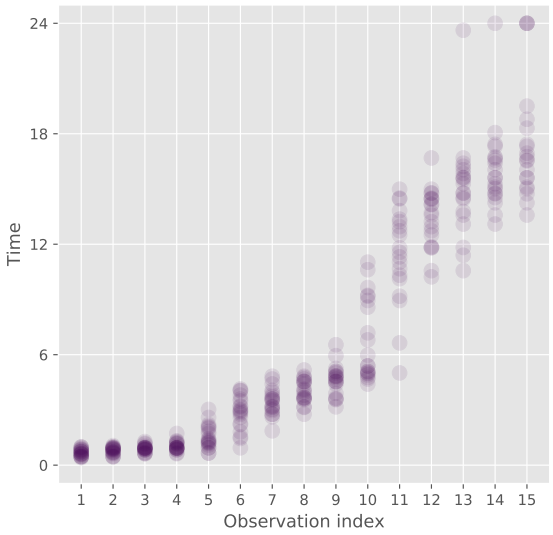

Figure 6.12: *Plots of 20 returned designs (from ACE) for the pharmacokinetic model under the SIG utility.*

The optimal designs under the AFIG, SIG and $\log|\mathcal{I}|$ utilities are intuitively better to learn the model parameters than the optimal found under the FIG utility. Since the model has 3 parameters, a design with observations at 3 or more times should be necessary to learn about all of the parameters. Figure 6.14 demonstrates that these designs typically give posteriors which are much more concentrated. The posterior when using the design returned by FIG shows little improvement in precision from the prior to posterior over the (θ_1, θ_2) and (θ_1, θ_3) margins. The (θ_2, θ_3) margin shows that the marginal posterior of a function of the parameters is highly concentrated. This is evidence that the FIG utility has maximised informativeness for a function of the parameters relating to one of the eigenvalues of the information $\mathcal{I}$ (since the trace of a matrix is equal to the sum of its eigenvalues).

6.6 Discussion

Throughout this chapter, SGD was compared to the Müller and ACE algorithms in a simulation study on the pharmacokinetic example under the FIG and $\log|\mathcal{I}|$ utility functions. The investigation into tuning of SGD in Section 6.2.2 suggested that a smaller batch size was more efficient computationally. The simulations showed that SGD was fast to converge but often to poor local maxima. ACE required a higher computational budget to converge however this often returned designs with higher utilities than those returned by SGD. Post processing could improve the returned design from both algorithms. With post processing, SGD returned designs with expected utilities a bit better than those returned by ACE.

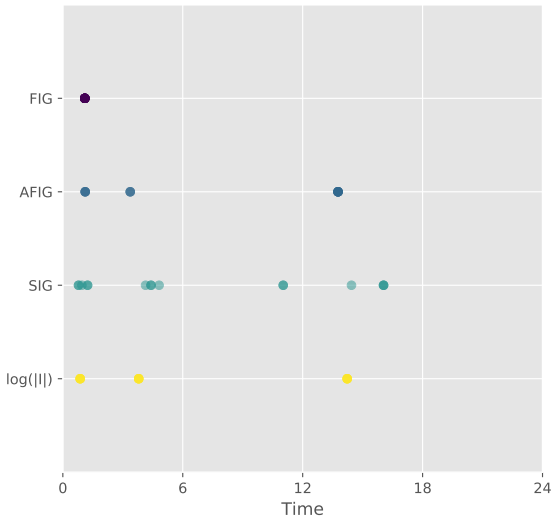

Figure 6.13: *Optimal designs returned under the FIG, AFIG, SIG and* $\log|\mathcal{I}|$ *utility functions for the pharmacokinetic model.*

The optimal design under the FIG utility placed all observations at a single time. This is unrealistic and produced an ill-conditioned posterior. This motivated the investigation into alternative utilities, namely the $\log|\mathcal{I}|$ utility which has less theoretical backing as a Bayesian utility and AFIG, an adversarial variant of the FIG utility. The AFIG utility returned designs similar to those found when using the SIG and $\log|\mathcal{I}|$, all of which gave concentrated posteriors. The objective function for AFIG is more complex than the other utilities as it also depends on a weighting matrix B and requires simultaneous optimisation with respect to both the design and the elements of B. This makes it difficult to draw comparisons between methods and to identify the global optimum. In the future it could be interesting to develop a post processing procedure similar to the point exchange algorithm used in ACE phase 2 that could identify the global optimum of the expected AFIG utility function.

Although only used for the FIG, AFIG and $\log|\mathcal{I}|$ utilities in this chapter, SGD methods can be applied to design problems under a wider range of utility functions. If an unbiased estimate of the gradient of the expected utility function can be obtained then SGD will be applicable.

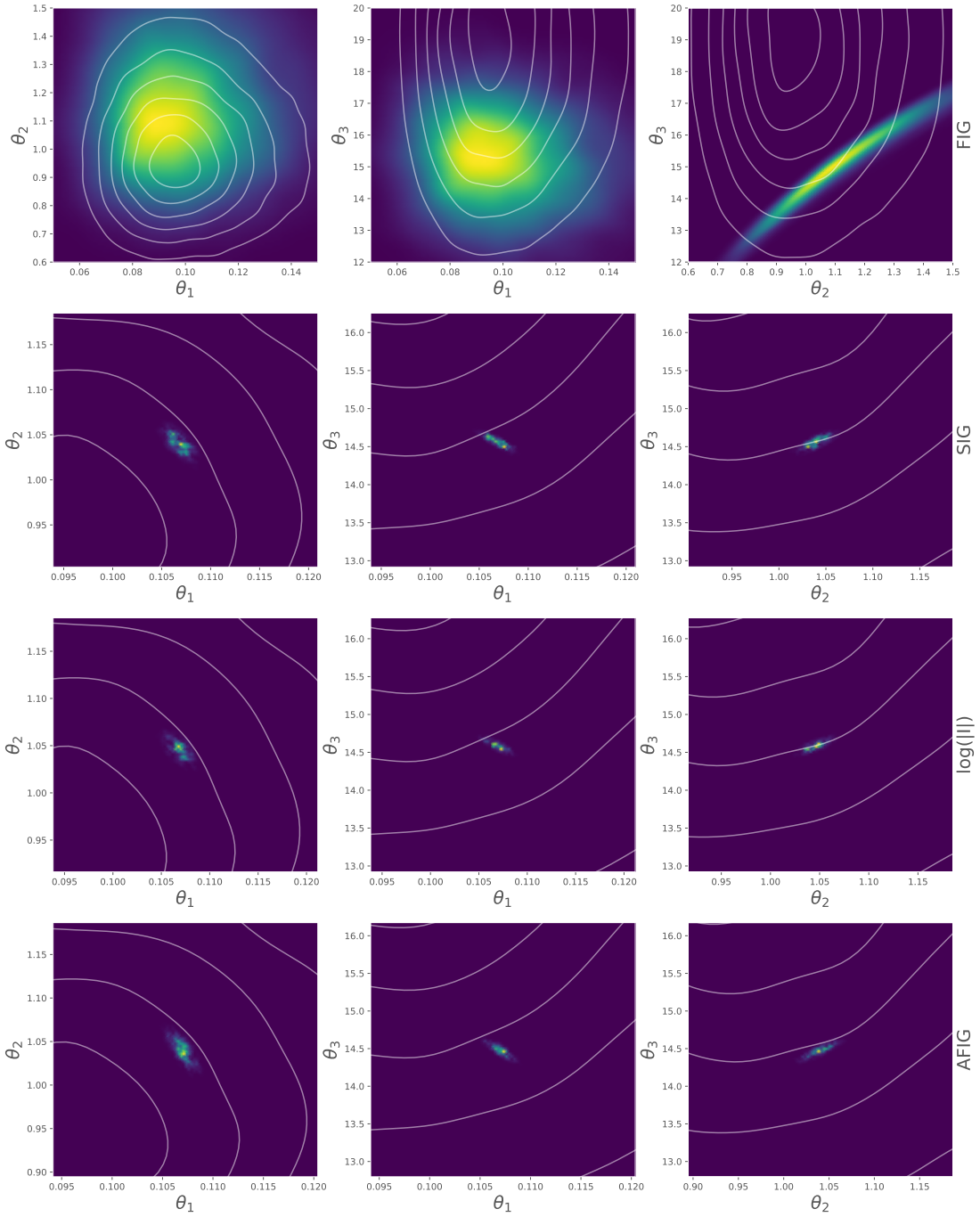

Figure 6.14: *Example bivariate posteriors for the parameters of the pharmacokinetic model for each of the designs in Figure 6.13 for a draw* (θ, y) *from the prior and model. Colour represents posterior density, with yellow showing highest density and purple lowest. From top to bottom, they correspond to FIG, SIG,* $\log|\mathcal{I}|$ *and AFIG. The priors are indicated by the white contours.*

Chapter 7

Surrogate utility function

This chapter looks at the use of surrogate functions for the utility within optimal design. Bayesian optimal design problems can require two stages of expensive approximations. In most problems the expected utility is intractable and thus has to be approximated, usually using Monte Carlo integration with a finite sum of utility realisations. Utility functions in Bayesian optimal design are often functions of a posterior distribution. This increases the computational burden when the posterior distribution is itself intractable and Monte Carlo methods have to be employed to get an estimate. Problems which require this can quickly become computationally challenging. Surrogates to these expensive to evaluate approximations could offer a method of finding the optimal design more efficiently in these settings.

Surrogate functions have been used extensively to emulate functions which are expensive to compute. For example, in computer experiments Gaussian processes are often used as surrogate functions (see, for example, Gramacy, 2020). Booker et al. (1999) outlines a rigorous framework for optimisation of functions via surrogates. This also appears as a step in ACE phase 1 where the practitioner has to choose a surrogate, normally a Gaussian process, to emulate the univariate marginal utility (see Section 2.6.2). Using a surrogate within the calculation of the expected utility function could potentially improve the computational efficiency of searching for the optimal design.

One approach is to construct a surrogate for the expected utility surface directly by fitting a model to some expensive utility evaluations made at lots of different designs. When computing a utility the main computational cost involved is estimation of the posterior distribution, except in cases where the utility function does not require this, e.g. ones using the Fisher information. In most cases the posterior would not be a standard distribution hence would have to be approximated using, for example, importance sampling or Markov chain Monte Carlo (see Section 2.1). Another issue arising from these approximations is that many rely on the ability to evaluate the likelihood function which, in some cases, may

be difficult or impossible. In these cases approximate Bayesian computation (ABC) may be a solution (for a review of ABC see Marin et al. (2012)). This approach scales poorly with dimension however the use of summary statistics offers a partial solution to this problem. Drovandi and Pettitt (2013); Hainy et al. (2014); Price et al. (2016) present work using ABC within experimental design. To approximate the posterior using any of these methods requires a lot of computing resource. This would be required for each realisation of a utility used within a Monte Carlo approximation of the expected utility. Many approximations of the expected utilities would have to be computed at various designs in order to fit a surrogate to the expected utility surface meaning this can be a computationally costly procedure. Examples surrogates could be a Guassian process or a neural network. The optimal design under the surrogate could then be found giving an approximation of the true optimal design. Using a surrogate within existing methods for finding the optimal design would be more computationally efficient however this does still incur a large cost in obtaining the expensive-to-evaluate approximations of the expected utility used to fit the surrogate.

