Innovative CVX-based Algorithms for Optimal Design Problems on Discretized Regions

by

Hanan Abousaleh
BSc, University of Victoria, 2021

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Mathematics and Statistics

Innovative CVX-based Algorithms for Optimal Design Problems
on Discretized Regions

by

Hanan Abousaleh
BSc, University of Victoria, 2021

Supervisory Committee

Dr. Julie Zhou, Supervisor
(Department of Mathematics and Statistics)

Dr. Michelle Miranda, Committee Member
(Department of Mathematics and Statistics)

ABSTRACT

We focus on a class of optimization problems known as optimal design problems, where the goal is to select design points optimally with respect to some criterion of interest. For regression models, the optimality criterion is based on the statistical model itself and is often a function of the information matrix. We solve A-, D-, and EI-optimal design problems in this thesis. The CVX program in MATLAB is a modelling tool and solver for convex optimization problems. As with other numerical methods in the literature, formulating an optimal design problem in a CVX-compatible way requires a discrete design space. We develop a CVX-based algorithm to solve optimal design problems on large and irregular discrete spaces for multiple regression models. The algorithm uses innovative rules to add several design points at each iteration, and clusters nearby points together at the end of iteration. Furthermore, we provide useful guidelines for discretizing irregular regions. These are based on derived theoretical properties which relate optimal designs on continuous and discrete design spaces. Several numerical examples and their MATLAB codes are presented for A-, D-, and EI-optimal designs for both linear and generalized linear models. The optimal designs found via the CVX solver are better than those presented in the literature. In addition, our guidelines to discretizing design spaces improve the efficiency of optimal designs, especially over irregular regions. We find that our iterative procedure overcomes the bottlenecks of typical sequential and multiplicative algorithms.

Table of Contents

ACKNOWLEDGEMENTS

First and foremost, I would like to praise Allah the Almighty, the Most Gracious and the Most Merciful, for His blessings given to me during my study and in completing this thesis. May Allah's blessings go to His final Prophet Muhammad (peace be upon him), his family, and his companions.

I wish to convey my appreciation for my supervisor, Dr. Julie Zhou. This thesis would not be possible without your generosity, encouragement, and expertise. For the opportunities that I was able to pursue under your leadership, for the countless times that your advice brought me clarity, and for your financial support of this research with your NSERC grants as well, I am forever grateful.

I would also like to sincerely thank my supervisory committee member, Dr. Michelle Miranda, and my external examiner, Dr. Nilanjana Roy. Your time and consideration in examining this thesis is invaluable.

Last but not least, I take this opportunity to express my profound gratitude to my father, my late mother, and my siblings. Thank you for your care and your never-ending support of my pursuits, including this thesis from start to finish. I thank Allah for the privilege of belonging to such an amazing family.

Chapter 1

Introduction

An experiment is a procedure with unknown outcomes that is undertaken to answer some research question(s). Many research questions are about identifying the relationships between two or more variables. For example, a radiologist may be interested in the effect of radiation dosage on tumor shrinkage. A clinical trial to investigate this may treat cancer patients with differing dosage levels and measure their responses. Maximizing the efficiency of such a dose-response clinical trial is crucial for both ethical and cost reasons. Optimal Design of Experiments (DoE) is the branch of statistics concerned with finding the best designs with respect to a given criterion; which may be based on a statistical model of interest. In this thesis, we will present and apply algorithms to compute approximate, model-based optimal designs. We focus on approximate designs (instead of exact designs) because they are easier to optimize and there exists a unified theory about them. Well-known design monographs include Atkinson et al. (2007), Berger and Wong (2009), and Fedorov (1972).

Chapter 1 is organized as follows. Section 1.1 presents the regression models. Section 1.2 introduces the optimization problem. Section 1.3 reviews the literature on optimal designs. Finally, Section 1.4 states the research problems of this thesis and its main contributions.

1.1 The regression models

For model-based designs, we assume that a regression model is used to describe the relationship between the variables of interest. In this section, we present the notation for multiple regression modelling, which relates the outcome variable to more than one explanatory variable.

Suppose we are interested in modelling some response variable, y, in terms of p independent predictor variables denoted by the vector $\mathbf{x} = (x_1, x_2, \ldots, x_p)^\top$, where $\mathbf{x} \in \Omega \subset \mathbb{R}^p$. The bounded set Ω is called the *design region* or *design space*. It consists of all the feasible values of $\mathbf{x}$ (design points), subject to experimental constraints. We consider two categories of regression models: the linear model (LM) and the generalized linear model (GLM). Assume the sample size is n.

1.1.1 Linear models

The LM can be written as

$$y_i = \mathbf{z}(\mathbf{x}_i)^\top \boldsymbol{\beta} + \epsilon_i, \qquad i = 1, \ldots, n, \tag{1.1}$$

where $\boldsymbol{\beta} \in \mathbb{R}^q$ is the unknown parameter vector, $\mathbf{z}(\mathbf{x}) = (z_1(\mathbf{x}), z_2(\mathbf{x}), \ldots, z_q(\mathbf{x}))^\top$ is the vector of known basis functions, and the errors ϵ_i are assumed uncorrelated, with mean 0 and variance σ^2. The LM is called "linear" because of its linearity in the parameters, $\boldsymbol{\beta}$, not the predictors, $\mathbf{x}$. Basis functions are used to allow the modelling of nonlinearity in the data while keeping linearity in the parameters. In other words, the LM relates y to the linear combination of q nonlinear functions of $\mathbf{x}$. The term $\mathbf{z}(\mathbf{x})^\top \boldsymbol{\beta}$ is called the *linear predictor* and is denoted as η. For brevity, $\mathbf{z}(\mathbf{x})$ will be referred to as the *basis vector* for the remainder of this thesis. Under these model assumptions we find

the conditional expectation and variance of y given $\mathbf{x}_i$ as $\mathbb{E}(y|\mathbf{x}_i) = \eta_i = \mathbf{z}(\mathbf{x}_i)^\top \boldsymbol{\beta}$ and $\text{Var}(y|\mathbf{x}_i) = \sigma^2$.

When the range of y is restricted (e.g., binary or count data) or the variance of y depends on the mean, GLMs may be used to address these challenges.

1.1.2 Generalized linear models

We provide a quick overview here and refer readers to Fedorov and Leonov (2014) and Stufken and Yang (2012) for more details. Let us assume that the observations of the response, y_i, are independent and follow some distribution from the exponential family with mean μ_i, for all $i = 1, \ldots, n$. The i-subscript on μ indicates that the mean is a function of the data; specifically, it is a function of the linear predictor $\eta_i = \mathbf{z}(\mathbf{x}_i)^\top \boldsymbol{\beta}$. Some examples of the exponential family are the normal, binomial, Poisson, and gamma distributions.

GLMs cannot be expressed in an additive form like (1.1) because the separation of the random (ϵ_i) and systemic (η_i) components is not possible for many distributions. Instead, a GLM is specified by

$$\mathbb{E}(y|\mathbf{x}_i) = \mu_i := h^{-1}(\eta_i),$$
$$\text{and } \text{Var}(y|\mathbf{x}_i) = \sigma_i^2 := \nu(\mu_i),$$

for all $i = 1, \ldots, n$. Function $h : \text{range}(y) \to (-\infty, \infty)$ is called the link function because it relates the mean response to the linear predictor. The constant variance assumption is not applicable for GLMs as $\text{Var}(y|\mathbf{x}_i)$ is a function of the mean response, denoted by $\nu(\cdot)$.

Bernoulli distribution: suppose $y_i \sim \text{Bernoulli}(\pi_i)$, where $\mathbb{E}(y_i) = \pi_i \in (0, 1)$

and $\text{Var}(y_i) = \pi_i(1-\pi_i)$. A popular choice of $h(\pi_i)$ is the logit function, such that

$$\begin{aligned} \eta_i = \mathbf{z}(\mathbf{x}_i)^\top \boldsymbol{\beta} = h(\pi_i) \quad &:= \ln\left(\frac{\pi_i}{1-\pi_i}\right) \\ \iff \pi_i = \mathbb{E}(y_i) \quad &= h^{-1}(\eta_i) := \frac{e^{\eta_i}}{1+e^{\eta_i}} \end{aligned} \tag{1.2}$$

for all $i = 1, \ldots, n$. Then we can also find the variance of the response as

$$\text{Var}(y_i) = \pi_i(1-\pi_i) = \frac{e^{\eta_i}}{(1+e^{\eta_i})^2}. \tag{1.3}$$

Normal distribution: suppose $y_i \sim \text{N}(\mu_i, \sigma^2)$. Since the mean response is already unrestricted, the link function is the identity function. That is, $\eta_i = h(\mu_i) = \mu_i$. Furthermore, the variance of the response is constant, so $\nu(\mu_i) = \sigma^2$. Hence, the LM in (1.1) is actually a special case of the GLM if we assume that the error terms follow a $\text{N}(0, \sigma^2)$ distribution.

1.1.3 Estimating the model parameters

For a given LM or GLM, we ultimately wish to estimate $\boldsymbol{\beta}^*$, the true and unknown values of $\boldsymbol{\beta}$, using some estimator. Note that an *estimator* of $\boldsymbol{\beta}$ is a function of the sample, while an *estimate* of $\boldsymbol{\beta}^*$ is the value of an estimator calculated from the sample. Clearly, estimates of $\boldsymbol{\beta}^*$ can only be obtained after the data are collected. In DoE contexts, the experiment has not yet taken place because we are designing it; therefore, we need only focus on estimators. Let us denote estimators of the parameters by $\hat{\boldsymbol{\beta}}$ and their covariance matrices by $\text{Cov}(\hat{\boldsymbol{\beta}})$.

For GLMs, the maximum likelihood estimator (MLE) is commonly used to estimate $\boldsymbol{\beta}$. MLEs do not typically have closed-form solutions unless in the normal linear case. Thus, they may be found through Fisher's Method of Scoring, an iterative algorithm

based on the Newton-Raphson method. The asymptotic properties of MLEs are used to estimate $\text{Cov}(\hat{\boldsymbol{\beta}})$. A popular estimator for LMs is the least squares estimator (LSE). In fact, it is the best linear unbiased estimator and is equal to the MLE in the normal linear case. More on the forms of LSEs and MLEs can be found in any regression textbook. The optimal design problem of this thesis pertains to selecting the design points from Ω that maximize the efficiency of $\hat{\boldsymbol{\beta}}$, i.e. minimize $\text{Cov}(\hat{\boldsymbol{\beta}})$ according to some criterion.

1.2 Experimental designs

We describe the optimization problem on the discrete design space and revisit the continuous space in Chapter 3. Consider a discrete subset of N points selected from Ω which cover it well, denoted by $\Omega_N = \{\mathbf{u}_1, \ldots, \mathbf{u}_N\} \subset \Omega$. An approximate design on Ω_N is specified as

$$\xi_N \;=\; \begin{pmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \ldots & \mathbf{u}_N \\ w_1 & w_2 & \ldots & w_N \end{pmatrix},$$

where $\mathbf{u}_1, \ldots, \mathbf{u}_N$ are possible design points at which we may measure y, and the weights $\boldsymbol{w} = (w_1, \ldots, w_N)^\top$ are non-negative real numbers which satisfy $\sum_{j=1}^{N} w_j = 1$. A point $\mathbf{u}_j$ with a strictly positive weight $w_j > 0$ is called a support point of ξ_N. We say the design ξ_N is approximate because each weight w_j *approximates the proportion* of trials held under $\mathbf{u}_j$, for $j = 1, \ldots, N$. In other words, the weights signify the importance or contribution of each point to the experiment. Multiplying $\boldsymbol{w}$ by a chosen sample size n and then rounding (subject to $nw_1 + \cdots + nw_N = n$) will recover the number of replicates at each point. See Pukelsheim and Rieder (1992) for various rounding rules between exact and approximate designs. For brevity, we shall refer to approximate optimal designs as optimal designs in this thesis. Writing ξ instead of ξ_N denotes designs on the continuous space Ω instead of the discrete space, Ω_N.