Rather than find a surrogate for the expected utility, an alternative approach could be to find a surrogate for the posterior distribution and use this to obtain cheap realisations of the approximate utility. Overstall et al. (2018) considers using a Laplace approximation for the posterior which is considerably cheaper to obtain than an approximation of the posterior using Monte Carlo methods. Laplace approximations require running an optimisation routine to find the posterior mode and can also be subject to numerical stability issues when returning the Hessian matrix. Although faster than estimating the true posterior directly, this routine can become expensive. This is amplified in the experimental design setting where thousands of utility functions need to be evaluated with a posterior estimate is required for each. For these reasons an alternative approach that could address some of these issues would be useful. Recent work by Papamakarios and Murray (2016) offers a scalable, likelihood free method of producing a surrogate for the posterior distribution using neural networks. This offers a promising method of obtaining quick to compute posterior approximations which in turn allow for computationally cheap realisation of the utility. Section 7.1 gives the background of this method. Bayesian optimal design using a posterior approximation is described in Section 7.2. Section 7.3 and Section 7.4 employ a surrogate for the posterior to get cheap approximate utility realisations using these to target the optimal design. Section 7.5 considers a novel approach to estimating the expected utility surface to identify which region of the design space yields the highest expected utilities.

7.1 Surrogate posterior using neural networks

A surrogate for the posterior distribution reduces the computational burden of calculating commonly used Bayesian utility functions e.g. posterior precision. Surrogates should be fast to compute whilst giving a good approximation to the true posterior. This section describes how neural networks can be used to approximate a posterior distribution. Although they incur a fixed computational cost to train, once trained posterior approximations are very quick and computationally cheap to obtain.

7.1.1 Neural network approximation of posterior

Inspired by biological neurons, a neural network is a connected system of nodes. Nodes belong to layers. Each node has a numerical value with the first layer representing the input variables. The connections between the layers are modelled with weights with each node being a weighted sum of the nodes of the previous layer plus some constant "bias" value. These weights and biases are found by training the network. The output from each node is subject to an activation function such as the identity, sigmoid, hyperbolic tangent, or ramp/relu function. These can introduce non-linearity into the system (Shukla, 2018) and also control the range of values that the network output can take.

Training uses pairs of inputs and outputs of the neural network, with an objective function providing a measure of the error between the network output and the true outputs. This should correlate well with the task the network is performing so that a network with optimal weights gives the output required. Common objective functions include cross entropy and mean square error (Chollet, 2018). The weights and biases in neural networks, denoted ϕ, are trained using stochastic gradient optimisation (see Chapter 3). The gradient of the objective function with respect to ϕ is used in the update step to train the network. For more background on neural networks, including details on implementation, see for example Goodfellow et al. (2016) and Ketkar et al. (2017).

Posterior approximation

Papamakarios and Murray (2016) proposed a fast likelihood free approach to estimate the posterior distribution of the parameter vector using a neural network. Following this work throughout this section an approximate posterior for θ is introduced which follows a normal distribution with mean $\mu = \mu(y, \tau)$ and variance $\Sigma = \Sigma(y, \tau)$,

$$\theta | y, \tau \sim N_p(\mu, \Sigma). \tag{7.1}$$

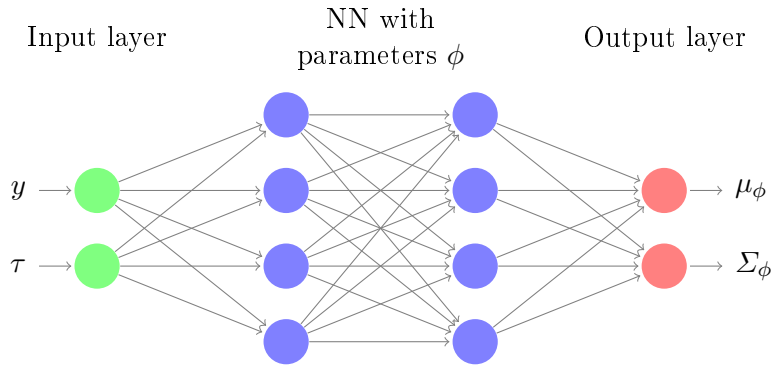

Figure 7.1: *Example structure of a neural network used to estimate the mean $\mu = \mu(y, \tau)$ and variance $\Sigma = \Sigma(y, \tau)$ of the approximate posterior.*

The functions $\mu(y, \tau)$ and $\Sigma(y, \tau)$ are estimated by a neural network.

The assumption that the posterior is a Normal distribution is not always applicable, as the true posterior may, for example, be multi-modal and thus the approximation to the posterior would be poor. In these cases Papamakarios and Murray (2016) extended the approach to fit a mixture of Normal distributions with m components,

$$\theta|y, \tau \sim \sum_{i=1}^{m} w_i N_p(\mu_i, \Sigma_i), \tag{7.2}$$

where the weights w are also outputs from the neural network. This is more flexible with the ability to model more features and thus would give a better approximation to the posterior. For the examples using the simple death model (see Section 5.1) considered in Section 7.3, Section 7.4 and Section 7.5 a single Normal distribution (Equation 7.1) was sufficient to approximate the posterior distribution.

7.2 Bayesian optimal design using a posterior approximation

Since many Bayesian utilities are functions of the posterior, a computationally convenient approach to approximating the utility can be found via a surrogate for the expensive to compute posterior. A surrogate posterior can be found using output from a neural network as described in Section 7.1 and hence a surrogate of the utility. Foster et al. (2019) employs a similar idea, using variational inference to estimate the posterior.

The neural network takes prior predictive data y and corresponding design τ as the inputs, outputting estimates of the mean and variance of the fitted posterior (Equation 7.1). Training data is obtained by sampling N triples (τ_i, θ_i, y_i), sampling τ_i uniformly from the design space, θ_i from the prior, and y_i from the model given τ_i, θ_i.

The neural network is trained by minimising the cross entropy loss,

$$L(\phi) = -\frac{1}{N}\sum_{i=1}^{N} \log[f(\theta^{(i)}|\mu_\phi(y^{(i)}, \tau^{(i)}), \Sigma_\phi(y^{(i)}, \tau^{(i)}))]. \tag{7.3}$$

This is equivalent to minimising the expected Kullback Leibler (KL) divergence (Kullback and Leibler, 1951) from the true posterior π to the approximation f. The KL divergence from π to f is given by

$$KL[\pi||f] = \mathbb{E}_{\theta\sim\pi(\theta|y;\tau)}[\log \pi(\theta|y;\tau) - \log f(\theta|y;\tau)]. \tag{7.4}$$

This quantity is optimised by minimising $-\mathbb{E}_{\theta\sim\pi(\theta|y;\tau)}[\log f(\theta|y;\tau)]$ as the remaining terms are constant with respect to the parameters used in the approximation. The neural network is trained by minimising the expected KL divergence which is approximated via Monte Carlo integration using a finite sample of size N (given by Equation 7.3).

The neural network outputs are an approximation of the mean vector and also a vector which is used to define the variance matrix of the posterior approximation. Care has to be taken so that the estimated variance matrix output from the network is valid. In a simple case where there is only one model parameter the posterior distribution is univariate and a valid variance can be obtained by using a suitable activation function, i.e. the softplus activation. More generally, for models with multiple parameters of interest, a valid variance matrix can be obtained when the network outputs the elements of the matrix which is the Cholesky decomposition of the variance matrix of the approximate posterior. This ensures the variance matrix is symmetric, positive semi-definite and has positive diagonal entries.

Once fitted, the output from the neural network can be used to obtain a computationally cheap approximation of the Bayesian utility functions. For example, an approximation of posterior precision (see Section 4.1) can be obtained by taking $|\Sigma_\phi|^{-1}$. This is demonstrated in Section 7.3, Section 7.4 and Section 7.5.

7.3 SGD using surrogate utility

SGD methods can be easily applied to find the design which results in the maximum expected utility according to the surrogate utility function. First a neural network approximation of the posterior is trained. This is used to define a surrogate utility function. Then automatic differentiation software can be used to implement SGD to return the optimal design by maximising the expected surrogate utility.

To implement SGD the gradient of the expected surrogate utility has to be computed. Equation 3.17 details how an estimated gradient can be found using a Monte Carlo

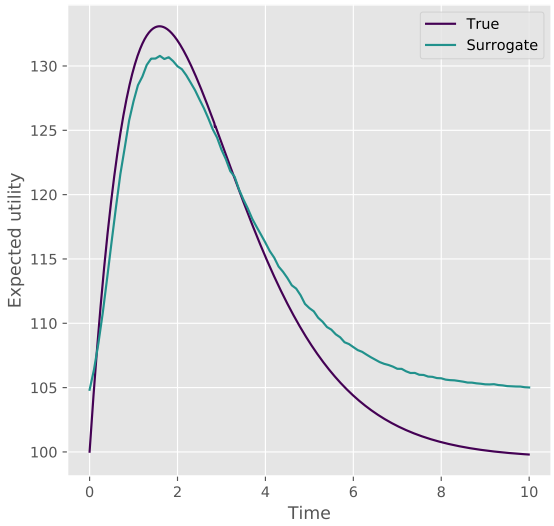

Figure 7.2: *Comparison of the surrogate and true expected posterior precision utility for the simple death model for a one observation design.*

approximation, $\hat{\nabla}\mathcal{J}(\tau) = \frac{1}{K}\sum_{i=1}^{K}\nabla\mathcal{U}(\tau, \theta_i, y_i)$ where (θ_i, y_i) is a draw from the prior distribution for the parameters and the model. This requires calculation of

$$\nabla\mathcal{U}(\tau, \theta, y) = \frac{\partial\mathcal{U}}{\partial\tau} + \frac{\partial\mathcal{U}}{\partial y}\frac{dy}{d\tau}. \tag{7.5}$$

Note that this derivative involves differentiating through y. When the data is generated from a continuous distribution this derivative can be easily computed. If the data is generated from a discrete distribution the derivative does not exist and so a continuous approximation is used. The remaining terms in Equation 7.5 are straightforward to compute.

Simple death model example

Consider the simple death model as described in Section 5.1. This section looks at a design problem where the aim is to identify the observation times which maximise the posterior precision, defined in Section 2.4, of the model parameter θ. In this example all observations should be made between times 0 and 10.

The surrogate model for the posterior that is used for this example follows the procedure outlined in Section 7.1. The network architecture consists of three fully connected layers each with sixteen nodes and tanh activation function. Inputs to the network are draws of y and τ and the outputs are the estimated mean $\mu(y, \tau)$ and variance $\Sigma(y, \tau)$ of the posterior for the model parameter θ. As θ is a single parameter in this example the mean and variance will be scalar quantities (denoted μ and σ^2). The loss function used to train the network is the cross entropy loss as defined in Equation 7.3. Here a one observation and

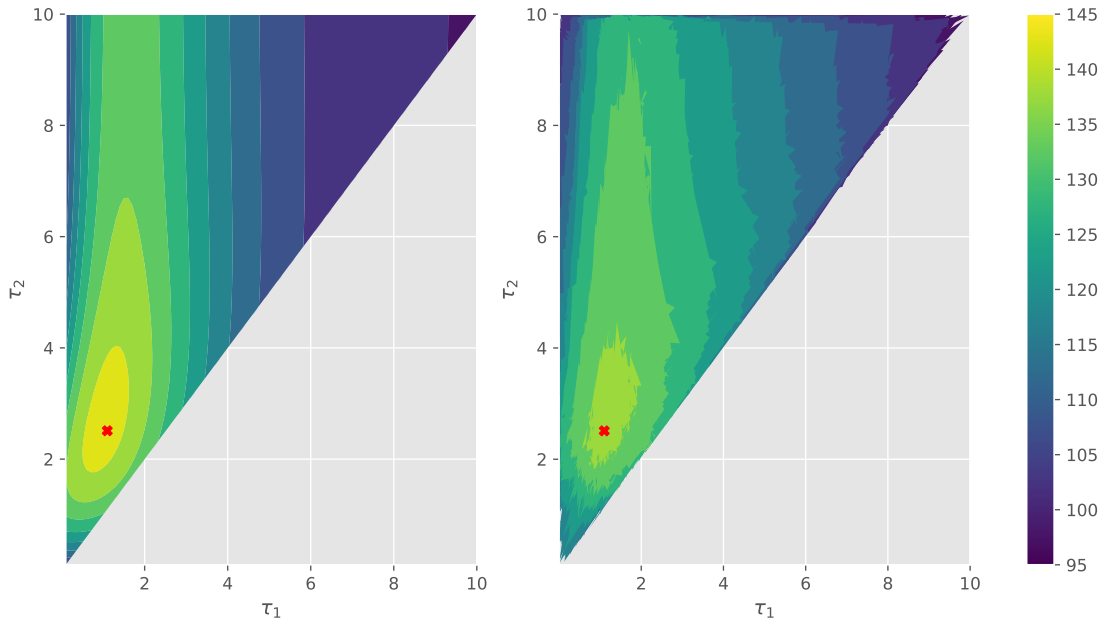

Figure 7.3: *Comparison of the expected posterior precision utility using the true posterior (left) and the neural network surrogate (right) for the simple death model for a two observation design.*

two observation design are considered. For each example the neural network approximation of the posterior was trained using one million designs with associated simulated data y over 100 epochs.

For this model, the practical consideration of computing the gradients via automatic differentiation has be considered carefully. Since the data is generated from a discrete (binomial) distribution this results in discontinuity. Maddison et al. (2016) overcomes this by using a continuous approximation to the discrete density which allows gradient computations to be made. They use a sigmoid transformed logistic distribution to approximate a Bernoulli density, known as a relaxed Bernoulli distribution. A realisation of a binomially distributed random variable can be found by taking a sum of Bernoulli realisations. Simulation of the data in this example could be made via a continuous approximation using a sum of realisations from a relaxed Bernoulli. In practice this was found to give large approximation errors for this example. Alternatively the binomial density can be approximated using a normal density with matched first and second moments. This continuous approximation worked well in the following examples.

Figure 7.4 shows that the optimal design found using SGD in the one observation example is $\tau^* \approx 1.70$. The expected utility at this design is 133.0. Te known optimal $\tau^* = 1.61$ for this model has an expected utility of 133.1. Similar results are seen for the two observation model. Figure 7.4 shows that SGD on the utility using the surrogate posterior converges to $(1.06, 2.48)$ which gives an expected utility close to that of the optimal design under the true utility surface, $\tau^* = (1.10, 2.51)$. The approximation to the optimal design is highly dependent upon the quality of the surrogate utility function. In both examples considered

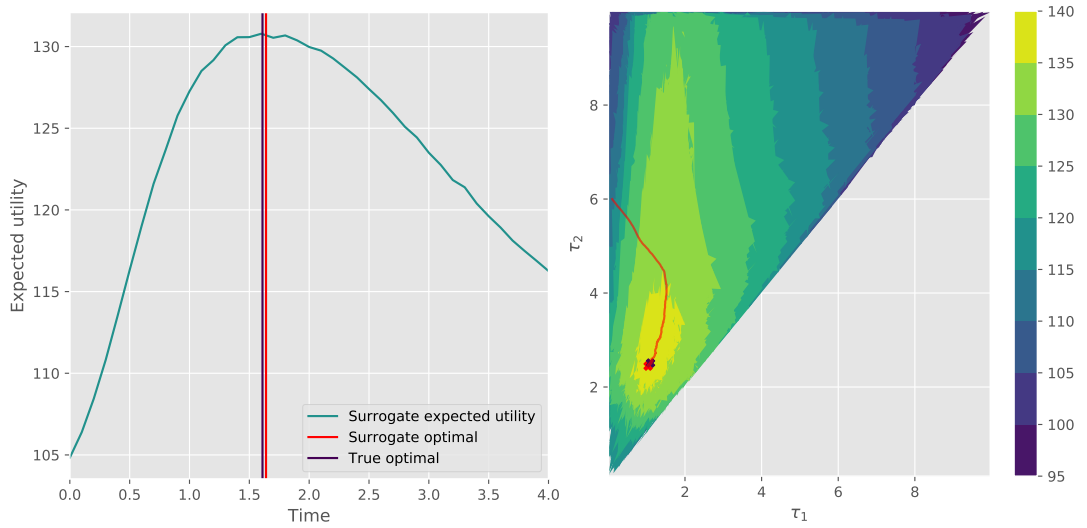

Figure 7.4: *SGD using a surrogate function for the posterior precision utility for the simple death model for a one observation (left) and two observation (right) design. The optimal design found using SGD on the surrogate is shown in red with the path of the SGD algorithms indicated for the two observation case. The known optimal is indicated by the purple marker.*

here the surrogate is a good approximation as it produces an expected utility surface which is similar in shape and with a mode close to that of the true expected utility surface. This can be seen in Figure 7.2 and Figure 7.3. Due to this the optimal found using SGD on the approximated expected utility is close to the known optimal.

Comments

Application of SGD methods is simple, returning the design which maximises the expected utility under the surrogate posterior. This means that the accuracy of the estimated optimum is highly dependent upon the quality of the surrogate estimate of the expected utility. The quality of the surrogate model could be assessed by sampling some points and computing the expected utility. A distance measure between these and the surrogate output (i.e. mean squared error) could be used and the surrogate refined until it became sufficiently close. This could be done using a transfer learning approach (Goodfellow et al., 2016) taking the previously trained network as the initial weights then training with additional data. Note that if the structure of the neural network is not flexible enough, i.e. not enough nodes or layers, then the surrogate may not be able to get sufficiently close.

This method should be simple to apply to higher dimensional designs than those considered here. Difficulties arise in verifying that the optimal design under the surrogate utility is close to the true optimal in a high dimensional design application. This is challenging due to the complexity in finding the true optimal design.

Using SGD returns a point estimate of the optimal design. Alternatively, different approaches which give the practitioner some information on the shape of the expected utility surface could be considered when using the surrogate. For example, the surrogate could be used to identify promising regions of the design space quickly and cheaply. Expensive utility evaluations could then be used in these promising regions to identify the optimal design. This could lead to an increase in efficiency as expensive utility evaluations would not be wasted in sub-optimal regions of the design space. Section 7.4 considers a closely related approach.

7.4 Delayed acceptance Markov chain Monte Carlo

Delayed acceptance MCMC (DA-MCMC) (Christen and Fox, 2005) is an adaptation of the Metropolis-Hastings algorithm (Algorithm 2.1) which uses a two stage procedure for the acceptance step. It aims to reduce the number of expensive evaluations of a target density by filtering bad proposals using a surrogate function for the target density. Algorithm 7.1 describes the DA-MCMC algorithm. In other applications this has been shown to speed up the time of computation and the statistical efficiency of estimating the target (see, for example, Golightly et al., 2015 and Sherlock et al., 2017).

DA-MCMC is implemented via a two stage procedure in the acceptance step of an Metropolis-Hastings iteration. The first stage accepts proposed draws probabilistically based on the value of a surrogate function for the target. This filters out proposed values which have low density in the surrogate model. The promising proposals, those that are not rejected in the first stage, are then accepted or rejected according to a probability using expensive evaluations of the target distribution. Together this two stage acceptance step ensures detailed balance holds and the chain has stationary distribution equal to the target distribution. DA-MCMC aims to speed up inference as bad proposals are screened at the first stage with only promising proposals used in the second computationally expensive stage. Competing MCMC methods are usually compared through speed and statistical efficiency using wall clock time and effective sample size per second (ESS/sec) respectively.

7.4.1 Application to optimal design

The Müller algorithm employs an MCMC based approach to the optimal design problem. Each evaluation of the acceptance probability requires evaluation of the utility which can be computationally expensive. This can be compared to inference problems where the likelihood function is expensive to calculate. DA-MCMC aims to improve the speed or efficiency of such problems and thus can be used with the Müller algorithm to efficiently find the optimal design.

Algorithm 7.1 Delayed acceptance Metropolis-Hastings Markov chain Monte Carlo algorithm with target $f(x)$ and proposal density $q(\cdot)$. An approximation to to the target is denoted by $\tilde{f}$.

1: Initialise $x^{(0)}$.
2: **for** $i = 1, 2, \ldots, N$ **do**
3: Propose $x' \sim q(x|x^{(i-1)})$.
4: Calculate the acceptance probability, $\alpha_1 = \min(1, A_1)$ where

$$A_1 = \frac{\tilde{f}(x')q(x^{(i-1)}|x')}{\tilde{f}(x^{(i-1)})q(x'|x^{(i-1)})} .$$

5: With probability α_1 continue to step 6. Otherwise, set $x^{(i)} = x^{(i-1)}$ and continue from step 2.
6: Calculate the acceptance probability, $\alpha_2 = \min(1, A_2)$ where

$$A_2 = \frac{f(x')\tilde{f}(x^{(i-1)})}{f(x^{(i-1)})\tilde{f}(x')} .$$

7: With probability α_2 continue to step, set $x^{(i)} = x'$. Otherwise, set $x^{(i)} = x^{(i-1)}$.
8: **return** $x^{(0)}, x^{(1)}, \ldots, x^{(N)}$

7.4.2 Example: two observation simple death model

This section considers an example using the simple death model as described in Section 5.1. Observations are assumed to follow a binomial distribution, $P(\tau) \sim Bin(n, e^{-\theta\tau})$. When multiple observations are required then simulations from the model are made using Equation 5.11. This section considers an example searching for the optimal two time point design when using the posterior precision utility function (described in Section 4.1).

The benefit, if any, of using DA-MCMC in the Müller algorithm is investigated. In this example the wall clock timings, statistical efficiency (ESS/s) and efficiency in terms of expensive utility evaluations, herein referred to as the utility efficiency, are considered for various choices of M, the powering up coefficient of the utility surface, in the Müller algorithm.

The cheap to evaluate surrogate utility function $\hat{\mathcal{U}}_C$ will be constructed from the output of a neural network. The network architecture is the same as described in Section 7.3, namely three fully connected layers each with sixteen nodes and tanh activation function, trained for one hundred epochs on one million draws (τ, θ, y) with τ sampled uniformly over the design space $[0, 10]^2$, model parameters θ drawn from the prior and y are samples from the model given θ and τ. As the utility function in this example is the posterior precision, the surrogate utility is computed by taking the inverse of the estimated variance. For this example there is only one model parameter and so the variance $\hat{\sigma}^2$ is a scalar hence

M	Method used in Müller	Average run time (secs)	Average $\mathcal{U}_E$ evaluations	Average ESS per sec	Average ESS per $\mathcal{U}_E$ evaluation
1	MCMC	1312	10000	6.59	0.86
	DA-MCMC	1239	9244	6.37	0.85
2	MCMC	2638	10000	2.84	0.75
	DA-MCMC	2327	8703	2.79	0.75
4	MCMC	5301	10000	1.10	0.59
	DA-MCMC	4121	7674	1.21	0.65

Table 7.1: *The mean results when searching for the optimal two observation design for the simple death model using MCMC and DA-MCMC within the Müller algorithm.*

$\hat{\mathcal{U}}_C(\tau, \theta, y) = 1/\hat{\sigma}^2(y, \tau)$. The neural network is trained before being used in the DA-MCMC scheme so the computation required has a fixed overhead however, once trained, draws of $\hat{\mathcal{U}}_C$ are cheap to obtain.

The expensive utility function $\hat{\mathcal{U}}_E$ used in this example will involve computing draws from the posterior which are then used to estimate the posterior precision. Importance sampling (Section 2.1.2) will be used to estimate the posterior distribution. The prior distribution for θ will be used as the proposal. One hundred thousand particles (N) were used in the importance sampling to ensure that the effective sample size was sufficiently large to avoid any degeneracy. The posterior precision is then estimated from the draws resampled with respect to their importance weight. This procedure is required for each realisation of $\hat{\mathcal{U}}_E$.