Because the points at which we observe y influence the efficiency of the MLE, we seek the optimal design points such that $\Phi(\mathrm{Cov}(\hat{\boldsymbol{\beta}}))$ is minimized, for some scalar objective function Φ. Letting $\eta_j^* = \mathbf{z}(\mathbf{u}_j)^\top \boldsymbol{\beta}^*$ for all $j = 1, \ldots, N$, we have

$$\mathrm{Cov}(\hat{\boldsymbol{\beta}}) = \frac{1}{n}\mathbf{I}^{-1}(\xi_N, \boldsymbol{\beta}^*), \tag{1.4}$$

where the $q \times q$ matrix $\mathbf{I}(\xi_N, \boldsymbol{\beta}^*)$ is the empirical Fisher information matrix defined by

$$\mathbf{I}(\xi_N, \boldsymbol{\beta}^*) = \sum_{j=1}^{N} w_j \mathbf{z}(\mathbf{u}_j)\, \gamma_j\, \mathbf{z}(\mathbf{u}_j)^\top, \tag{1.5}$$

$$\text{and } \gamma_j = \gamma(\mathbf{u}_j) = \left[\frac{1}{\sigma_j} \cdot \frac{\partial h^{-1}(\eta)}{\partial \eta}\right]^2 \Bigg|_{\eta=\eta_j^*}. \tag{1.6}$$

The term σ_j is the variance of y under the design point $\mathbf{x} = \mathbf{u}_j$. The naming of $\mathbf{I}(\xi_N, \boldsymbol{\beta}^*)$ comes from its inverse-proportional relationship to $\mathrm{Cov}(\hat{\boldsymbol{\beta}})$. Less uncertainty, i.e. variation, in $\hat{\boldsymbol{\beta}}$ makes the estimator more informative (Fedorov & Leonov, 2014). The points with zero weight clearly make no contribution to (1.5).

We can see that $\mathbf{I}(\xi_N, \boldsymbol{\beta}^*)$, and therefore $\mathrm{Cov}(\hat{\boldsymbol{\beta}})$, may actually depend on the parameters via the γ_j term. For such cases, $\boldsymbol{\beta}^*$ needs to be estimated using results from prior studies or seeking expert opinion, for example. Regardless, the dependency limits us to finding locally optimal designs. The exception is the normal linear case where $\gamma_j = 1/\sigma^2$ is a constant; a convenient property which allows us to find globally optimal designs for LMs. For brevity, we will refer to the empirical Fisher information matrix in (1.5) as the information (matrix) for the remainder of this thesis. In addition, $\mathbf{I}(\xi_N)$ is written instead of $\mathbf{I}(\xi_N, \boldsymbol{\beta}^*)$ when there is no confusion because it is understood that $\boldsymbol{\beta}^*$ will be estimated where needed.

If $\mathbf{u}_1, \ldots, \mathbf{u}_N$ are fixed in Ω_N, then both ξ_N and $\mathbf{I}(\xi_N)$ are completely determined by the weight vector, $\boldsymbol{w}$. The optimal design problem on Ω_N is then defined as the

following constrained optimization problem,

$$\begin{cases} \min\limits_{\boldsymbol{w}} \;\; \Phi(\xi_N) \\ \text{s.t.} \quad \sum_{j=1}^{N} w_j = 1 \;\; \text{and} \;\; w_j \geq 0, \quad \forall j = 1, \ldots, N. \end{cases} \tag{1.7}$$

The notation $\Phi(\xi_N)$ is shorthand for $\Phi(\mathbf{I}^{-1}(\xi_N))$, which emphasizes that objective function is truly a function of $\text{Cov}(\hat{\boldsymbol{\beta}})$. A design which is a solution to problem (1.7) is called *Φ-optimal* on Ω_N and is denoted ξ_N^* with corresponding weight vector $\boldsymbol{w}^*$. Additionally, when $\Phi(\xi_N)$ is a convex function of $\boldsymbol{w}$, Problem (1.7) is a convex optimization problem (Kiefer, 1974). Several commonly used optimality criteria, including A-, c-, D-, I- and L-optimality, have been shown to be convex functions of $\boldsymbol{w}$; see Bose and Mukerjee (2015), Boyd and Vandenberghe (2004), and Wong and Zhou (2019) for details. Furthermore, the Φ-optimal design on the discrete space Ω_N satisfies the following optimality condition:

$$\begin{cases} d_\Phi(\mathbf{u}_j, \xi_N^*) = 0, & \text{for all } \mathbf{u}_j \in \Omega_N \;\; \text{with} \;\; w_j > 0, \\ d_\Phi(\mathbf{u}_j, \xi_N^*) < 0, & \text{otherwise.} \end{cases} \tag{1.8}$$

The function d_Φ depends on the optimality criterion, and the equality to zero only holds at the support points of ξ_N^*. Condition (1.8) is based on the general equivalence theory following Kiefer and Wolfowitz (1959) and Kiefer (1974).

For example, the D-optimality criterion minimizes the determinant, or some convex function of the determinant, of $\text{Cov}(\hat{\boldsymbol{\beta}})$. For this thesis, we define the D-optimality criterion on Ω_N as

$$\Phi_D(\xi_N) = -\{\det[\mathbf{I}(\xi_N)]\}^{1/q}, \tag{1.9}$$

where q is the number of estimated parameters. From Atkinson and Woods (2015),

equivalence theory for D-optimality of GLMs has

$$d_D(\mathbf{u}_j, \xi_N^*) = \gamma_j \mathbf{z}(\mathbf{u}_j)^\top \mathbf{I}^{-1}(\xi_N^*)\mathbf{z}(\mathbf{u}_j) - q. \tag{1.10}$$

1.3 Literature review

Kiefer and Wolfowitz (1959) and Kiefer (1974) established some of the early fundamentals of experimental design. This includes the equivalence theories of various optimality criteria and the introduction of convex optimization techniques for DoE. It is usually difficult to find optimal designs analytically. As such, several numerical methods have been developed for finding optimal designs.

Fedorov (1972) proposed a sequential point-exchange algorithm to solve optimal designs on discrete spaces. Design points are added and deleted by evaluating all possible exchanges of pairings and those that result in the largest improvement in criterion value are selected. This algorithm depends on the quality of the initial design, and it can be computationally expensive to check many pairs of points for exchange. Cook and Nachtsheim (1980) suggested improvements to speed up convergence. Mitchell (1974) was the first to separate the addition and deletion rules for sequential algorithms with their proposed "DETMAX" algorithm. Then, Silvey et al. (1978) introduced the multiplicative algorithm. With this method, the weights for the entire candidate set are simultaneously updated. When the candidate pool of points is large, the convergence is slow and may also lead to a large number of support points. Applications of multiplicative and sequential algorithms can be found in Bose and Mukerjee (2015), Duan et al. (2019, 2022), Goos et al. (2016), and Li and Deng (2021)

More recently, mathematical programming techniques have been used to find optimal designs. For example, see the cocktail algorithm (Yu, 2011), semidefinite program-

ming methods (De Castro et al., 2019; Papp, 2012; Ye et al., 2017), descent-type algorithms like Newton-Rahpson method (Molchanov & Zuyev, 2002; Yang et al., 2013), simulated annealing method (Haines, 1987; Rempel & Zhou, 2014), particle swarm optimization (Chen et al., 2015), and various algorithms based on the CVX solver (Abousaleh & Zhou, 2023; Gao & Zhou, 2017; Rankin & Zhou, 2023; Wong & Zhou, 2019; Yeh & Zhou, 2021).

1.4 Research problems and contributions

There are still challenging optimal design problems that need to be solved, including problems with irregularly-shaped design spaces or large candidate sets. We aim to demonstrate that CVX is an excellent tool for DoE for these cases in particular, and can lead to better optimal designs than those in the literature. Here are the main contributions of this thesis. We have developed an innovative algorithm for computing optimal designs on discrete design spaces, provided guidelines to discretize continuous design spaces, derived several theoretical results for the relationship between optimal designs on discrete and continuous design spaces, and developed MATLAB code for various applications.

This thesis is organized as follows. In Chapter 2, we will study EI-optimal designs and use the CVX solver to solve various EI-optimal design problems. In Chapter 3, we will prove theoretical results about the relationship between the continuous design space and its discretized design space. Insights are provided on discretizing continuous design regions. In Chapter 4, we will present the iterative CVX-based algorithm along with some numerical examples. Chapter 5 contains the summary and conclusion remarks. The Appendix contains the MATLAB code developed in this thesis.

Chapter 2

Constructing EI-optimal designs via the CVX solver

Prediction-based optimality criteria for regression models can be beneficial in experimental design. The I-optimality criterion, for example, seeks to minimize the average prediction variance over a design region Ω. This criterion assumes that the entire design region is of equal importance. If we want to take a somewhat weighted average of the prediction variance, we will want to use the Elastic I (EI) criterion defined by Li and Deng (2021). This chapter pertains to finding EI-optimal designs for various models. In Section 2.1, we define the EI-optimality criterion and prove its scale-invariance and reflection-symmetric properties for LMs. Section 2.2 presents a simple CVX-based algorithm for the computation of EI-optimal designs. An algorithm for Monte-Carlo numerical integration is also provided, which can be useful in the EI-optimal design problem. Numerical examples are given in Section 2.3. The results are compared with the designs found through other sequential-multiplicative or multiplicative algorithms. Finally, a brief summary concluding the chapter is given in Section 2.4.

2.1 EI-optimality definition and properties

Li and Deng (2021) define the EI-optimality criterion for GLMs on Ω:

$$\Phi_{EI}(\xi, F) = \mathrm{Tr}\big\{\mathbf{A}\mathbf{I}^{-1}(\xi)\big\}, \tag{2.1}$$

$$\text{where} \qquad \mathbf{A} = \int_S \mathbf{z}(\mathbf{x})\mathbf{z}(\mathbf{x})^\top \left(\frac{\partial h^{-1}}{\partial \eta}\right)^2 dF(\mathbf{x}). \tag{2.2}$$

In (2.2), F denotes a cumulative distribution function (CDF) that is supported on $S \subseteq \Omega$. We assume that F belongs to a multivariate location-scale family. By dF, we denote its probability density function (PDF) so that $dF(\mathbf{x}) = f(\mathbf{x})\, d\mathbf{x}$.

When F is the uniform distribution function, we denote it F_{UNIF}, and we have $dF_{\mathsf{UNIF}}(\mathbf{x}) = f_{\mathsf{UNIF}}(\mathbf{x})\, d\mathbf{x} = \frac{1}{|S|}\, d\mathbf{x}$ where $|S|$ is the volume of the support region S. Furthermore, when $S = \Omega$ and $F = F_{\mathsf{UNIF}}$, (2.1) becomes the familiar I-optimality criterion with

$$\mathbf{A} = \frac{1}{|\Omega|} \int_\Omega \mathbf{z}(\mathbf{x})\mathbf{z}(\mathbf{x})^\top \left(\frac{\partial h^{-1}}{\partial \eta}\right)^2 d\mathbf{x}\,.$$

Of course, the $\left({}^{\partial h^{-1}}/_{\partial \eta}\right)^2$ term in (2.2) disappears for LMs because the link function is the identity function.

Note that the I- and EI-optimality criteria minimize the average variance of the predicted response. The form of (2.1) is very similar to the A-optimality criterion, which is not prediction-based but rather a descriptor of the confidence region. To minimize the average estimated variance of $\hat{\boldsymbol{\beta}}$, we need only let $\mathbf{A} = I_q$, where I_q is the identity matrix of size q. Thus, the definition of the A-optimality criterion is

$$\Phi_A(\xi) = \mathrm{Tr}\big\{\mathbf{I}^{-1}(\xi)\big\}. \tag{2.3}$$

Then, the equivalence theory for this general form of optimality criteria (Fedorov &

Leonov, 2014) gives the optimality condition function as follows:

$$d(\mathbf{x}, \xi^*) = \mathrm{Tr}\Big\{ \mathbf{A}\mathbf{I}^{-1}(\xi^*)\Big[\mathbf{z}(\mathbf{x})\gamma(\mathbf{x})\mathbf{z}(\mathbf{x})^\top\Big]\mathbf{I}^{-1}(\xi^*) - \mathbf{A}\mathbf{I}^{-1}(\xi^*)\Big\}, \tag{2.4}$$

for all $\mathbf{x} \in \Omega$. Subscripts A, I, or EI are added to distinguish the criterion of interest, and $d_A(\mathbf{x}, \xi^*)$ corresponding to the A-optimality criterion in (2.3) has $\mathbf{A} = I_q$.