Table 7.1 and Figure 7.5 show the results of 20 independent runs of the standard MCMC Müller and DA-MCMC within Müller starting from the same starting states. Each run consisted of $10,000$ iterations. In all cases, Müller using DA-MCMC is faster to run than Müller using MCMC according to the wall clock timings. This improvement becomes larger as M increases. In contrast the statistical and utility efficiency does not show any marked difference for $M = 1$ and $M = 2$. For $M = 4$, using DA-MCMC appears marginally more efficient than using the standard MCMC routine.

7.4.3 Example: five observation simple death model

This section considers the same model as described above however here the optimal design problem requires five observation times. The training of the neural network used to construct the surrogate in this example used the same architecture (3 layers of 16 nodes with tanh activation) and the same number of epochs (100) as the previous example however due to the increase in the number of observations in the design 10^7 training points, (τ, θ, y), were used.

Table 7.2 and Figure 7.6 show the results of 20 independent runs of the standard MCMC Müller and DA-MCMC within Müller starting from the same starting states. As with the

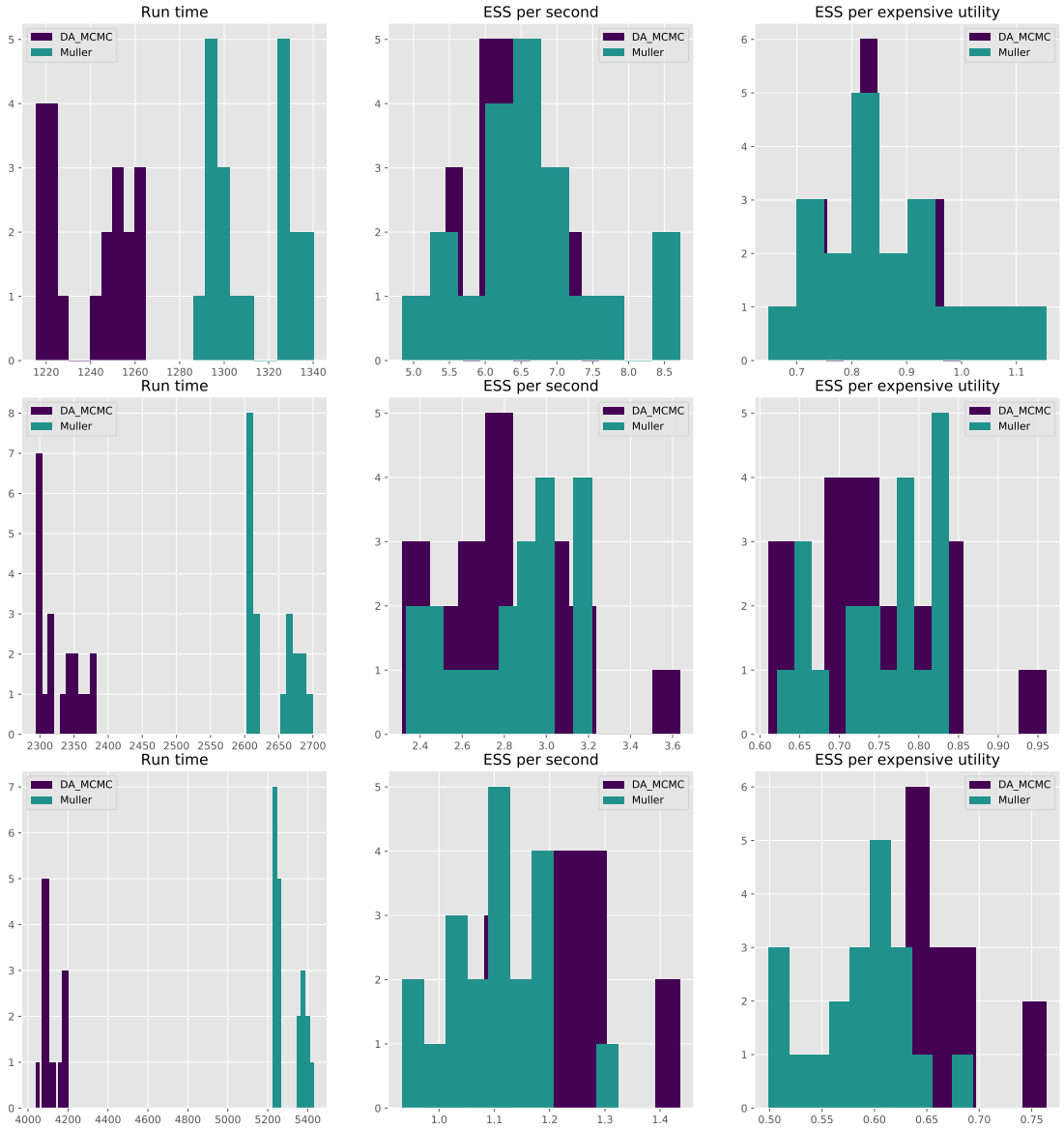

Figure 7.5: *Run time (left column), ESS/s (centre column) and ESS per expensive utility evaluation (right column) for the two observation simple death model for $M = 1$ (top row), 2 (middle row), and 4 (bottom row).*

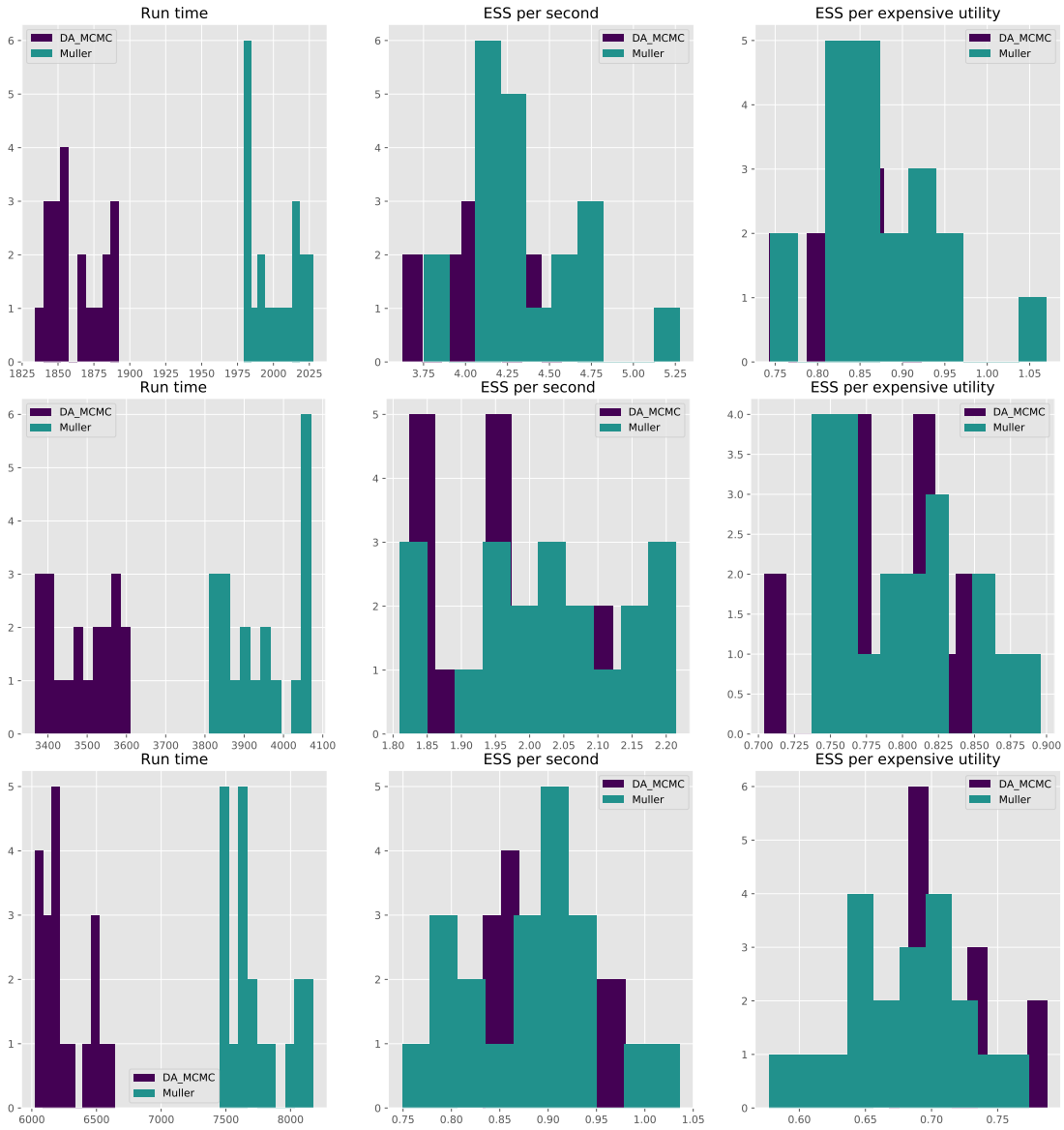

Figure 7.6: *Run time (left column), ESS/s (centre column) and ESS per expensive utility evaluation (right column) for the five observation simple death model for* $M = 1$ *(top row),* 2 *(middle row), and* 4 *(bottom row).*

M	Method used in Müller	Average run time (secs)	Average $\mathcal{U}_E$ evaluations	Average ESS per sec	Average ESS per $\mathcal{U}_E$ evaluation
1	MCMC	2000	10000	4.35	0.87
	DA-MCMC	1862	9187	4.20	0.85
2	MCMC	3942	10000	2.02	0.80
	DA-MCMC	3486	8756	1.97	0.79
4	MCMC	7730	10000	0.88	0.68
	DA-MCMC	6263	8022	0.89	0.70

Table 7.2: *The mean results when searching for the optimal five observation design for the simple death model using MCMC and DA-MCMC within the Müller algorithm.*

previous example, using DA-MCMC is beneficial in terms of wall clock timings. There appears little discrepancy for the statistical efficiency and utility efficiency between the two methods for this example. This is true for all choice of M considered.

7.4.4 Summary

The output from both Müller using MCMC and Müller using DA-MCMC is very similar for both of the examples considered in this section. The efficiency of using DA-MCMC in Müller was assessed using the wall clock time of computation and the average number of expensive utility evaluations. The effective sample size per second and per expensive utility evaluation was considered as a measure of statistical efficiency and utility efficiency respectively. The results showed that using DA-MCMC improves speed of computation however there is no indication that it is better in terms of statistical efficiency or utility efficiency than using MCMC.

7.5 Regression trees

Regression trees provide a method of computing a piecewise constant approximation to a response variable. They can be applied to problems in high dimensional setting, giving a quick approximation to the value of the response in regions of the input space.

First introduced by Morgan and Sonquist (1963), regression trees provide an algorithm which approximates the expected response in sub regions of the explanatory variables. Regression trees sequentially identify thresholds at which to split variables, aiming to produce an approximation which minimises the error in predicting the response. Trees start at a root node. Each decision causes a split where two child nodes are created. The tree grows as more decisions are made recursively until some stopping criteria is met. The terminal nodes of the tree are called leaf nodes. Trees partition the data according to

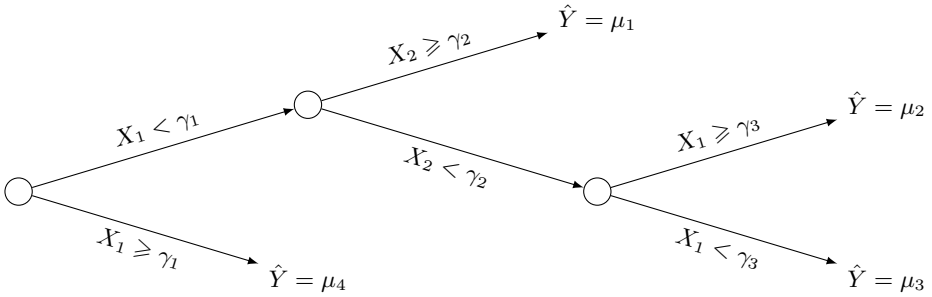

Figure 7.7: *An example regression tree to predict Y from variables X_1 and X_2. Here γ_i for $i = 1, 2, 3$ are the (constant) thresholds at which the splits occur.*

Algorithm 7.2 Pseudo code for tree construction.