The EI criterion can be useful if there are sub-regions of Ω which are more important to the modelling process than others. These regions are prioritized by both the form of F as well as its support region S. The EI criterion has some special properties for LMs. The remainder of this section is dedicated to listing and proving two of them in particular.

2.1.1 Scale-invariance for LMs

Scale-invariance, sometimes just called invariance, is a useful property which can simplify the construction of optimal designs. For example, rather than considering $\Omega = [0, a]$, one may instead find ξ^* over a standardized region like $[0, 1]$ and then scale out the optimal design to $[0, a]$ for any $a > 0$. D-optimal designs for many LMs are well-known to be scale-invariant (Yeh & Zhou, 2021); A-optimal designs, however, are usually not. We derive a result for the scale-invariance of EI-optimal designs below.

Let V be a non-singular $p \times p$ diagonal matrix (so that V^{-1} exists). For a design space Ω, we define the *V-scaled design space* as $\Omega^V := \{V\mathbf{x} \mid \mathbf{x} \in \Omega\}$. Furthermore, we denote by $F^V(\boldsymbol{x})$ the V-scaled distribution of $F(\mathbf{x})$. By definition of location-scale families, $F^V(\boldsymbol{x}) := F(V^{-1}\boldsymbol{x})$ for any $\boldsymbol{x} \in \Omega^V$ (Rinne, 2010). We say that a Φ-optimal design ξ^* on Ω is *scale-invariant* if the Φ-optimal design on Ω^V, denoted ξ^*_V, can be obtained from the scale transformation on ξ^* with scale matrix V.

Theorem 2.1. *Consider a linear model with basis vector* $\mathbf{z}(\mathbf{x})$*. Let* $F(\mathbf{x})$ *be a location-scale CDF on* Ω*. For a scale transformation matrix* V*, if there exists a non-singular, diagonal constant matrix* Q *such that* $\mathbf{z}(V\mathbf{x}) = Q\mathbf{z}(\mathbf{x})$ *for all* $\mathbf{x} \in \Omega$*, then the EI-optimal design* ξ^* *is scale-invariant.*

Proof of Theorem 2.1: Let ξ_V^* denote the EI-optimal design on Ω^V which minimizes $\Phi_{EI}(\xi_V, F^V)$ over ξ_V, where $\Phi_{EI}(\xi_V, F^V) = \mathrm{Tr}\{\mathbf{A}_V\mathbf{I}^{-1}(\xi_V)\}$. Note that all elements $\boldsymbol{x} \in \Omega^V$ can be written as $\boldsymbol{x} = V\mathbf{x}$ for some $\mathbf{x} \in \Omega$. Assume the premise is true so that $\mathbf{z}(V\mathbf{x}) = Q\mathbf{z}(\mathbf{x})$ holds for all $\mathbf{x} \in \Omega$. Thus, we find that

$$\begin{aligned}
\mathbf{A}_V &= \int_{\Omega^V} \mathbf{z}(\boldsymbol{x})\mathbf{z}(\boldsymbol{x})^\top \, dF^V(\boldsymbol{x}) \\
&= \int_{\Omega} Q\mathbf{z}(\mathbf{x})\mathbf{z}(\mathbf{x})^\top Q^\top \, dF(\mathbf{x}) \cdot |\det(V)| \qquad (2.5) \\
&= c\,Q\left[\int_{\Omega} \mathbf{z}(\mathbf{x})\mathbf{z}(\mathbf{x})^\top \, dF(\mathbf{x})\right] Q^\top = c\,Q\mathbf{A}Q,
\end{aligned}$$

where $c = |\det(V)|$ and $Q = Q^\top$ due to its diagonal structure. Equation (2.5) follows from the multivariate transformation method of CDFs. Furthermore,

$$\begin{aligned}
\mathbf{I}(\xi_V) &= \sum_{i=1}^{N} w_i \mathbf{z}(\boldsymbol{x}_i)\mathbf{z}(\boldsymbol{x}_i)^\top \\
&= \sum_{i=1}^{N} w_i Q\mathbf{z}(\mathbf{x}_i)\mathbf{z}(\mathbf{x}_i)^\top Q^\top = Q\mathbf{I}(\xi)Q.
\end{aligned}$$

Therefore, we find that

$$\begin{aligned}
\Phi_{EI}(\xi_V, F^V) &= \mathrm{Tr}\big\{\mathbf{A}_V\mathbf{I}^{-1}(\xi_V)\big\} \\
&= \mathrm{Tr}\big\{c\,Q\mathbf{A}Q \cdot Q^{-1}\mathbf{I}^{-1}(\xi)Q^{-1}\big\} \\
&= c\ \mathrm{Tr}\big\{\mathbf{A}\mathbf{I}^{-1}(\xi)Q^{-1}Q\big\} \\
&= c\ \mathrm{Tr}\big\{\mathbf{A}\mathbf{I}^{-1}(\xi)\big\} = c\,\Phi_{EI}(\xi, F).
\end{aligned}$$

Thus minimizing $\Phi_{EI}(\xi_V, F^V)$ over ξ_V is equivalent to minimizing $\Phi_{EI}(\xi, F)$ over ξ. This implies that one can find ξ^* and apply a scale transformation with V to get ξ_V^*. □

Proving this property is more challenging when GLMs are considered. We refer the reader to Idais and Schwabe (2021), which presents some general results for the equivariance of D- and EI-optimal designs for GLMs. Equivariance is a weaker form of scale-invariance.

2.1.2 Reflection-symmetry for LMs

For our discussion below, symmetry is with respect to the origin. Let $\ell \in \{1, 2, \ldots, p\}$ be some coordinate axis in p-dimensional space. Then, T_ℓ denotes a $p \times p$ diagonal matrix with diagonal entries $T_\ell[\ell, \ell] = -1$ and $T_\ell[j, j] = 1$ for all $j \neq \ell$. A design region $\Omega_N = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N\}$ is *(reflection-) symmetric* in the x_ℓ direction if the transformation on Ω_N induced by T_ℓ is self-mapping, i.e. $T_\ell\Omega_N := \{T_\ell\mathbf{u}_1, T_\ell\mathbf{u}_2, \ldots, T_\ell\mathbf{u}_N\} = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N\}$.

Let $\widetilde{\xi}_\ell$ denote the reflection of a design ξ in the x_ℓ direction,

$$\widetilde{\xi}_\ell \;=\; \begin{pmatrix} T_\ell\mathbf{u}_1 & T_\ell\mathbf{u}_2 & \ldots & T_\ell\mathbf{u}_N \\ w_1 & w_2 & \ldots & w_N \end{pmatrix} \;=\; \begin{pmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \ldots & \mathbf{u}_N \\ \widetilde{w}_1 & \widetilde{w}_2 & \ldots & \widetilde{w}_N \end{pmatrix}.$$

If $\boldsymbol{w} = \widetilde{\boldsymbol{w}}$, then ξ is symmetric in x_ℓ. This property may be examined for any $\ell = 1, \ldots, p$. For example, consider the following design of $p = 2$ variables on $[-1, 1]^2$:

$$\xi \;=\; \begin{pmatrix} (-1,-1)^\top & (-1,1)^\top & (-0.5,0)^\top & (0,0)^\top & (1,-1)^\top & (1,1)^\top \\ 0.15 & 0.15 & 0.2 & 0.2 & 0.15 & 0.15 \end{pmatrix}$$

This design is not symmetric in x_1, because $T_1\mathbf{u}_3 = \left[\begin{smallmatrix}-1 & 0\\ 0 & 1\end{smallmatrix}\right]\left[\begin{smallmatrix}-0.5\\ 0\end{smallmatrix}\right] = (0.5, 0)^\top$ is not a support point of ξ. It is, however, symmetric in x_2 because T_2 is self-mapping and the reflected weight vector is unchanged.

Proposition 2.2. *$\Phi_{EI}(\xi, F)$ is a convex function of $\boldsymbol{w}$.*

For proof of Proposition 2.2, see Lemmas 2 and 4 in Li and Deng (2021).

Theorem 2.3. *Consider a linear regression model of the form* (1.1). *The sufficient conditions that there exists a symmetric EI-optimal design ξ^* in x_ℓ are:*

(i) *Ω_N is symmetric in x_ℓ,*

(ii) *$F(\mathbf{x})$ is symmetric with respect to x_ℓ,*

(iii) *there exists a $q \times q$ diagonal constant matrix Q such that $\mathbf{z}(T_\ell\mathbf{u}_i) = Q\mathbf{z}(\mathbf{u}_i)$ for all $i = 1, \ldots, N$, and $Q^\top Q = Q^2 = I_q$.*

Proof of Theorem 2.3: For a design ξ with weights $\boldsymbol{w}$, let $\widetilde{\xi}_\ell$ and $\widetilde{\boldsymbol{w}}$ denote the reflected design and weights, respectively. We aim to show that $\Phi_{EI}(\xi, F) = \Phi_{EI}(\widetilde{\xi}_\ell, F)$ for all designs ξ when the sufficient conditions are satisfied. Assume all three conditions hold.

The symmetry of $F(\mathbf{x})$ in terms of x_ℓ means that $F(\mathbf{x}) = F(T_\ell\mathbf{x})$. Using conditions (i)-(iii), we can write $\mathbf{A}$ as follows:

$$\begin{aligned}
\mathbf{A} &= \int_\Omega \mathbf{z}(\mathbf{x})\mathbf{z}(\mathbf{x})^\top \, dF(\mathbf{x}) \\
&= \int_\Omega Q^\top Q\mathbf{z}(\mathbf{x})\mathbf{z}(\mathbf{x})^\top Q^\top Q \, dF(T_\ell\mathbf{x}) \\
&= Q^\top \left[\int_\Omega Q\mathbf{z}(\mathbf{x})\mathbf{z}(\mathbf{x})^\top Q^\top \, dF(T_\ell\mathbf{x})\right] Q \\
&= Q^\top \left[\int_\Omega \mathbf{z}(T_\ell\mathbf{x})\mathbf{z}(T_\ell\mathbf{x})^\top \, dF(T_\ell\mathbf{x})\right] Q \\
&= Q^\top \mathbf{A} Q. \qquad (2.6)
\end{aligned}$$

Then, under (iii), we find that

$$\begin{aligned}
\mathbf{I}(\widetilde{\xi}_\ell) &= \sum_{i=1}^{N} w_i \mathbf{z}(T_\ell \mathbf{u}_i)\mathbf{z}(T_\ell \mathbf{u}_i)^\top \\
&= \sum_{i=1}^{N} w_i Q\mathbf{z}(\mathbf{u}_i)\mathbf{z}(\mathbf{u}_i)^\top Q^\top \\
&= Q\left[\sum_{i=1}^{N} w_i \mathbf{z}(\mathbf{u}_i)\mathbf{z}(\mathbf{u}_i)^\top\right] Q^\top \\
&= Q\mathbf{I}(\xi)Q^\top. \qquad (2.7)
\end{aligned}$$

Therefore,

$$\begin{aligned}
\Phi_{EI}(\widetilde{\xi}_\ell, F) &= \mathrm{Tr}\Big\{\mathbf{A}\cdot\mathbf{I}^{-1}(\widetilde{\xi}_\ell)\Big\} \\
&= \mathrm{Tr}\Big\{Q^\top\mathbf{A}Q\cdot(Q^\top)^{-1}\mathbf{I}^{-1}(\xi)Q^{-1}\Big\}, \quad \text{by (2.6) and (2.7)} \\
&= \mathrm{Tr}\big\{Q\mathbf{A}\big[Q\cdot Q^{-1}\big]\mathbf{I}^{-1}(\xi)Q^{-1}\big\}, \qquad \text{since } Q^\top = Q \text{ by diagonality} \\
&= \mathrm{Tr}\big\{\mathbf{A}\big[Q\cdot Q^{-1}\big]\mathbf{I}^{-1}(\xi)\big[Q^{-1}\cdot Q\big]\big\}, \quad \text{cyclic commutativity under Trace} \\
&= \mathrm{Tr}\big\{\mathbf{A}\mathbf{I}^{-1}(\xi)\big\} \;=\; \Phi_{EI}(\xi, F). \qquad (2.8)
\end{aligned}$$

We define a new design as the average of ξ^* and $\widetilde{\xi}^*_\ell$,

$$\xi^*_{0.5} = 0.5\xi^* + 0.5\widetilde{\xi}^*_\ell.$$

Then $\xi^*_{0.5}$ is symmetric, and

$$\begin{aligned}
\Phi_{EI}(\xi^*_{0.5}, F) &= \Phi_{EI}(0.5\xi^* + 0.5\widetilde{\xi}^*_\ell, F) \\
&\le 0.5\Phi_{EI}(\xi^*, F) + 0.5\Phi_{EI}(\widetilde{\xi}^*_\ell, F), \qquad \text{by convexity of } \Phi_{EI} \\
&= \Phi_{EI}(\xi^*, F), \qquad \text{by (2.8)}.
\end{aligned}$$

However, $\Phi_{EI}(\xi^*_{0.5}, F)$ cannot be strictly less than $\Phi_{EI}(\xi^*, F)$, as that would violate the optimality of ξ^*. Therefore it must be true that $\Phi_{EI}(\xi^*_{0.5}, F) = \Phi_{EI}(\xi^*, F)$, which implies that there exists a symmetric EI-optimal design. □

2.2 The simple CVX algorithm

Li and Deng (2021) propose a two-step iterative procedure to find EI-optimal designs. Their sequential– multiplicative algorithm performs a greedy search for support points based on the directional derivative of their criterion. Then, the weights of those points are updated using a ratio based on their proven necessary and sufficient conditions of optimality.

We present Algorithm 1 below to compute EI-optimal designs. Note that Algorithm 1 can be applied to other optimality criteria, like D- or A-optimality, very easily. We may refer to this as the "simple" CVX algorithm, since the CVX solver is used only once and with the entire candidate set (line 5 of the algorithm). The matrices in line 4 are needed so that we may express the optimization problem in terms of the weights $w_1, \ldots, w_N$ in CVX. MATLAB code implementing this algorithm is in the Appendix.

Algorithm 1: Using CVX to find EI-optimal designs

1. Choose the candidate set size, N
2. Generate $\Omega_N = \{\mathbf{u}_1, \ldots, \mathbf{u}_N\} \subset \Omega$, if Ω is not already discrete
3. Compute the $\mathbf{A}$ matrix in (2.2)
4. Compute $\mathbf{z}(\mathbf{u}_j)\gamma_j\mathbf{z}(\mathbf{u}_j)^\top$ for $j = 1, \ldots, N$
5. Use CVX to minimize $\Phi_{EI}(\xi_N, F)$. Denote the resulting design as ξ_N^* with weights $\boldsymbol{w}^*$
6. Check the EI-optimality condition using (2.4) and a small tolerance $\delta > 0$.

Line 6 of the algorithm, the final step, requires that $d_{EI}(\mathbf{u}_j, \xi_N^*) \leq \delta$ be satisfied for all $\mathbf{u}_j \in \Omega_N$, where equality is only met at the EI-optimal points in ξ_N^*. It is clear to

see that this is a relaxed version of the optimality condition described in (1.8). The relaxation of the upper bound from zero to $\delta > 0$ is to account for any rounding errors that come from numerical implementation.

In line 2, the matrix $\mathbf{A}$ needs to be computed. Analytical formulations are preferable, but are not always possible or practical. In these cases, a simple numerical integration method is provided in Algorithm 2. We essentially apply a Riemann sum to the matrix of integrands in (2.2). As the number of randomly generated points, n, increases, the average distance between adjacent points decreases.

The next section provides some examples where these algorithms are implemented to find EI-optimal designs. The purpose of the numerical examples is to highlight that (1) CVX is a powerful tool in optimization with easier implementation than most "handwritten" algorithms, and (2) algorithms proposed for optimal designs should not be judged by their convergence speed alone.

Algorithm 2: Monte-Carlo method of integration for matrices

- For a given sample size n and matrix of integrand functions $G(\mathbf{x})$ on S:

1. Compute the volume of the design region as $|S|$
2. Generate n uniformly random points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ in S
3. Compute $\bar{G} := \frac{1}{n} \sum_{i=1}^{n} G(\mathbf{x}_i)$
4. **Return:** $A := \bar{G} \cdot |S|$

2.3 Numerical examples

All algorithms and examples were coded in MATLAB R2023a and executed on a Dell Latitude 9430 laptop computer, with a 12th Gen Intel(R) Core(TM) i7-1265U (1.80

GHz). We set $\delta = 10^{-7}$ for the tolerance value in Algorithm 1.

2.3.1 Polynomial regression

We find EI-optimal designs for a polynomial linear regression model studied in Li and Deng (2021) and Yang et al. (2013). Consider a model in the form of (1.1) with $\mathbf{x} = (x_1, x_2)^\top$, $\Omega = [-1, 1] \times [0, 1]$, $\boldsymbol{\beta} \in \mathbb{R}^5$, and $\mathbf{z}(\mathbf{x}) = (1, x_1, x_1^2, x_2, x_1 x_2)^\top$. We discretize the design region into a lattice of $N = 101^2$ uniformly-spaced points defined by

$$\Omega_N = \left\{ \left(\frac{2i}{101} - 1, \, \frac{j}{101} \right)^\top : 0 \leq i, \, j \leq 101, \quad i, \, j \in \mathbb{Z} \right\}.$$

We consider two probability distributions for the computation of the $\mathbf{A}$ matrix: the uniform distribution with $F_{\mathsf{UNIF}}(\mathbf{x}) = 0.5 x_2 (x_1 + 1)$ for $\mathbf{x} \in \Omega$ and the arc-sine distribution with $F_{\mathsf{ASIN}}(\mathbf{x}) = 4\pi^{-2} \arcsin\Big(\sqrt{0.5(x_1 + 1)}\Big) \arcsin\big(\sqrt{x_2}\big)$ for $\mathbf{x} \in (-1, 1) \times (0, 1)$. Figure 2.1 below is a visual representation of the arc-sine PDF. It is not difficult in either case to compute $\mathbf{A}$ analytically:

$$\mathbf{A}_{\mathsf{UNIF}} = \int_\Omega \mathbf{z}(\mathbf{x}) \mathbf{z}(\mathbf{x})^\top \, dF_{\mathsf{UNIF}}(\mathbf{x}) = \begin{bmatrix} 1 & 0 & 1/3 & 1/2 & 0 \\ 0 & 1/3 & 0 & 0 & 1/6 \\ 1/3 & 0 & 1/5 & 1/6 & 0 \\ 1/2 & 0 & 1/6 & 1/3 & 0 \\ 0 & 1/6 & 0 & 0 & 1/9 \end{bmatrix},$$

$$\mathbf{A}_{\mathsf{ASIN}} = \int_\Omega \mathbf{z}(\mathbf{x}) \mathbf{z}(\mathbf{x})^\top \, dF_{\mathsf{ASIN}}(\mathbf{x}) = \begin{bmatrix} 1 & 0 & 1/2 & 1/2 & 0 \\ 0 & 1/2 & 0 & 0 & 1/4 \\ 1/2 & 0 & 3/8 & 1/4 & 0 \\ 1/2 & 0 & 1/4 & 3/8 & 0 \\ 0 & 1/4 & 0 & 0 & 3/16 \end{bmatrix}.$$

Note that for scale matrix $V = \left[\begin{smallmatrix} a & 0 \\ 0 & b \end{smallmatrix}\right]$ such that $\Omega^V = [-a, a] \times [0, b]$, we can find a

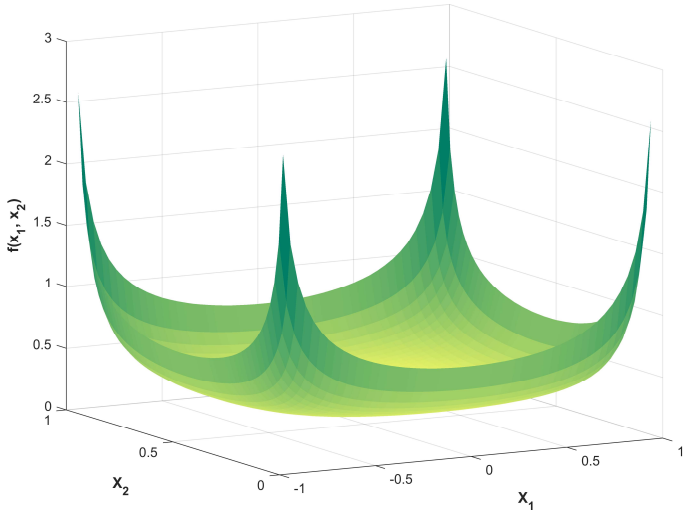


Figure 2.1: Arc-sine PDF $f(x_1, x_2) = dF_{\text{ASIN}}(\mathbf{x})$ on $(-1, 1) \times (0, 1)$

diagonal matrix $Q = \text{Diag}(1, a, a^2, b, ab)$. Therefore, by Theorem 2.1, the EI-optimal design for this model is scale-invariant. Furthermore, both F_{UNIF} and F_{ASIN} are symmetric in x_1. By Theorem 2.3 we can find, for the reflection matrix $T_1 = \left[\begin{smallmatrix} -1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$, a diagonal matrix $Q = \text{Diag}(1, -1, 1, 1, -1)$. Thus, there exists an EI-optimal design which is symmetric in x_1 under both distributions.

Table 2.1 presents our results using Algorithm 1. What we observe is that our six support points are identical in both cases. They exist on the boundary of Ω and satisfy the symmetry property of Theorem 2.3 in x_1. The only change is in their weights, with higher importance placed on the four corners of Ω under the arc-sine distribution than under the uniform distribution. This is expected, as Figure 2.1 shows that the arc-sine PDF is strictly convex. Figure 2.2 demonstrates the optimality condition used to confirm our designs, with the six support points circled in the plots. We can see that the range of d_{EI} is wider under F_{ASIN}, showing the influence of F on the optimization problem. The EI-optimality condition is also clearly satisfied given that the support points are the maximizers of d_{EI}, which is approximately zero at their evaluation.

Support points	Weights	
$(x_1, x_2)^*$	$\boldsymbol{w}^*_{\text{UNIF}}$	$\boldsymbol{w}^*_{\text{ASIN}}$
$(\pm 1, 0)$	0.131	0.158
$(0, 0)$	0.238	0.183
$(0, 1)$	0.238	0.183
$(\pm 1, 1)$	0.131	0.158
$\Phi_{EI}(\xi^*_N, F)$	2.684	3.299
Rel. Eff.	0.997	0.998
run time (s)	16.382	16.051

Table 2.1: EI-optimal designs under F_{UNIF} and F_{ASIN} for a polynomial model; Generated using Algorithm 1. The corners of Ω are indicated by blue text.