1: Start at the root node.
2: Choose the best partition of the dataset according to a splitting rule.
3: If a stopping criterion is met then stop. Otherwise apply step 2 to each child node.

splitting rules so that the leaf nodes contain the most homogeneous outcomes possible (Moisen, 2008).

The threshold at which the data is subset is chosen according to a splitting rule using an impurity measure. This grows the tree by creating two child nodes from the current node. For regression trees this is often chosen to be the mean squared error between the prediction and the data. Alternatively the mean absolute difference (Moisen, 2008) could be used. Compared to the least squares approach this provides a more robust model as it is less sensitive to outliers. However it performs poorly if a data set has a large number of zeros.

The tree recursively splits until a stopping criterion is met. Stopping criteria are enforced so that a tree does not over fit to the data. Typical stopping criteria are satisfied when the reduction in impurity is below a pre-specified threshold (Loh, 2011) or less than a pre-set fraction of the impurity at the root node (Loh, 2014). Alternative stopping criteria can be based on the number of observations in each node or limitations on the structure of the tree, i.e. setting a maximum number of layers or leaf nodes. Once the tree has stopped growing, the value at each of the terminal nodes provides a piecewise constant estimate of the regression function. Algorithm 7.2 outlines the method of constructing a tree with Figure 7.7 illustrating an example of the structure of a regression tree.

There are many extensions which can be considered to the basic method of constructing trees. Friedman et al. (2001) discusses some limitations of trees as well as describing some extensions to overcome some of these difficulties.

Algorithm 7.3 Pseudo code for narrowing down the design space to only include near optimal designs.

1: Define a surrogate for the utility function, $u(\theta, y, \tau)$. Note that ideally the surrogate would be computationally efficient however it could also be expensive to compute, e.g. importance sampling utility estimates.
2: Draw a large number of design points uniformly over the design space and use the surrogate to get an approximation to the utility at each design.
3: Fit a regression tree to get an approximation of the expected utility surface, $\mathcal{U}(\tau)$.
4: Identify the regions of the design space which have the highest expected utility.
5: Either use an existing method to find the optimal design using expensive utility evaluations or define a new surrogate for the utility in the most promising regions and repeat steps 2 to 4.

7.5.1 Application to optimal design

When considering optimal design problems, most of the design space considered will yield designs which are sub optimal. Any search of this space reduces the efficiency of the search algorithm for the optimal design and thus should be avoided if possible. Using a regression tree on realised utility draws will give a piecewise constant approximation to the expected utility surface. This will allow regions of the design space which yield the highest expected utility values to be identified. Other methods could also be considered for approximating the expected utility surface from realised utility draws, such as Bayesian optimisation (Frazier, 2018). Typically these do not scale well and are only applicable for problems up to ten dimensions (Wang et al., 2013).

Once the smaller area of the design space which gives rise to higher expected utilities has been found then a number of algorithms could be used to identify the optimal design. For example, points could be sampled in a grid from areas of high expected utility according to the regression tree. This would allow a simple grid search to be conducted over this grid of points or a particle based approach, such as that of Gillespie and Boys (2019), could be used to efficiently find the optimal design using expensive utility evaluations. Alternatively, the surrogate model may be refined for the smaller region and the procedure of fitting the regression tree applied recursively until the space of near optimal designs was deemed sufficiently small or the expected utility sufficiently flat by the practitioner. Algorithm 7.3 outlines this procedure.

7.5.2 Example: one observation simple death model

Consider the simple death model where a design of one observation is required. The same surrogate as described in Section 7.3 will be used throughout this example to obtain cheap utility evaluations. The regression tree will be fit using the existing software Scikit-learn

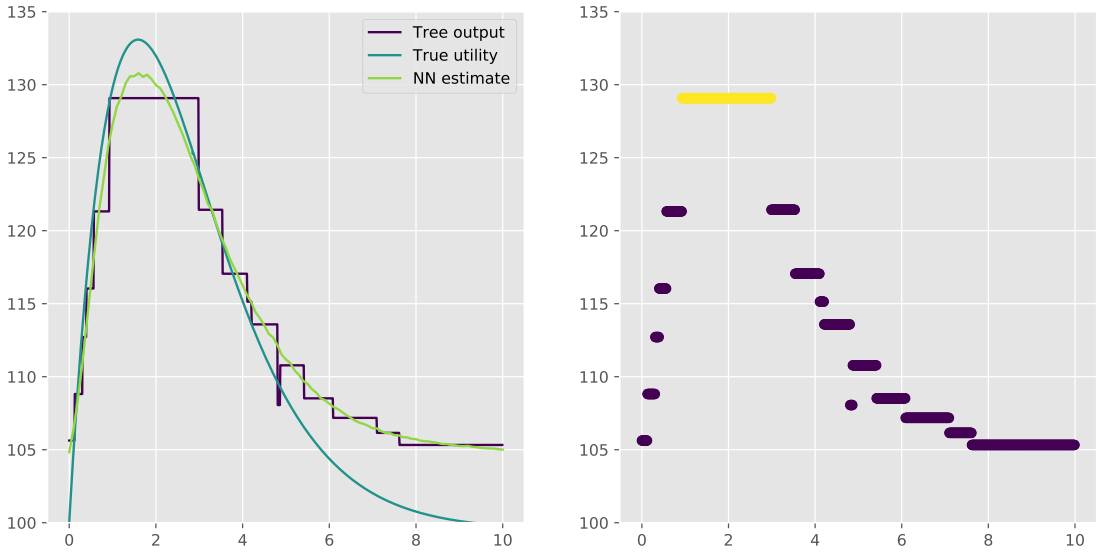

Figure 7.8: *The true expected utility surface found using numerical integration and regression tree prediction (left) for the expected utility for the simple death model with a one observation design. Areas which result in the top quartile (shown in yellow) for expected utility according to the regression tree output are shown in the plot on the right.*

(Pedregosa et al., 2011) using the mean squared error as the measure of impurity. The tree is grown until there are no new splits that result in a decrease in impurity or until all leaf nodes contain only one sample. After, the region of promising designs has been found using a fitted regression tree, designs will be sampled from this space and the optimum identified using the particle based approach of Gillespie and Boys (2019) (see Section 2.6.3).

Figure 7.8 shows the tree fitted on cheap utility evaluations from the surrogate function. When comparing this to the surrogate, the tree appears to fit well with both the tree and surrogate closely matching that of the true expected utility. Also displayed in Figure 7.8 is a plot showing where the expected utility estimated by the regression tree is in the top quartile of sampled responses. This allows the promising areas of the design space to be identified. Here the known optimum is contained in the selected region.

Here the Gillespie Boys algorithm will be used to target the optimal design with expensive utility evaluations made using importance sampling to estimate the posterior (with the same settings as outlined in Section 7.4.2). The Gillespie Boys algorithm here uses a sample of 100 points from the selected region, using 10000 utility estimates in each stage keeping designs which yield the top 50%, 25% then 12.5% of estimated expected utilities. Note that these tuning parameters have been selected fairly arbitrarily in order to illustrate the method. The output of this method can be seen in Figure 7.9. This identifies points close to the known optimum of $\tau^* = 1.61$, visiting them much more frequently than the sub optimal designs.

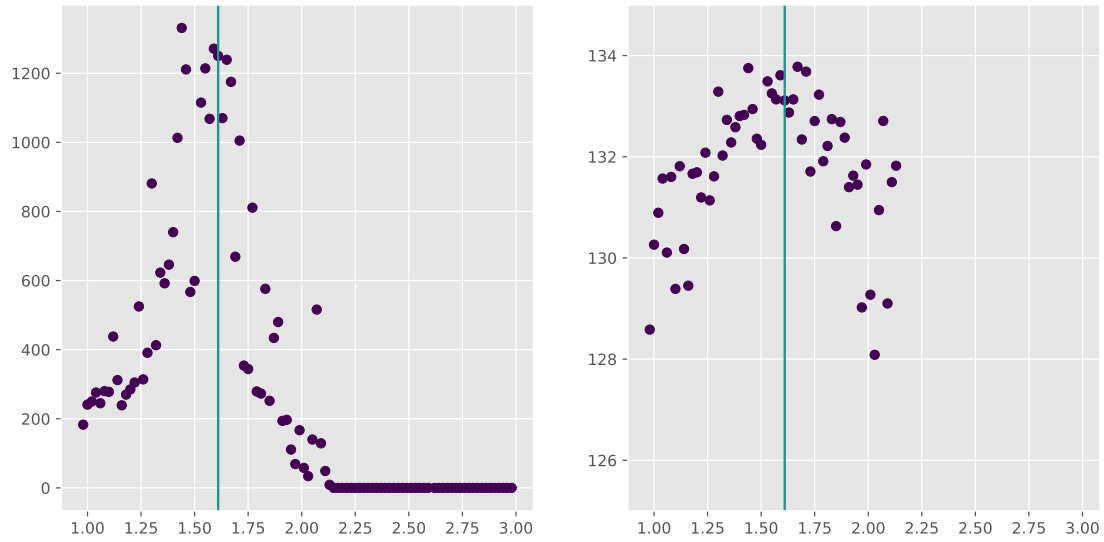

Figure 7.9: *Output of the Gillespie Boys algorithm showing the number of times a sampled design was visited (left) and the estimated expected utility (right) for the simple death model with a one observation design. Optimal design* $\tau^* \approx 1.61$ *indicated by blue vertical line.*

7.5.3 Example: two observation simple death model

This approach is illustrated through application to the simple death model, as described in 5.1, where the aim is to identify the optimal two observation design. The optimal design will be the one which maximises the posterior precision, defined in Section 2.4, of the model parameter θ. The observations are ordered chronologically, $\tau = (\tau_1, \tau_2)$ for $\tau_1 \leqslant \tau_2$. In this example all observations should be made between times 0 and 10.

The surrogate model used for this example follows the procedure outlined in Section 7.1. First a neural network is trained on some simulated data from the prior predictive distribution to predict the mean, $\hat{\mu}$ and variance $\hat{\sigma}^2$ of the posterior which is assumed to follow a Normal distribution. The output is then used to estimate the utility function, $\hat{\mathcal{U}}_P(\tau, \theta, y) = 1/\hat{\sigma}^2$. The same trained network as was used in Section 7.3 and Section 7.4.2 is used in this example.

To fit the regression tree, 10^4 designs are sampled uniformly over the design space and an estimate of the utility is computed via the surrogate model for each. Figure 7.10 shows the approximate expected utility surface predicted from the regression tree, $\hat{\mathcal{J}}(\tau)$, against the actual expected utility surface, $\mathcal{J}(\tau)$. Although the value of the utility differs, the approximation appears to have a very similar shape to the true expected utility. The areas of high utility correspond and thus the computationally cheap approximation to the expected utility will identify regions in the design space where designs are near optimal.