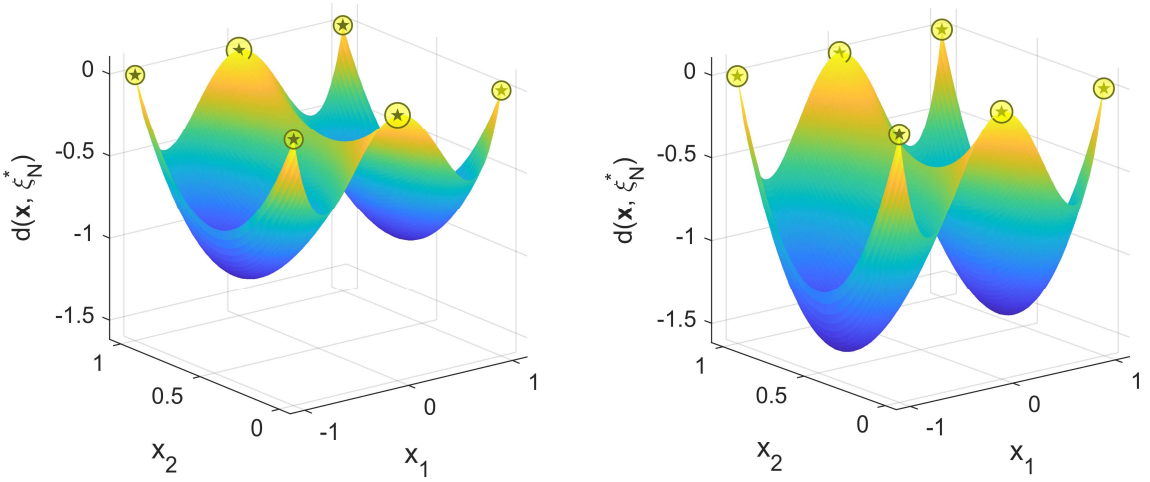


Figure 2.2: EI-optimality condition and six support points (which are circled in the plots) for a 2D polynomial regression model using F_{UNIF} (left) and F_{ASIN} (right)

The relative efficiency is evaluated as our EI-optimal function value divided by that of Li and Deng (2021). If the relative efficiency is less than 1, then our EI-optimal design is better. In Table 2.1, the relative efficiencies are less than 1 for the two cases, so our EI-optimal designs are better. Furthermore, Figure 2.3 overlays our EI-optimal design points over those in Li and Deng (2021). Not only do their designs have far more support points, but they do not have the symmetric property. Our EI-optimal designs

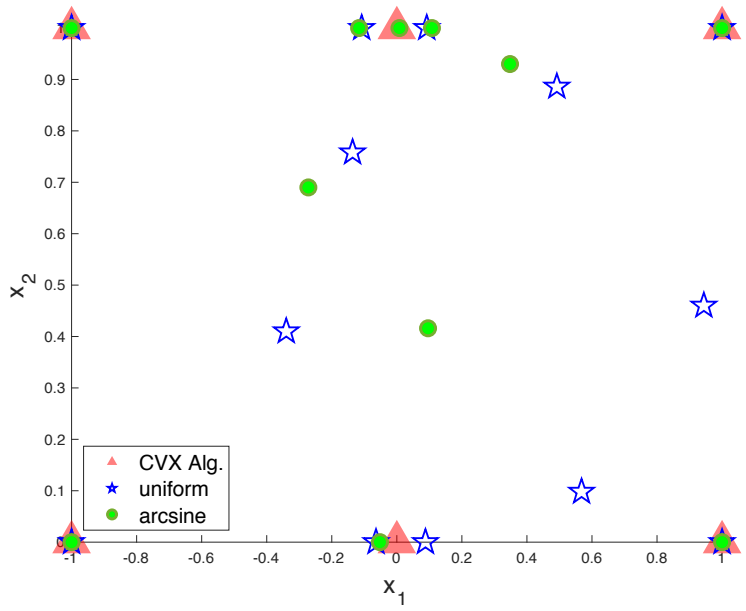


Figure 2.3: Our EI-optimal design points (red colour) versus those presented in Li and Deng (2021) (in blue and green colours)

are symmetric in x_1, which is consistent with the theoretical result. Admittedly we note that our mesh size of $N = 101^2$ points is small, which is why Algorithm 1 does not fail or take a long time to run. CVX cannot handle very large problems in one step. An iterative algorithm is presented in the Chapter 4 that improves run times for large candidate sets.

2.3.2 Logistic regression

For our second example we aim to find an EI-optimal design for a logistic regression model studied in Li and Deng (2021). This GLM has y as a binary response variable and $\mathbf{x}$ as a two-dimensional predictor variable on $\Omega = [-1, 1]^2$. The basis vector is $\mathbf{z}(\mathbf{x}) = (1, x_1, x_2)^\top$ and the parameters are estimated by $\hat{\boldsymbol{\beta}} = (2, 1, -2.5)^\top$. The mean

and variance of the response are found as (1.2) and (1.3), respectively:

$$\mathbb{E}(y|\mathbf{x}_i) = h^{-1}(\eta_i) = \frac{e^{\eta_i}}{1+e^{\eta_i}}, \qquad \text{Var}(y|\mathbf{x}_i) = \mathbb{E}(y|\mathbf{x}_i)(1-\mathbb{E}(y|\mathbf{x}_i)) = \frac{e^{\eta_i}}{(1+e^{\eta_i})^2}.$$

We can then generate the information matrix,

$$\mathbf{I}(\xi_N, \hat{\boldsymbol{\beta}}) = \sum_{i=1}^{N} w_i \left[\frac{e^{\hat{\eta}_i}}{(1+e^{\hat{\eta}_i})^2}\right] \mathbf{z}(\mathbf{u}_i)\mathbf{z}(\mathbf{u}_i)^\top,$$

where similar to the previous example, we construct $\Omega_N = \{\mathbf{u}_1, \ldots, \mathbf{u}_N\}$ with a regular lattice of $N = 101^2 = 10,201$ points on Ω. Furthermore, we use Algorithm 2 with $1,000,000$ points to estimate the $\mathbf{A}$ matrix. There are two notes to be made about the computation of $\mathbf{A}$. Firstly, our distribution function of choice is the uniform distribution, $F_{\mathsf{UNIF}}(\mathbf{x})$. Secondly, the region of integration is a subset of the design space, $S = [0,1]^2 \subset \Omega$ which has a total area of 1 unit^2. Thus, $\mathbf{A}$ is found:

$$\mathbf{A} = \int_{[0,1]^2} \mathbf{z}(\mathbf{x})\mathbf{z}(\mathbf{x})^\top \left(\frac{\partial h^{-1}}{\partial \eta}\right)^2 dF_{\mathsf{UNIF}}(\mathbf{x}) \approx \begin{bmatrix} 0.0321 & 0.0142 & 0.0214 \\ 0.0142 & 0.0088 & 0.0097 \\ 0.0214 & 0.0097 & 0.0161 \end{bmatrix}.$$

The results from our optimization are presented in Table 2.2 and Figure 2.4. We see again that the EI-efficiency is less than one, indicating that our reported design is better than that of Li and Deng (2021). Additionally, the EI-optimal design in Li and Deng (2021) has five support points whereas ours has four. Although the support points are on the boundary of Ω, they are not the four corners of Ω and the design is not symmetric.

Figure 2.4 presents both a 2D and 3D surface plot of the optimality condition function. This is simply to highlight the interpretation of the 2D plot, which will be used in the later chapters of this thesis. We can see that the optimality condition is

Support points $(x_1, x_2)^*$	Weights $\boldsymbol{w}^*$
$(-1, -0.3)$	0.2508
$(-1, 1)$	0.1891
$(1, 0.7)$	0.2301
$(1, 1)$	0.3300
$\Phi_{EI}(\xi_N^*, F_{\text{UNIF}})$	0.2750
Rel. Eff.	0.9977
run time (s)	22.6125

Table 2.2: EI-optimal design for a 2D logistic model; Generated via Algorithm 1.

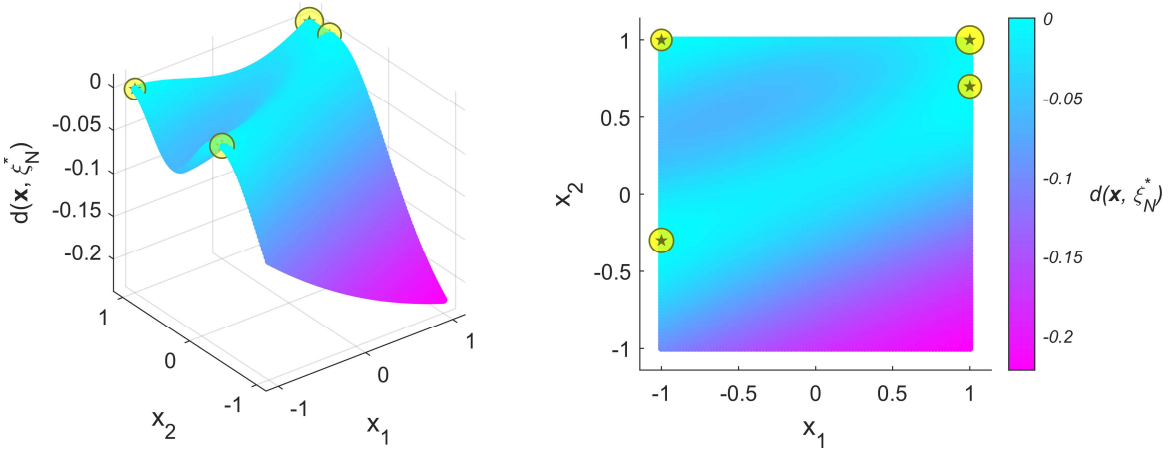


Figure 2.4: *EI*-optimal design points from Algorithm 1 indicated by circles in the plots and overlaid on the EI-optimality condition surface

satisfied, verifying that our design is EI-optimal. Interestingly, the support points are not constrained to the area of integration, $S = [0, 1]^2$.

2.3.3 Mixture models

Mixture models take the form of (1.1) with an added constraint that for $\mathbf{x} \in \mathbb{R}^p$, all variates (called *ingredients* in this context) $x_1, \ldots, x_p$ are non-negative and sum to

unity. This constraint on the p ingredients restricts the design region to a simplex shape in $\mathbb{R}^{p-1}$, such that

$$\Omega = \{\mathbf{x} \in \mathbb{R}^p \mid x_1 + x_2 + \cdots + x_p = 1,\ x_i \geq 0,\ \forall i = 1, 2, \ldots, p\}.$$

For example, the design region for a mixture model with two ingredients is the line segment connecting the vertices $(0, 1)$ and $(1, 0)$ in the 2D plane. For three ingredients, it is the equilateral triangle formed with the vertices $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$. And with four ingredients, it is the tetrahedron contained by $(1, 0, 0, 0)$, $(0, 1, 0, 0)$, $(0, 0, 1, 0)$, and $(0, 0, 0, 1)$. A $(p-1)$-simplex has volume $\frac{1}{(p-1)!}$. Figure 2.5 shows the three-ingredient design region.