Similarly to the example considered in Section 7.5.2, here the Gillespie Boys algorithm is applied to particles sampled from the promising regions identified by the regression

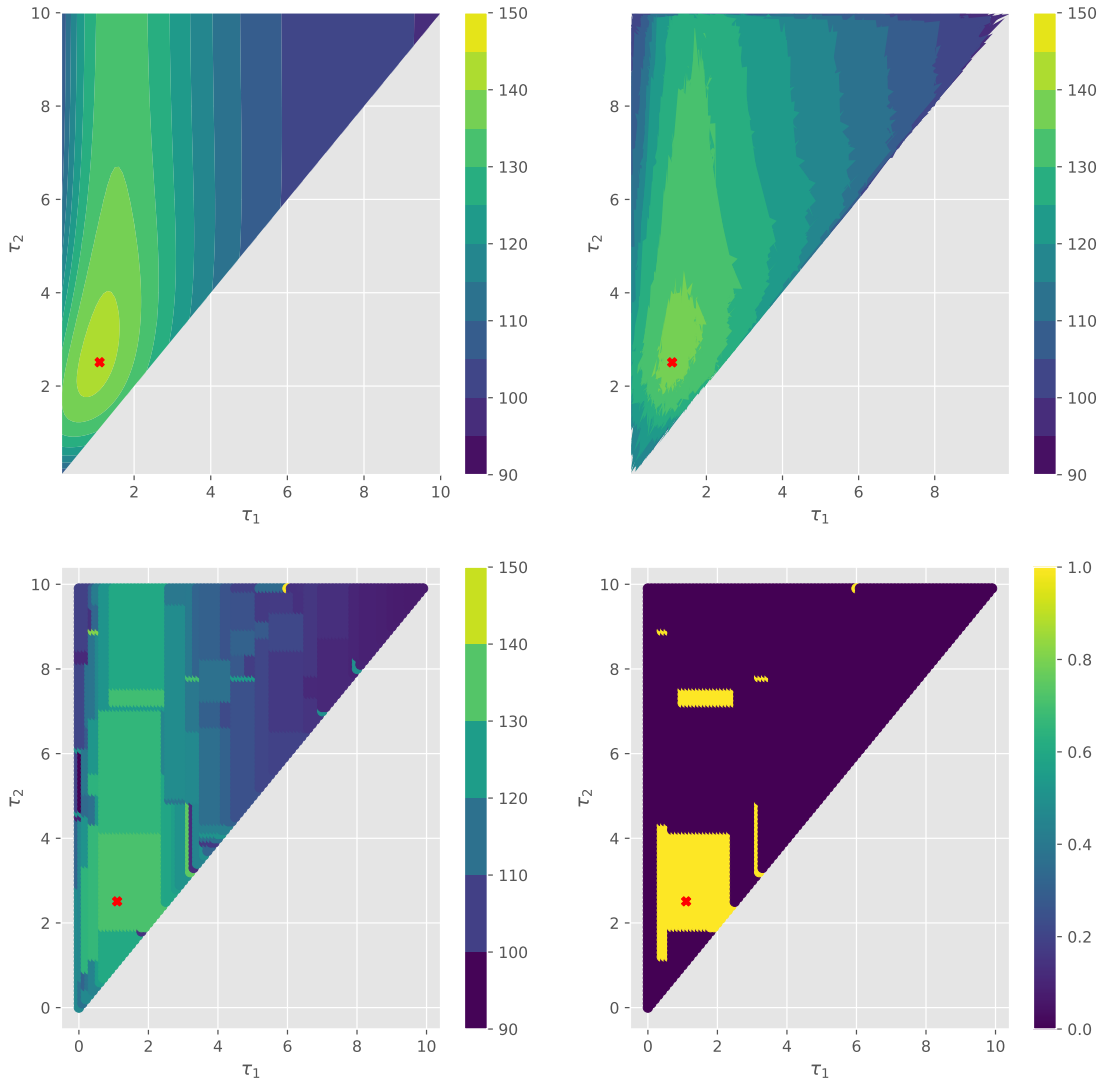

Figure 7.10: *The true expected utility surface found using numerical integration (top left), the neural network surrogate (top right) and regression tree prediction (bottom left) for the expected utility surface in the simple death model with a two observation design,* (τ_1, τ_2)*. Also included is a plot showing areas which give expected utilities in the top quantile of the regression tree output (bottom right). Areas which result in the top quantile (shown in yellow) for expected utility according to the regression tree output are shown in the plot on the right. The optimal design* $\tau^* = (1.1, 2.51)$ *is indicated by a cross.*

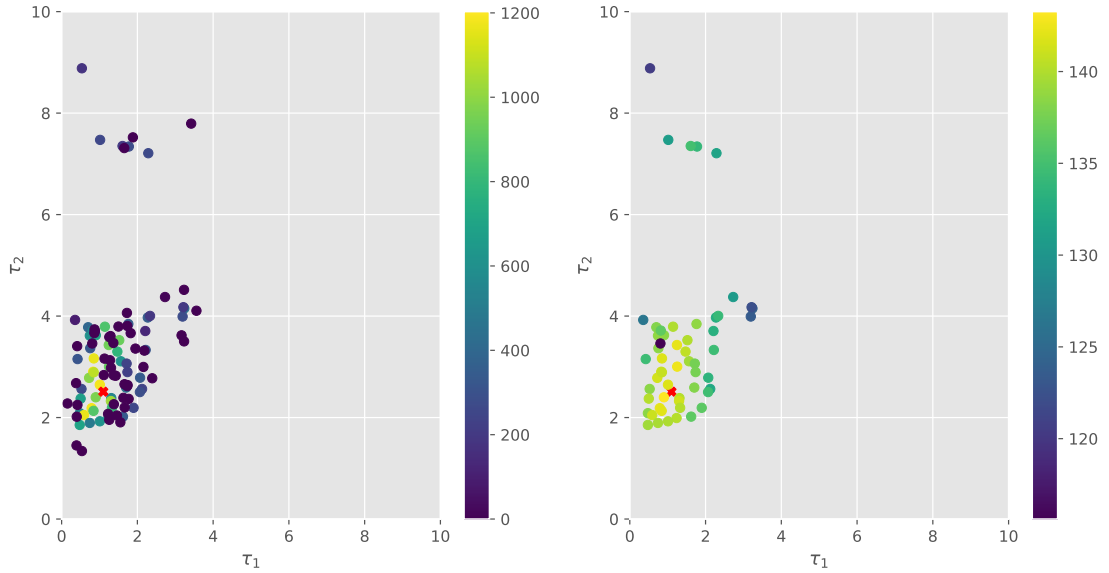

Figure 7.11: *Output of the Gillespie Boys algorithm showing number of times the particle has been visited (left) and the estimated expected utility (right). The optimal design* $\tau^* = (1.1, 2.51)$ *is indicated by a red cross.*

tree. Again, expensive utility evaluations will be computed from a posterior estimated via importance sampling. Figure 7.11 shows the output from the algorithm. It shows that the most visited points are those around the true optimal.

7.6 Discussion

This chapter has considered constructing a surrogate function for the expected utility based on using a neural network to approximate the posterior distribution. This approach incurs a one off computation to train the neural network after which realisations of utilities which are functions of the posterior can be made easily and cheaply. These cheap estimated utilities can then be used to target the optimal design.

The main cost involved with defining a surrogate is training the neural network to predict parameters in the approximate posterior. This can have a significant computational cost since many simulated (τ, θ, y) are required however once trained approximate posteriors can be realised easily from simulated (τ, y) draws. This means that the computational cost of obtaining surrogate utility realisations is minimal, requiring only simple calculations.

The neural network approximation of the posterior parameters could be refined more. This could potentially be required for more complex models than those considered in this chapter. One approach could be to evaluate the expensive utility at some points in the design space and observe how far away the surrogate utility under the approximate posterior is. If the distance is outside of a user defined threshold then more (τ, θ, y)

draws could be made and the network trained further starting from the weights of the previously trained network similar to transfer learning (Goodfellow et al., 2016). This could be repeated until the surrogate utility gave values close enough to those found under the expensive utility.

Another way the network could be improved is by approximating the posterior as a mixture model. Papamakarios and Murray (2016) suggest using the neural network to approximate the weightings of the various Gaussian distributions as well as the respective means and covariances. This provides a more flexible method to approximate the posterior and may be required for a good approximation if the posterior has an unusual shape. The extra complexity could also mean that more data is required to train the network sufficiently.

Use of a surrogate utility that is cheap to evaluate should allow improvements in efficiency of finding the optimal design as cheap utility draws can used in existing methods to approximate the optimal design (as in Section 7.3), in adaptations to existing methods (Section 7.4) or to reduce the design space so that expensive utility evaluations are not wasted in sub optimal regions (Section 7.5). For the approaches considered in this chapter, targetting the optimal design of the surrogate expected utility and reducing the design space appear the most promising. Using cheap utility evaluations within the Müller algorithm did reduce the wall clock time to run the algorithm but did not appear to increase its efficiency. Future work could explore the approaches identified as promising and evaluate if the there is an increase in overall efficiency of these methods accounting for the training of the neural network.

Chapter 8

Conclusion

This thesis has considered the problem of optimal experimental design in the Bayesian framework with the main interest in finding computationally convenient methods of finding the optimal design. Firstly, stochastic gradient optimisation was considered as an alternative to existing methods. This is scalable to high dimension designs as well as efficient in terms of utility (gradient) evaluations required for the algorithm to converge to a locally optimal design. Another way of reducing the computational burden was to consider utility functions that are cheap to compute. Often Bayesian utilities are functions of the posterior distribution which are intractable requiring an expensive estimation. The FIG utility was considered as an alternative Bayesian utility which is cheap to compute for models with observations from some standard distributions. Finally surrogates were considered as alternatives to the expensive to compute approximations of the expected utility.

8.1 SGD methods

Throughout this thesis comparisons have been drawn between SGD and the existing methods of Müller and ACE to find the optimal design. The Müller algorithm has some theoretical backing to show that it does target a density whose margin in τ is proportional to the expected utility. It does not scale well to higher dimensional designs and, in the examples in this thesis, it was difficult to select good tuning parameters of the proposal density. The ACE algorithm addresses some of the issues of the Müller algorithm. The default parameters seem to work well for most examples and it is scalable to higher dimensional designs than Müller . The limitation of ACE is that it requires a lot of utility realisations and that it can take a long time to converge especially if the observations in the design are highly correlated.

8.1.1 SGD for Bayesian optimal design

This thesis has used stochastic optimisation to target the optimal design. This was shown to be faster and more efficient than ACE and Müller in terms of utility realisations required to converge. SGD is scalable and easy to implement using existing automatic differentiation software given that the gradient of the utility can be computed. Here all experiments were run using Tensorflow on a CPU however further speed improvements could be made by using Tensorflow on a GPU due to more efficient parallelisation.

SGD is prone to converging to local optima. This is undesirable as in practice multiple runs from different initial conditions have to be made to increase the probability of identifying the global optimum, in turn increasing the computation cost of obtaining an estimate of the optimal design. In examples where the returned design can contain repeated observations existing methods, such as the point exchange algorithm of ACE phase 2, can be used as post processing to assess how many observations should be made at each unique time point. This was shown to target better designs in the example considered in Chapter 6. There are various methods that could help SGD escape local optima which could be considered in future work. These include using random perturbations in the update step (Robert and Casella, 2013), tempering methods (Ye et al., 2017) or periodically conducting a line search (such as one iteration of ACE phase 1) within the SGD algorithm. These adaptations may provide more chance of escaping bad local optima however it would still remain advisable to run the algorithm from multiple initial conditions to identify the returned design which has the highest expected utility.

SGD returns a point estimate of the optimal design. This is desired by the practitioner however this gives no information about the shape of the expected utility surface and thus the sensitivity to perturbations of the design points. In practice, measurements through an experiment will be taken close to but not exactly at the time of the optimal design and so quantification of how sensitive the expected utility is to these perturbations would be useful. Gradient optimisation methods do not provide such information. Future work could consider this, e.g. by looking at the second derivatives near the optimum. The output of the Müller algorithm are draws whose margin in τ is proportional to the expected utility surface and so the shape of the utility surface can be easily assessed by the practitioner. For ACE a point estimate of the design is output however at each iteration of phase 1 the estimated marginal expected utility is found for each element of τ. For the latter iterations this could be output allowing the practitioner to see the shape of the estimated margins of the expected utility surface.

8.1.2 FIG utility

Section 4.2 considered a computationally convenient utility function, based on the Fisher information matrix, that was shown to have a derivation from a decision theoretic perspective and hence could be considered a Bayesian utility function. In the example considered in Chapter 6 the design which maximised the expected FIG produced designs with repeated points at one time leading to a posterior that was concentrated for one linear combination of the parameters but diffuse for others (see Figure 6.14). Overstall (2020) notes that designs which maximise the expected Fisher information gain can result in a singular Fisher information matrix and an ill conditioned posterior. One potential reason why the FIG has this property is that under some observed data the repeated observation times can produce a highly concentrated posterior and hence a high realised utility. The expected FIG will reward this and so can be considered a risk seeking utility.

In scenarios where nuisance parameters are present in the model the Fisher information becomes intractable (see Section 4.2.5). This means that the FIG utility function loses some appeal in terms of computational convenience as unbiased estimates of the Fisher information would have to be computed increasing the overall computation required.

A limitation of utilities based on the Fisher information matrix is that they only apply to experimental design for inference of continuous parameters. Generalisation of the utility to allow for inference for discrete parameters or model choice is not obvious. Ryan et al. (2016) gives a review of some hybrid utility functions involving $\mathcal{I}$ for joint model and parameter inference. Future work could investigate these. This thesis considered utility functions which provide information abut the model parameters and so another direction of future work could be to consider designs which optimise utilities based on the prediction of continuous future observations.

Adversarial approach

An adversarial variation on the FIG utility was considered due to the drawbacks of FIG. This considered the FIG under a parameterisation of θ to $A\theta$ for some matrix A. The elements of A were chosen to minimise the Fisher information gain whilst the design aimed to maximise the expected FIG under the adversarial transformation. The optimal design under this utility was found using SGD with simultaneous updates on the weights and the design. This returned a design that was similar to the optimal designs found under other utility functions, namely SIG and $\log|\mathcal{I}|$. In practice this appears to work well however there is less theoretical backing for the adversarial approach, both for its use as a Bayesian utility function and for convergence guarantees in the simultaneous update step required to find the optimal design. This could be the subject of future work.

8.2 Surrogate utilities

The final area considered in this thesis was using surrogate functions to alleviate some of the computational burden of finding the optimal design. This considered using a neural network to approximate the posterior allowing for fast approximations of the utility to be made. The neural network approach is a likelihood free approach to obtaining the approximate posterior only requiring realisations of (τ, θ, y) to train. Chapter 7 considered examples using the simple death model with various numbers of observations. This is a simple model with only one model parameter where realisations from the model can be made easily. It would be interesting to investigate using this surrogate more thoroughly both for this simple example and also for examples with more model parameters to see the benefit, if any, in terms of wall clock timing and efficiency of finding the optimal design.

Section 7.3 considered using these cheap utility realisations within SGD to target the design which maximised the approximate expected utility. The designs returned by this were close to the optimal design in the examples considered. Further work could involve methods to assess the quality of the surrogate as this method is highly reliant on a good approximation with an optimum close to that of the true expected utility surface.

The cheap utilities were also used within a delayed acceptance MCMC step within the Müller algorithm in Section 7.4. Initial results showed an improvement in the time taken to run the algorithm however the statistical efficiency did not seem to improve. Further work would be required to assess the time taken to converge to a good design.

Additionally a regression tree was used to identify "good" regions of the design space. Further investigation is required to assess any benefit this has in reducing the computational burden of identifying the optimal design. A direction of further work could be to incorporate this approach into existing algorithms that use a line search. For example, in ACE phase 1 rather than proposing a point which optimises a surrogate (usually a Gaussian process) of the expected marginal utility, a regression tree could be fit to realised utility draws in that margin and a proposed point drawn uniformly from the region which corresponds to the highest prediction from the tree. This could potentially reduce the overall number of utility realisations needed in the computation of the ACE algorithm.

8.3 Summary

Overall, this thesis has shown that SGD offers a scalable and fast to converge alternative to existing methods of optimal design. The computationally convenient FIG utility can

be derived from a decision theoretic perspective. Although for some examples this results in an ill conditioned posterior, adaptations have been proposed which appear to target a design similar to the optimal found under other utility functions. Finally, surrogates were considered as a way to gain a cheap-to-compute estimate of the posterior and hence a realised utility. This approach offers many avenues for future work to explore with the aim of improving the efficiency of finding the optimal design.

Bibliography

M. Abadi et al. Tensorflow: a system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

R. A. Adams. *Calculus : a complete course*. Pearson Canada, Toronto, 7th edition, 2009.

B. Amzal, F. Y. Bois, E. Parent, and C. P. Robert. Bayesian-optimal design via interacting particle systems. *Journal of the American Statistical Association*, 101:773–785, 2006.

A. C. Atkinson and R. Bailey. One hundred years of the design of experiments on and off the pages of Biometrika. *Biometrika*, 88:53–97, 2001.

A. C. Atkinson and V. Fedorov. The design of experiments for discriminating between two rival models. *Biometrika*, 62:57–70, 1975.

A. C. Atkinson, K. Chaloner, A. M. Herzberg, and J. Juritz. Optimum experimental designs for properties of a compartmental model. *Biometrics*, 49:325–337, 1993.

A. C. Atkinson, A. Donev, and R. Tobias. *Optimum experimental designs, with SAS*, volume 34. Oxford University Press, 2007.

D. Balduzzi, S. Racaniere, J. Martens, J. Foerster, K. Tuyls, and T. Graepel. The mechanics of n-player differentiable games. *arXiv e-prints*, 1802.05642, 2018.

D. L. Banks, J. M. R. Aliaga, and D. R. Insua. *Adversarial risk analysis*. CRC Press, 2015.

A. G. Baydin and B. A. Pearlmutter. Automatic differentiation of algorithms for machine learning. *arXiv e-prints*, 1404.7456, 2014.

A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood. Online learning rate adaptation with hypergradient descent. *arXiv e-prints*, 1703.04782, 2017.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

J. O. Berger. *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

R. Bergquist and L. Rinaldi. Health research based on geospatial tools: a timely approach in a changing environment. *Journal of helminthology*, 84:1–11, 2010.

J. M. Bernardo. Expected information as expected utility. *The Annals of Statistics*, 7: 686–690, 1979.

J. M. Bernardo and A. F. Smith. *Bayesian theory*. Wiley, 1994.

C. Bielza, P. Müller, and D. R. Insua. Decision analysis by augmented probability simulation. *Management Science*, 45:995–1007, 1999.

M. Binois, J. Huang, R. B. Gramacy, and M. Ludkovski. Replication or exploration? Sequential design for stochastic simulation experiments. *Technometrics*, 61:7–23, 2019.

P. L. Bonate. *Pharmacokinetic-pharmacodynamic modeling and simulation*. Springer US, Springer Science+Business Media, LLC, 2nd edition, 2011.

A. J. Booker, J. E. Dennis, P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural optimization*, 17:1–13, 1999.

L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60:223–311, 2018.

A. Boukouvalas, D. Cornford, and M. Stehlík. Optimal design for correlated processes with input-dependent noise. *Computational Statistics & Data Analysis*, 71:1088–1102, 2014.

G. E. Box. Choice of response surface design and alphabetic optimality. *Utilitas Mathematica (Canada)*, 21:11–55, 1982.

J. F. Box. RA Fisher and the design of experiments, 1922–1926. *The American Statistician*, 34:1–7, 1980.

L. M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR computational mathematics and mathematical physics*, 7:200–217, 1967.

G. W. Brier. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78:1–3, 1950.

S. Brooks, A. Gelman, G. Jones, and X.-L. Meng. *Handbook of Markov chain Monte Carlo*. CRC press, 2011.

K. Chaloner and K. Larntz. Optimal Bayesian design applied to logistic regression experiments. *Journal of statistical planning and inference*, 21:191–208, 1989.

K. Chaloner and I. Verdinelli. Bayesian experimental design: a review. *Statistical Science*, 10:273–304, 1995.

F. Chollet. *Deep learning with Python*. Manning Publications Company, 2018.

J. A. Christen and C. Fox. Markov chain Monte Carlo using an approximation. *Journal of Computational and Graphical statistics*, 14:795–810, 2005.

B. S. Clarke and A. R. Barron. Information-theoretic asymptotics of Bayes methods. *IEEE Transactions on Information Theory*, 36:453–471, 1990.

R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: a Matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

A. R. Cook, G. J. Gibson, and C. A. Gilligan. Optimal observation times in experimental epidemic processes. *Biometrics*, 64:860–868, 2008.

H. B. Curry. The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, 2:258–261, 1944.

A. C. Davison. *Statistical models*. Cambridge series on statistical and probabilistic mathematics. Cambridge University Press, Cambridge, U.K. ; New York, 2003.

A. P. Dawid. *Probability forecasting*. American Cancer Society, 2014.

A. P. Dawid and M. Musio. Theory and applications of proper scoring rules. *Metron*, 72: 169–183, 2014.

H. Dette, V. B. Melas, P. Shpilev, et al. T-optimal designs for discrimination between two polynomial models. *The Annals of Statistics*, 40:188–205, 2012.

P. Diggle. *Model-based geostatistics*. Springer series in statistics. Springer, New York, NY, 2007.

C. C. Drovandi and A. N. Pettitt. Bayesian experimental design for models with intractable likelihoods. *Biometrics*, 69:937–948, 2013.

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*, 57:320–335, 2013.

B. P. Duarte, W. K. Wong, and A. C. Atkinson. A semi-infinite programming based algorithm for determining T-optimum designs for model discrimination. *Journal of multivariate analysis*, 135:11–24, 2015.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

G. Elfving et al. Optimum allocation in linear regression theory. *The Annals of Mathematical Statistics*, 23:255–262, 1952.

J. Fellman. Gustav Elfving's contribution to the emergence of the optimal experimental design theory. *Statistical Science*, 14:197–200, 1999.

D. Firth and J. Hinde. Parameter neutral optimum design for non-linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 59:799–811, 1997.

R. A. Fisher. Theory of statistical estimation. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 22, pages 700–725. Cambridge University Press, 1925.

R. A. Fisher. *The design of experiments,*. Oliver and Boyd, Edinburgh, London, 1935.

A. Foster, M. Jankowiak, E. Bingham, P. Horsfall, Y. W. Teh, T. Rainforth, and N. Goodman. Variational Bayesian optimal experimental design. *arXiv e-prints*, 1903.05480, 2019.

P. I. Frazier. A tutorial on Bayesian optimization. *arXiv e-prints*, 1807.02811, 2018.

J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

C. Gaetan and X. Guyon. *Spatial statistics and modeling*, volume 90. Springer, 2010.

A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian data analysis*, volume 3. Chapman and Hall/CRC, 2013.

C. S. Gillespie and R. J. Boys. Efficient construction of Bayes optimal designs for stochastic process models. *Statistics and Computing*, pages 1–10, 2019.

T. Gneiting and A. E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102:359–378, 2007.

D. E. Goldberg. Genetic and evolutionary algorithms come of age. *Communications of the ACM*, 37:113–120, 1994.

A. Golightly, D. A. Henderson, and C. Sherlock. Delayed acceptance particle MCMC for exact inference in stochastic kinetic models. *Statistics and Computing*, 25:1039–1055, 2015.