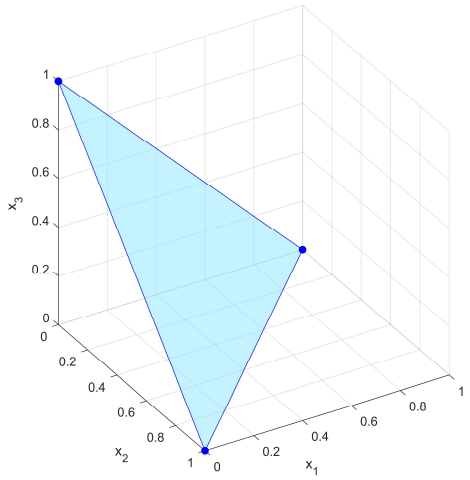


Figure 2.5: Design region for a mixture model with three ingredients

Consider a special cubic model in $p \geq 3$ ingredients (Scheffé, 1958, as cited in Goos et al., 2016), which has $p + \binom{p}{2} + \binom{p}{3}$ terms in its basis vector:

$$\mathbb{E}(y) = \sum_{i=1}^{p} \beta_i x_i + \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} \beta_{ij} x_i x_j + \sum_{i=1}^{p-2} \sum_{j=i+1}^{p-1} \sum_{k=j+1}^{p} \beta_{ijk} x_i x_j x_k.$$

We use Algorithm 1 to compute I-optimal designs for the special cubic model with $p = 3, 4$, and 5 ingredients. The analytical form of $\mathbf{A}$ has entries equal to

$$\frac{\prod_{i=1}^{p} d_i!}{|\Omega|(p-1+\sum_{i=1}^{p} d_i)!},$$

where d_i refers to the degree of the i^{th} ingredient in each entry of the integrand matrix for $i = 1, \ldots, p$. For example, the three-ingredient special cubic model has

$$\mathbf{A} = \int_{\Omega} \mathbf{z}(\mathbf{x})\mathbf{z}(\mathbf{x})^{\top} \, dF_{\mathsf{UNIF}}(\mathbf{x}) = \begin{bmatrix} 1/6 & 1/12 & 1/12 & 1/30 & 1/30 & 1/60 & 1/180 \\ 1/12 & 1/6 & 1/12 & 1/30 & 1/60 & 1/30 & 1/180 \\ 1/12 & 1/12 & 1/6 & 1/60 & 1/30 & 1/30 & 1/180 \\ 1/30 & 1/30 & 1/760 & 1/90 & 1/180 & 1/180 & 1/630 \\ 1/30 & 1/60 & 1/30 & 1/180 & 1/90 & 1/180 & 1/630 \\ 1/60 & 1/30 & 1/30 & 1/180 & 1/180 & 1/90 & 1/630 \\ 1/180 & 1/180 & 1/180 & 1/630 & 1/630 & 1/630 & 1/2520 \end{bmatrix}.$$

Furthermore, we discretize Ω to include the full simplex-centroid design points according to Goos et al. (2016). Meaning that for p ingredients: the pure components are the $\binom{p}{1}$ vertices defining the simplex, the binary mixtures are the $\binom{p}{2}$ permutations of $(1/2, 1/2, 0, \ldots, 0)^{\top}$, the ternary mixtures are the $\binom{p}{3}$ permutations of $(1/3, 1/3, 1/3, 0, \ldots, 0)^{\top}$, and so forth. Hence, Ω_N consists of a total of $N = \sum_{i=1}^{p} \binom{p}{i}$ points for a given number of ingredients, p.

Table 2.3 presents our I-optimal designs. We compare our designs with those in Goos et al. (2016), which are generated using a multiplicative algorithm. A relative efficiency < 1 indicates that our models are better. We can see that our designs are as good as those presented in Goos et al. (2016) for $p = 3$ and $p = 4$ ingredients, and better for $p = 5$ ingredients. Furthermore, all the points in Ω_N are support points of the EI-optimal design. Figure 2.6 below is a ternary plot which demonstrates the optimality condition function for the three-ingredient model. We generate $10,000$ random points in the simplex to verify that the condition is satisfied, given that N is so small.

mixtures	number of ingredients (p) 3	4	5
pure	$(3, 0.0925)$	$(4, 0.0426)$	$(5, 0.0227)$
binary	$(3, 0.1483)$	$(6, 0.0557)$	$(10, 0.0248)$
ternary	$(1, 0.2776)$	$(4, 0.0991)$	$(10, 0.0409)$
quaternary		$(1, 0.0988)$	$(5, 0.0414)$
quinary			$(1, 0.0230)$
N	7	15	31
$\Phi_I(\xi_N^*)$	3.7543	5.8607	8.4005
Rel. Eff.	1.0000	1.0000	0.9998
run time (s)	0.5372	0.6590	2.1765

Table 2.3: I-optimal weights for the special cubic mixture model with three, four, and five ingredients. Table entries show the number of permutations for each mixture and the corresponding weight of each one.

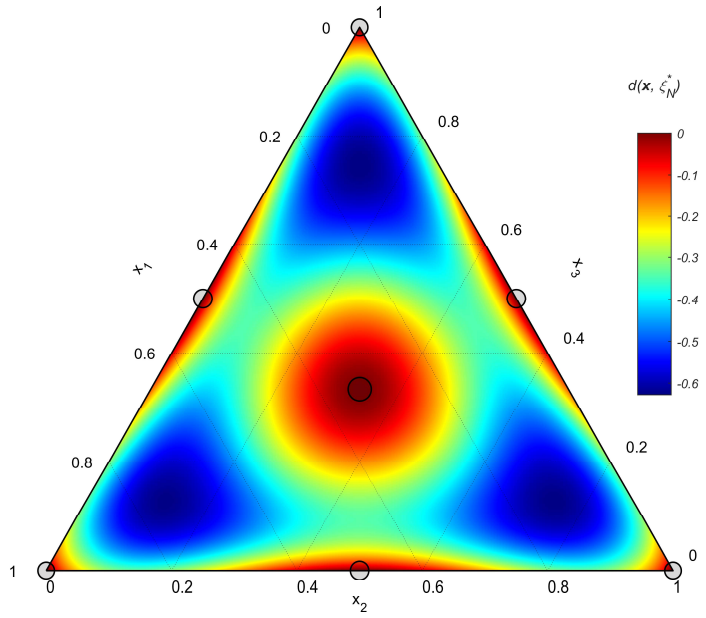


Figure 2.6: I-optimal design points (black circles) and optimality condition function for a three-ingredient, special cubic mixture model; Generated via Algorthim 1.

2.4 Summary

We discussed the EI-optimality criterion and condition. The reflection-symmetry and scale-invariance property of the EI criterion were derived. We applied the CVX solver in MATLAB to find EI-optimal designs for GLMs as well as mixture models. The CVX solver was effective and the resulting EI-optimal designs were better than those previously found in the literature. The MATLAB code for these numerical examples can be found in Section A.2 of the Appendix.

Chapter 3

Discretizing continuous design spaces

In this brief chapter, we explore the relationship between optimal designs on discrete and continuous design spaces. Two theoretical results are proven in Section 3.1, including an upper bound between the discrete and continuous optimality condition for a Φ-optimality criterion. In Section 3.2, we provide some practical guidelines for discretizing a design region. Section 3.3 is an example of a D-optimal design on an irregular arbelos region, where the candidate set Ω_N is formed using the guidelines of Section 3.2.

3.1 The relationship between optimal designs on discrete and continuous spaces for LMs

In Chapter 1, we introduced the optimization problem on the discrete space and defined ξ_N^* as the solution to Problem (1.7). But what about on the continuous space? Designs ξ on a continuous space Ω cannot be specified by the weight vector alone because there are infinitely many design points in Ω. Hence, the analogous optimization problem on the continuous space to Problem (1.7) is far more challenging to solve. Let ξ^*

denote the Φ-optimal design on Ω. We study the relationship between ξ_N^* and ξ^*, and derive two important theoretical results that connect them. We seek to evaluate the performance of the optimal design ξ_N^* on Ω rather than Ω_N. There are two cases to consider: when the support points of ξ^* are in Ω_N, and when they are not.

Theorem 3.1. *If Ω_N contains all the support points in ξ^*, then ξ_N^* is also an optimal design on Ω with $\Phi(\xi_N^*) = \Phi(\xi^*)$.*

Proof of Theorem 3.1: Let $\mathbf{x}_1, \ldots, \mathbf{x}_m$ be the support points of ξ^*, and their corresponding weights are $w_1^*, \ldots, w_m^*$. Assume Ω_N contains $\mathbf{x}_1, \ldots, \mathbf{x}_m$. Without loss of generality, let $\Omega_N = \{\mathbf{x}_1, \ldots, \mathbf{x}_m, \mathbf{u}_{m+1}, \ldots, \mathbf{u}_N\} \subset \Omega$. It is clear that ξ^* is a design measure on Ω_N with weight vector $\boldsymbol{w} = (w_1^*, \ldots, w_m^*, 0, \ldots, 0)^\top$. Thus,

$$\Phi(\xi_N^*) \leq \Phi(\xi^*). \tag{3.1}$$

On the other hand, ξ^* minimizes $\Phi(\xi)$ over all ξ on Ω and ξ_N^* minimizes $\Phi(\xi_N^*)$ over all ξ_N on Ω_N. Since $\Omega_N \subset \Omega$ and ξ_N on Ω_N is also a design measure on Ω, it is also true that

$$\Phi(\xi_N^*) \geq \Phi(\xi^*). \tag{3.2}$$

Combining (3.1) and (3.2), we have $\Phi(\xi_N^*) = \Phi(\xi^*)$. Therefore, ξ_N^* is a Φ-optimal design on Ω. □

Often ξ_N^* and ξ^* are exactly the same for many applications. If the optimal designs on Ω are not unique, then ξ_N^* and ξ^* may be different. The result in Theorem 3.1 is helpful for discretizing the design space Ω, which is discussed in next section.

When Ω_N does not contain all the support points in ξ^*, we derive a useful result to see if ξ_N^* can be a good approximation of ξ^*. In particular, we investigate the optimality criterion, $d_\Phi(\mathbf{x}, \xi_N^*)$, for any $\mathbf{x} \in \Omega$. Recall the equivalence theory for various optimality

criteria. For c-, I-, and L-optimality, $d_\Phi(\mathbf{x}, \xi^*)$ can be written as a general form below,

$$d_\Phi(\mathbf{x}, \xi^*) = \mathbf{z}(\mathbf{x})^\top \mathbf{I}^{-1}(\xi^*)\mathbf{C}\mathbf{C}^\top \mathbf{I}^{-1}(\xi^*)\mathbf{z}(\mathbf{x}) - \mathrm{Tr}\{\mathbf{C}^\top \mathbf{I}^{-1}(\xi^*)\mathbf{C}\} \tag{3.3}$$

where the choice of the constant $q \times r$ $(r \leq q)$ matrix $\mathbf{C}$ determines c-, I-, or L-optimality. Note that we minimize

$$\Phi(\xi) = \mathrm{Tr}\{\mathbf{C}^\top \mathbf{I}^{-1}(\xi)\mathbf{C}\} \tag{3.4}$$

over ξ on Ω to find c-, I-, and L-optimal designs ξ^*. When $\mathbf{C} = I_q$ (the identity matrix of size q), (3.4) becomes the A-optimality criterion. Furthermore, the EI- or I-optimality criteria are specified by (3.4) when $\mathbf{C}\mathbf{C}^\top = \mathbf{A}$ as in (2.2).

Theorem 3.2. *Let ξ_N^* be the Φ-optimal design on Ω_N, and let $\overline{\Omega}$ be a convex region that includes Ω. Assume all the entries of $\frac{\partial \mathbf{z}(\mathbf{x})}{\partial \mathbf{x}^\top}$ are bounded functions for all $\mathbf{x} \in \overline{\Omega}$, and let B denote the bound for the absolute value of each entry of $\frac{\partial \mathbf{z}(\mathbf{x})}{\partial \mathbf{x}^\top}$ for all $\mathbf{x} \in \overline{\Omega}$. For a given point $\mathbf{x} \in \Omega$, find a point $\mathbf{u}_j \in \Omega_N$ that is the closest to $\mathbf{x}$. Then the function $d_\Phi(\mathbf{x}, \xi_N^*)$ can be calculated in the two following situations:*

(i) If $\mathbf{x} = \mathbf{u}_j$, then $d_\Phi(\mathbf{x}, \xi_N^*) = d_\Phi(\mathbf{u}_j, \xi_N^*) \leq 0$;

(ii) If $\mathbf{x} \neq \mathbf{u}_j$, then

$$\begin{aligned} |d_\Phi(\mathbf{x}, \xi_N^*) - d_\Phi(\mathbf{u}_j, \xi_N^*)| \;\leq\; & q\lambda_{max}(\mathbf{M})\, B^2 \left\|\mathbf{x} - \mathbf{u}_j\right\|_1^2 \\ & +\; 2(q\lambda_{max}(\mathbf{M}))^{1/2}\, B \left\|\mathbf{x} - \mathbf{u}_j\right\|_1 \sqrt{\mathrm{Tr}\{\mathbf{C}^\top \mathbf{I}^{-1}(\xi_N^*)\mathbf{C}\}}\ , \end{aligned}$$

where $\mathbf{M} = \mathbf{I}^{-1}(\xi_N^*)\mathbf{C}\mathbf{C}^\top \mathbf{I}^{-1}(\xi_N^*)$, $\lambda_{\max}(\mathbf{M})$ is the largest eigenvalue of $\mathbf{M}$, and $\|\cdot\|_1$ is the L_1 norm.

Proof of Theorem 3.2: For (i), the result is obvious. For (ii), consider that the vector $\mathbf{z}(\mathbf{x})$ can be written as

$$\mathbf{z}(\mathbf{x}) = \mathbf{z}(\mathbf{u}_j) + \mathbf{D}_j(\mathbf{x} - \mathbf{u}_j), \tag{3.5}$$

where $\mathbf{D}_j$ is a $q \times p$ matrix and is defined as

$$\left.\frac{\partial \mathbf{z}(\mathbf{x})}{\partial \mathbf{x}^\top}\right|_{\mathbf{x}=\mathbf{v}_j},$$

and $\mathbf{v}_j$ is a point between $\mathbf{x}$ and $\mathbf{u}_j$. Using (3.3) and (3.5), we obtain

$$\begin{aligned} d(\mathbf{x}, \xi_N^*) = & \ \mathbf{z}(\mathbf{x})^\top \left[\mathbf{I}^{-1}(\xi_N^*)\mathbf{C}\mathbf{C}^\top\mathbf{I}^{-1}(\xi_N^*)\right]\mathbf{z}(\mathbf{x}) - \ \mathrm{Tr}\{\mathbf{C}^\top\mathbf{I}^{-1}(\xi_N^*)\mathbf{C}\} \\ = & \ [\mathbf{z}(\mathbf{u}_j) + \mathbf{D}_j(\mathbf{x}-\mathbf{u}_j)]^\top \mathbf{M}\,[\mathbf{z}(\mathbf{u}_j) + \mathbf{D}_j(\mathbf{x}-\mathbf{u}_j)] \\ & - \ \mathrm{Tr}\{\mathbf{C}^\top\mathbf{I}^{-1}(\xi_N^*)\mathbf{C}\}. \end{aligned}$$

This leads to

$$\begin{aligned} & |d(\mathbf{x}, \xi_N^*) - d(\mathbf{u}_j, \xi_N^*)| \\ & = \left|(\mathbf{D}_j(\mathbf{x}-\mathbf{u}_j))^\top\mathbf{M}\mathbf{D}_j(\mathbf{x}-\mathbf{u}_j) + 2\mathbf{z}(\mathbf{u}_j)^\top\mathbf{M}\mathbf{D}_j(\mathbf{x}-\mathbf{u}_j)\right| \\ & \le \left|(\mathbf{D}_j(\mathbf{x}-\mathbf{u}_j))^\top\mathbf{M}\mathbf{D}_j(\mathbf{x}-\mathbf{u}_j)\right| + 2\left|\mathbf{z}(\mathbf{u}_j)^\top\mathbf{M}\mathbf{D}_j(\mathbf{x}-\mathbf{u}_j)\right| \\ & \le \tau + 2\tau^{1/2}\Big(\mathbf{z}(\mathbf{u}_j)^\top\mathbf{M}\mathbf{z}(\mathbf{u}_j)\Big)^{1/2}, \end{aligned} \tag{3.6}$$

where $\tau = (\mathbf{D}_j(\mathbf{x}-\mathbf{u}_j))^\top\mathbf{M}\mathbf{D}_j(\mathbf{x}-\mathbf{u}_j)$. From the optimality condition in (1.8) and (3.3), it follows that

$$\mathbf{z}(\mathbf{u}_j)^\top\mathbf{M}\mathbf{z}(\mathbf{u}_j) \le \mathrm{Tr}\{\mathbf{C}^\top\mathbf{I}^{-1}(\xi_N^*)\mathbf{C}\}. \tag{3.7}$$

Note that $\mathbf{v}_j$ is a point between $\mathbf{x}$ and $\mathbf{u}_j$, and $\mathbf{x}$ and $\mathbf{u}_j$ are in $\Omega \subset \overline{\Omega}$ (a convex region). Point $\mathbf{v}_j$ must be in $\overline{\Omega}$. From the assumptions, each entry of $\mathbf{D}_j$ is bounded by B. It is easy to show that

$$(\mathbf{D}_j(\mathbf{x}-\mathbf{u}_j))^\top\mathbf{D}_j(\mathbf{x}-\mathbf{u}_j) \le qB^2\|\mathbf{x}-\mathbf{u}_j\|_1^2.$$

Also,

$$\begin{aligned} \tau &= (\mathbf{D}_j(\mathbf{x}-\mathbf{u}_j))^\top\mathbf{M}\mathbf{D}_j(\mathbf{x}-\mathbf{u}_j) \\ &\le \lambda_{max}(\mathbf{M})(\mathbf{D}_j(\mathbf{x}-\mathbf{u}_j))^\top\mathbf{D}_j(\mathbf{x}-\mathbf{u}_j) \\ &\le qB^2\lambda_{max}(\mathbf{M})\|\mathbf{x}-\mathbf{u}_j\|_1^2. \end{aligned} \tag{3.8}$$

Putting (3.7) and (3.8) into (3.6) gives the result in part (ii). □

The convex region $\overline{\Omega}$ is needed for the Taylor expansion of function $d_\Phi(\mathbf{x}, \xi_N^*)$ in the proof. If Ω is convex, then we can set $\overline{\Omega} = \Omega$. If Ω is not convex, any convex region that includes Ω works. Many regression design problems satisfy the assumption of bounded functions for the absolute value of each entry of $\frac{\partial \mathbf{z}(\mathbf{x})}{\partial \mathbf{x}^\top}$, and it is easy to verify the assumption. If each entry of $\frac{\partial \mathbf{z}(\mathbf{x})}{\partial \mathbf{x}^\top}$ is a continuous function of $\mathbf{x}$ and Ω is a bounded region, then there exists an upper bound B for the absolute value of each entry.

It is clear that Theorem 3.2 can be applied for A-, c-, I-, and L-optimality easily. Furthermore, it can also be applied for D-optimality by setting $\mathbf{C}\mathbf{C}^\top = \mathbf{I}(\xi_N^*)$. The takeaway of Theorem 3.2 is as follows. For any point $\mathbf{x} \in \Omega$, if $\|\mathbf{x} - \mathbf{u}_j\|_1$ is very small, where $\mathbf{u}_j \in \Omega_N$ is the closest point to $\mathbf{x}$, then $|d(\mathbf{x}, \xi_N^*) - d(\mathbf{u}_j, \xi_N^*)|$ is also a small number. This implies that the optimal design ξ_N^* on Ω_N satisfies $d_\Phi(\mathbf{x}, \xi_N^*) \leq \delta$ for all $\mathbf{x} \in \Omega$, where δ is a small positive number. Furthermore, this means that ξ_N^* is a good approximation of a Φ-optimal design on Ω. For regular shaped design spaces, taking Ω_N as the Cartesian product of many equally spaced grid points in each dimension makes $\|\mathbf{x} - \mathbf{u}_j\|_1$ small for any point $\mathbf{x} \in \Omega$.

3.2 Constructing Ω_N

This section is devoted to the process of discretizing a continuous space Ω in $\mathbb{R}^p$. The set Ω_N is simply a finite set of sampled points from Ω which covers Ω well. When a design region $\Omega \subset \mathbb{R}^p$ takes the form of a box,

$$\prod_{i=1}^{p} [a_i, b_i] := [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_p, b_p],$$

it is called *regular*. Any design region that is not a box is called *irregular*. Here are guidelines to discretize a continuous design space Ω so that the optimal designs on the

discrete design space Ω_N can approximate the optimal designs on Ω well.

(i) By Theorem 3.1, construct Ω_N to include all the known support points of ξ^* on Ω.

(ii) By Theorem 3.2, Ω_N should cover Ω well such that $\min_{\mathbf{u}_j \in \Omega_N} \|\mathbf{x} - \mathbf{u}_j\|_1$ is small for any $\mathbf{x} \in \Omega$. This can lead to a huge N when there are several design variables.

(iii) If the design space is symmetric with respect to a transformation and ξ^* on Ω is symmetric from theoretical results, then Ω_N should be symmetric too.

(iv) For regular design spaces, such as an interval or a cube, Ω_N can be formed by the Cartesian product of equally spaced grid points for each design variable and the boundary points of each variable are included in the equally-spaced grid points. This is called a lattice.

(v) For irregular design spaces, Ω_N should include many points on the boundary of Ω and a lot of points in the interior that cover Ω well. Special points of Ω, like centroids or other centre points, should also be included in Ω_N as well.

(vi) Wherever a lattice is used, it is beneficial to use an odd number of points along each coordinate axis. This ensures that the midpoints of the intervals are included in Ω_N.

(vii) For some practical applications, Ω_N should only include the points that are reasonable to implement in the experiment.

What we are essentially proposing with these guidelines is that some sub-regions of Ω may require more coverage by Ω_N than others based on the theoretical properties of ξ^* on Ω. When care is taken to preserve the theoretical properties of Ω in its discretization Ω_N, we may ensure that ξ_N^* remains approximate to ξ^* for the given criterion Φ.

As we saw in Chapter 2, lattices cover boxes efficiently as both their interiors and boundaries are proportionately included in Ω_N. When $N = m^p$ for some integer $m \geq 2$, then $\frac{(m-2)^p}{N}$ is the proportion of interior points in the lattice. This is not true for irregular regions. One simple way to discretize an irregular region is through rejection sampling.

Algorithm 3: Rejection Sampling Method for irregular Ω

1. Let $\overline{\Omega}$ denote a bounding box of Ω
2. Define $M := \left\lceil \frac{|\overline{\Omega}|}{|\Omega|} \cdot N \right\rceil$, where N is user-defined and $\lceil \cdot \rceil$ is the ceiling operator
3. Create a lattice of approximately M points covering $\overline{\Omega}$
4. Reject all points in the lattice that lie outside Ω
5. Return Ω_N as the remaining ($\approx N$) points in Ω

In the third step of Algorithm 3, there are many ways to create the lattice. One is to use $\approx \sqrt[p]{M}$ equally-spaced points along each coordinate axis. This is easy to do, but if the lengths of the box are not equal, some dimensions will have larger distance between adjacent points than others. If equal distance between adjacent points in all dimensions is desired, then one would generate $\approx (b_i - a_i)\sqrt[p]{\frac{M}{|\Omega|}}$ equally-spaced points along the i^{th} coordinate for all $i = 1, \ldots, p$. It is best to keep this number odd, as per guideline (vi) above.

3.3 A brief example

An arbelos is a two-dimensional shape bounded by three semicircles in a special configuration. Figure 3.1 below shows a plot of the following arbelos design region,

$$\Omega = \Big\{ \mathbf{x} = (x_1, x_2)^\top \;\Big|\; x_1^2 + x_2^2 \leq 1,\ (x_1 - 0.4)^2 + x_2^2 \geq 0.6^2, \\ (x_1 + 0.6)^2 + x_2^2 \geq 0.4^2,\ x_2 \geq 0,\ \mathbf{x} \in \mathbb{R}^2 \Big\},$$

as well as its bounding box $\overline{\Omega} = [-1, 1] \times [0, 1]$, which is represented by the red lines. We want to find a D-optimal design for an arbitrary Poisson model over Ω with $\mathbf{z}(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)^\top$ and $\hat{\boldsymbol{\beta}} = (1, 1, 1, 1, 1, 1)^\top$.