I. J. Good. Rational decisions. In *Breakthroughs in statistics*, pages 365–377. Springer, 1992.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

C. M. Gotwalt, B. A. Jones, and D. M. Steinberg. Fast computation of designs robust to parameter uncertainty for nonlinear settings. *Technometrics*, 51:88–95, 2009.

R. B. Gramacy. *Surrogates: Guassian process modeling, design and optimization for the applied sciences*. Chapman Hall/CRC, Boca Raton, Florida, 2020.

A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. Siam, 2008.

M. Hainy, W. G. Müller, and H. Wagner. Likelihood-free simulation-based optimal design: an introduction. In *Topics in Statistical Simulation*, pages 271–278. Springer, 2014.

S. Harbisher, C. S. Gillespie, and D. Prangle. Bayesian optimal design using stochastic gradient optimisation and Fisher information gain. *arXiv e-prints*, 1904.05703, 2019.

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

M. Heusel et al. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6626–6637. Curran Associates, Inc., 2017.

K. Hinkelmann and A. Maman. *Design and analysis of experiments: advanced experimental design*, volume 2. Wiley Series in Probability and Statistics, 01 2005.

A. Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6:695–709, 2005.

H. Karimi, J. Nutini, and M. Schmidt. Linear convergence of gradient and proximal-gradient methods under the Polyak-łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 795–811. Springer, 2016.

N. Ketkar et al. *Deep learning with Python*, volume 1. Springer, 2017.

J. Kiefer. Optimum experimental designs. *Journal of the Royal Statistical Society: Series B (Methodological)*, 21:272–304, 1959.

D. Kingma and J. Ba. Adam: a method for stochastic optimization. *arXiv e-prints*, 1412.6980, 2014.

A. Klenke. *Probability theory: a comprehensive course*. Springer Science & Business Media, 2013.

A. Krause, R. Rajagopal, A. Gupta, and C. Guestrin. Simultaneous placement and scheduling of sensors. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, pages 181–192. IEEE Computer Society, 2009.

J. Kruschke. *Doing Bayesian data analysis: a tutorial with R, JAGS, and Stan*. Academic Press, 2014.

S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22:79–86, 1951.

H. Kushner and G. G. Yin. *Stochastic approximation and recursive algorithms and applications*, volume 35. Springer Science & Business Media, 2003.

P.-S. Laplace. Mémoire sur les approximations des formules qui sont fonctions de très grands nombres et sur leur applications aux probabilités. *Memoires de l'Academie des Sciences de Paris*, 1810.

P. M. Lee. *Bayesian statistics: an introduction*, volume 4. Wiley Publishing, 2012.

M. Leitner. *Crime modeling and mapping using geospatial technologies*, volume 8. Springer Science & Business Media, 2013.

B. Lindgren. *Statistical theory*, volume 22. CRC Press, 1993.

D. V. Lindley. *Bayesian statistics, a review*, volume 2. SIAM, 1972.

D. V. Lindley et al. On a measure of the information provided by an experiment. *The Annals of Mathematical Statistics*, 27:986–1005, 1956.

X. Liu and L.-C. Tang. A Bayesian optimal design for accelerated degradation tests. *Quality and Reliability Engineering International*, 26:863–875, 2010.

W.-Y. Loh. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1:14–23, 2011.

W.-Y. Loh. Fifty years of classification and regression trees. *International Statistical Review*, 82:329–348, 2014.

S. L. Lohr. Optimal Bayesian design of experiments for the one-way random effects model. *Biometrika*, 82:175–186, 1995.

C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: a continuous relaxation of discrete random variables. *arXiv e-prints*, 1611.00712, 2016.

L. Malagò and G. Pistone. Information geometry of the Guassian distribution in view of stochastic optimization. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII*, pages 150–162, 2015.

C. C. Margossian. A review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9:1305, 2019.

J.-M. Marin, P. Pudlo, C. P. Robert, and R. J. Ryder. Approximate Bayesian computational methods. *Statistics and Computing*, 22:1167–1180, 2012.

L. Martino, V. Elvira, and F. Louzada. Effective sample size for importance sampling based on discrepancy measures. *Signal Processing*, 131:386–401, 2017.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21: 1087–1092, 1953.

P. Minton, H. Raiffa, and R. Schlaifer. Applied statistical decision theory. *American Mathematical Monthly*, 69:72, 1962.

G. Moisen. Classification and regression trees. *Elsevier*, 1:582–588, 2008.

J. F. Monahan. *Numerical methods of statistics*. Cambridge University Press, 2011.

J. N. Morgan and J. A. Sonquist. Problems in the analysis of survey data, and a proposal. *Journal of the American statistical association*, 58:415–434, 1963.

P. Müller. Simulation-based optimal design. In *Bayesian Statistics 6: Proceedings of Sixth Valencia International Meeting*, pages 459–474. Oxford University Press, 1999.

P. Müller, B. Sansó, and M. De Iorio. Optimal Bayesian design by inhomogeneous Markov chain simulation. *Journal of the American Statistical Association*, 99:788–798, 2004.

V. Nagarajan and J. Z. Kolter. Gradient descent GAN optimization is locally stable. In *Advances in neural information processing systems*, pages 5585–5595, 2017.

Y. E. Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

C. J. Oates, J. Cockayne, D. Prangle, T. J. Sullivan, and M. Girolami. Optimality criteria for probabilistic numerical methods. *arXiv e-prints*, 1901.04326, 2019.

A. O'Hagan and J. J. Forster. *Kendall's advanced theory of statistics*, volume 28. Bayesian inference, 2004.

A. M. Overstall. A note on properties of using Fisher information gain for Bayesian design of experiments. *arXiv e-prints*, 2003.07315, 2020.

A. M. Overstall and J. McGree. Bayesian design of experiments for intractable likelihood models using coupled auxiliary models and multivariate emulation. *Bayesian Analysis*, 15:101–131, 03 2018.

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

A. M. Overstall, D. C. Woods, and M. Adamou. acebayes: An R package for Bayesian optimal design of experiments via approximate coordinate exchange. *arXiv e-prints*, 1705.08096, 2017.

A. M. Overstall, J. M. McGree, and C. C. Drovandi. An approach for finding fully Bayesian optimal designs using normal-based approximations to loss functions. *Statistics and Computing*, 28:343–358, 2018.

A. M. Overstall, D. C. Woods, and B. M. Parker. Bayesian optimal design for ordinary differential equation models with application in biological science. *Journal of the American Statistical Association*, pages 1–36, 2019.

G. Papamakarios and I. Murray. Fast ε-free inference of simulation models with Bayesian conditional density estimation. In *Advances in Neural Information Processing Systems*, pages 1028–1036, 2016.

G. Parmigiani and L. Inoue. *Decision theory: principles and approaches*, volume 812. John Wiley & Sons, 2009.

M. Parry, A. P. Dawid, S. Lauritzen, et al. Proper local scoring rules. *The Annals of Statistics*, 40:561–592, 2012.

F. Pedregosa et al. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

D. J. Price, N. G. Bean, J. V. Ross, and J. Tuke. On the efficient determination of optimal Bayesian experimental designs using ABC: A case study in optimal observation of epidemics. *Journal of Statistical Planning and Inference*, 172:1–15, 2016.

D. J. Price, N. G. Bean, J. V. Ross, and J. Tuke. An induced natural selection heuristic for finding optimal Bayesian experimental designs. *Computational Statistics & Data Analysis*, 126:112–124, 2018.

L. Pronzato and É. Walter. Robust experiment design via stochastic approximation. *Mathematical Biosciences*, 75:103–120, 1985.

N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks : the official journal of the International Neural Network Society*, 12 1:145–151, 1999.

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

E. Renshaw. *Modelling biological populations in space and time*, volume 11. Cambridge University Press, 1993.

H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, 22:400–407, 1951.

C. Robert and G. Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3:175–184, 1960.

S. Ruder. An overview of gradient descent optimization algorithms. *arXiv e-prints*, 1609.04747, 2016.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature.*, 323(6088):533–536, 1986.

E. G. Ryan, C. C. Drovandi, M. H. Thompson, and A. N. Pettitt. Towards Bayesian experimental design for nonlinear models that require a large number of sampling times. *Computational Statistics & Data Analysis*, 70:45–60, 2014.

E. G. Ryan, C. C. Drovandi, and A. N. Pettitt. Fully Bayesian experimental design for pharmacokinetic studies. *Entropy*, 17:1063–1089, 2015.

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*, 84:128–154, 2016.

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

P. D. Sampson and P. Guttorp. Nonparametric estimation of nonstationary spatial covariance structure. *Journal of the American Statistical Association*, 87:108–119, 1992.

C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

C. Sherlock, A. Golightly, and D. A. Henderson. Adaptive, delayed-acceptance MCMC for targets with expensive likelihoods. *Journal of Computational and Graphical Statistics*, 26:434–444, 2017.

N. Shukla. *Machine learning with TensorFlow*. Manning Publications Co., 2018.

K. Smith. On the standard deviations of adjusted and interpolated values of an observed polynomial function and its constants and the guidance they give towards a proper choice of the distribution of observations. *Biometrika*, 12:1–85, 1918.

R. D. Snee. Graphical analysis of process variation studies. *Journal of Quality Technology*, 15:76–88, 1983.

J. R. Stroud, P. Müller, and G. L. Rosner. Optimal sampling times in population pharmacokinetic studies. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 50:345–359, 2001.

N.-Z. Sun and A. Sun. *Model calibration and parameter estimation: for environmental and water resource systems*. Springer, 2015.

T. D. Team. Theano: a Python framework for fast computation of mathematical expressions. *arXiv e-prints*, 1605.02688, 2016.

T. N. Tozer and M. Rowland. *Introduction to pharmacokinetics and pharmacodynamics: the quantitative basis of drug therapy*. Lippincott Williams & Wilkins, 2006.

C. Tsallis. Possible generalization of Boltzmann-Gibbs statistics. *Journal of statistical physics*, 52:479–487, 1988.

P. J. Van Laarhoven and E. H. Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer, 1987.

P. Walker. *Examples and theorems in analysis*. Springer Science & Business Media, 2012.

S. G. Walker. Bayesian information in an experiment and the Fisher information distance. *Statistics & Probability Letters*, 112:5–9, 2016.

Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. De Freitas. Bayesian optimization in high dimensions via random embeddings. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

M. D. Ward and K. S. Gleditsch. *Spatial regression models*, volume 155. Sage Publications, 2018.

L. J. Wolfson, J. B. Kadane, and M. J. Small. Expected utility as a policy-making tool: an environmental health example. *Statistics Textbooks and Monographs*, 151:261–278, 1996.

X. Wu, R. Ward, and L. Bottou. WNGrad: learn the learning rate in gradient descent. *arXiv e-prints*, 1803.02865, 2018.

H. P. Wynn. Jack Kiefer's contributions to experimental design. *The Annals of Statistics*, 12:416–423, 1984.

F. Yates. Complex experiments. *Supplement to the Journal of the Royal Statistical Society*, 2:181–247, 1935.

F. Yates. Sir Ronald Fisher and the design of experiments. *Biometrics*, 20:307–321, 1964.

N. Ye, Z. Zhu, and R. K. Mantiuk. Langevin dynamics with continuous tempering for training deep neural networks. *arXiv e-prints*, 1703.04379, 2017.

G. Zaccone, M. R. Karim, and A. Menshawy. *Deep learning with TensorFlow*. Packt Publishing Ltd, 2017.

M. D. Zeiler. ADADELTA: an adaptive learning rate method. *arXiv e-prints*, 1212.5701, 2012.