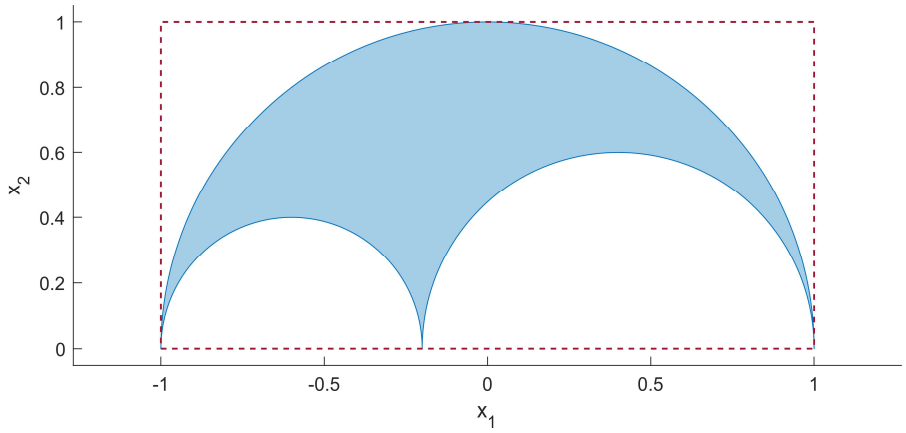


Figure 3.1: The arbelos design region in $\mathbb{R}^2$ shaded in blue, and its bounding box $\overline{\Omega}$ represented by the red dashed lines. Here, $\left|\overline{\Omega}\right| = 2$ and $|\Omega| \approx 0.7540$.

We assume that $y_i = y|\mathbf{x}_i \sim \text{Poisson}(\mu_i)$, with rate parameter $\mu_i > 0$, for all observations in the sample. Recall that under the Poisson distribution, $\mathbb{E}(y_i) = \text{Var}(y_i) = \mu_i$. To map the range of the mean response from $(0, \infty)$ to $(-\infty, \infty)$, we can use the log-link function: $h(\mu_i) = \ln \mu_i = \eta_i$. Thus for a Poisson GLM, we have

$$\mathbb{E}(y_i) = \text{Var}(y_i) = h^{-1}(\eta_i) := e^{\mathbf{z}(\mathbf{x}_i)^\top \boldsymbol{\beta}},$$

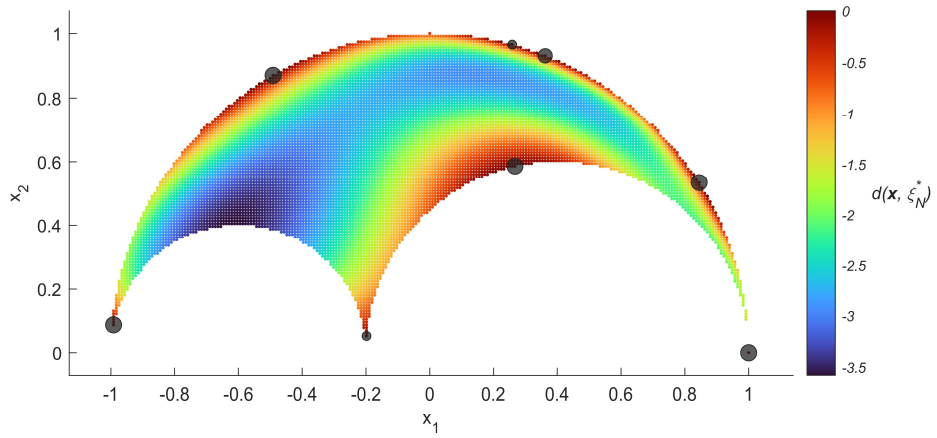
and the information matrix in (1.5) becomes $\mathbf{I}(\xi_N, \hat{\boldsymbol{\beta}}) = \sum_{i=1}^{N} w_i \, e^{\hat{\eta}_i} \, \mathbf{z}(\mathbf{u}_i)\mathbf{z}(\mathbf{u}_i)^\top$.

We will use the simple CVX Algorithm to compute the D-optimal design using two different discretizations of Ω. The D-optimality criterion is defined in (1.9). Let Ω_N denote the set generated exclusively via Algorithm 3, and Ω'_N the set which samples some boundary points in addition to the lattice from Algorithm 3. For Ω_N, we make a lattice of $M = 27,260$ points using a grid of 233×117 equally-spaced points on $\overline{\Omega}$. We find that $10,144$ points are accepted from this lattice to form Ω_N. Only two of these points exists on the boundary. Algorithm 1 takes 50.5 seconds to solve for the D-optimal design on this region, which has 8 support points and an objective value of $\Phi_D(\xi_N^*) = -1.2778$.

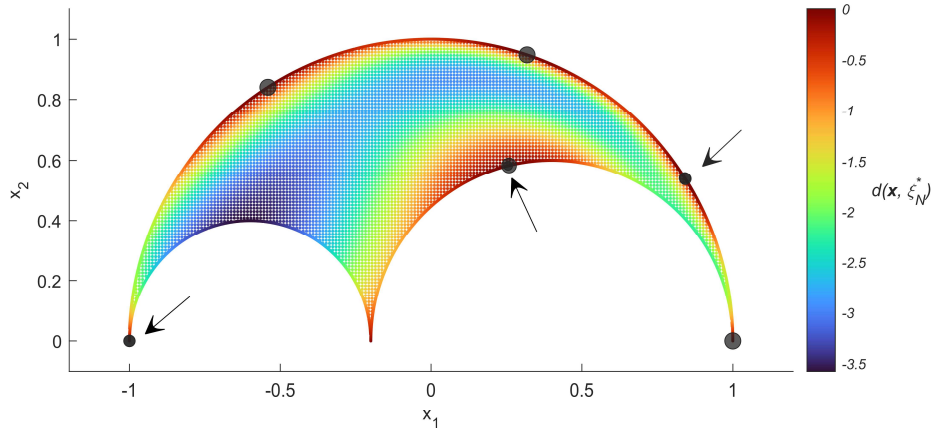
For the second set, Ω'_N, we first allocate $2,000$ points to be sampled along the boundary explicitly; $1,000$ for the top semicircle and 400 and 600 for the left and right concave circles, respectively. A lattice of $M = 17,205$ (185×93) points on $\overline{\Omega}$ results in $6,371$ lying inside Ω, so that $8,371$ points comprise Ω'_N in total. Algorithm 1 takes 33.8 seconds to find the D-optimal design on Ω'_N, which has 9 support points and an objective value of $\Phi_D\big(\xi_N'^*\big) = -1.3396$. Hence, the design on Ω_N is only 95.4% as efficient as that on Ω'_N, even though it has a finer mesh of points. Including the boundary points is better for the optimal design problem.

Figure 3.2 presents the optimality condition as well as the support points over both discrete design regions. Notice that Ω_N is visibly staggered near the boundaries and especially the three apexes at $(\pm 1, 0)$ and $(-0.2, 0)$. In comparison, Ω'_N is much smoother and covers the boundaries well. Also, the D-optimal design on Ω'_N has 9 support points, but only six appear to be visible in Figure 3.2 (b). This is addressed by the arrows. Each arrow in the plot indicates a cluster of *two* support points. These pairs are very close to each other, and without zooming into the image the circles appear to overlap. This may indicate that the true support points near the clusters

were not included in Ω'_N.



(a) Ω_N, found via rejection sampling only. $N = 10,144$.



(b) Ω'_N, includes boundary sampling. $N = 8,440$.

Figure 3.2: D-optimal designs using two different candidate sets on Ω; Generated via Algorithm 1 using a tolerance value of $\delta = 10^{-6}$. Sizes of circles are proportional to the optimal weights, and arrows indicate clusters of 2 support points.

MATLAB code for this chapter can be found in Section A.3 of the Appendix. In the next chapter, we provide several applications with irregular design spaces and use the guidelines to discretize design spaces. Since N can be large, we will also develop an innovative algorithm to compute optimal designs for the challenging optimization problems. This algorithm will incorporate clustering at the end of iteration.

Chapter 4

Iterative procedure for large design problems via CVX solver

In this chapter, we consider models with large candidate sets for their design spaces. We consider "large" to be more than 10,000 candidate points. The CVX solver, though excellent with small optimization problems, has poor performance with large ones. So, rather than just running it once, we opt to incorporate CVX in an iterative algorithm. The main algorithm of this thesis is presented in Section 4.1. Starting with an initial sample from the design region, points are sequentially added to the working candidate set according to a relaxed optimality rule. This rule is compared with the one that is commonly used for sequential algorithms in the literature. Some other suggestions are made regarding the stopping criterion and sequential updating. Numerical examples are presented in Section 4.2 and the chapter is concluded with a brief analysis of the proposed algorithm in Section 4.3.

4.1 The iterative CVX algorithm

Here we present the algorithm and provide suggestions for its inputs and decision rules. For input, one needs to have a discrete design region Ω_N, an optimality criterion $\Phi(\xi_N)$ and its respective optimality condition $d_\Phi(\mathbf{u}_j, \xi_N^*)$, two small tolerance values $\delta_1 \geq \delta_2 > 0$, a size factor $0 \leq \alpha \leq 1$, a maximum number of iterations $max.iter$, and an initial sample size $N_0 \leq 1000$. The optional use of a clustering scheme or a reflection matrix, T, is discussed in the remarks following the algorithm.

Algorithm 4: Iterative CVX algorithm for Φ-optimal designs

1. Set $d.max = 0$
2. Select an initial sample $\Omega_{(0)} \subset \Omega_N$ of size N_0
3. Set $m = 1$. **While** $m < max.iter$**:**
4. Use CVX (Algorithm 1) to find $\xi_{m-1}^* := \left(\Omega_{(m-1)},\, \boldsymbol{w}_{(m-1)}^*\right)$
5. Let $\xi_m^* = \left(\Omega_{(m)},\, \boldsymbol{w}_{(m)}^*\right)$ be the subset of ξ_{m-1}^* with weights $\geq \delta_1$
6. Compute $d_\Phi\big(\mathbf{u}_j, \xi_{m-1}^*\big)$ for all $\mathbf{u}_j \in \Omega_N$
7. **If** $\left| d.max - \max\limits_{\mathbf{u}_j \in \Omega_N} d_\Phi\big(\mathbf{u}_j, \xi_{m-1}^*\big) \right| < \delta_2$**:** **Break.**
8. Set $d.max = \max\limits_{\mathbf{u}_j \in \Omega_N} d_\Phi\big(\mathbf{u}_j, \xi_{m-1}^*\big)$
9. Find $\Lambda_{(m)} := \big\{\mathbf{u}_j \in \Omega_N \mid d_\Phi\big(\mathbf{u}_j, \xi_{m-1}^*\big) \geq \alpha \cdot d.max\big\}$
10. Update $\Omega_{(m)} = \Omega_{(m)} \cup \Lambda_{(m)}$
11. *If applicable:* Update $\Omega_{(m)} = \Omega_{(m)} \cup T\Omega_{(m)}$, where $T : \Omega_N \to \Omega_N$
12. Increment $m = m + 1$
13. **end While.**
14. *Optional:* Update $\xi_{(m)}^*$ to merge any clustered points
15. Return $\xi_{(m)}^*$ as the Φ-optimal design, ξ_N^*

Remark 1 (Initialization)**.** $\Omega_{(0)}$ is chosen by user preference. We take a random sample of points all across the design region, but one may also choose to sample from the boundary or the interior exclusively. The choice of $\Omega_{(0)}$ may influence the convergence speed. If N is not too large (i.e. less than 10,000) one can set $\Omega_{(0)} = \Omega_N$ and $N_0 = N$ to mimic the simple CVX Algorithm 1 from Chapter 2.

Remark 2 (Updating $\Omega_{(m)}$)**.** Line 5 is the deletion rule and lines 10-11 describe the adding rule used to update the working set, $\Omega_{(m)}$, incrementally. We can see that the m^{th} working set is updated by first removing the non-contributing points, then adding a *neighbourhood set*, $\Lambda_{(m)}$. This set (defined in line 9) contains the points in Ω_N at which the optimality condition values are within a neighbourhood of the maximum, $d.max$. The size of the neighbourhood is determined by α. When $\alpha = 1$, line 10 becomes the traditional adding rule of sequential algorithms in the literature. The bottleneck in the algorithm comes from line 6; if the candidate set is large, re-evaluating $d_\Phi(\mathbf{u}_j, \xi_N^*)$ for all N points will be slow. This computation is repeated in each iteration. By reducing the number of iterations, we reduce its number of re-computations. And to do that, we need to rule out as many points as possible at each update. Adding too many points will slow down the CVX solver, but adding in only one point at a time will require more iterations. We found from trial and error that $\alpha = 0.5$ is an ideal and efficient size factor for the neighbourhood.

Line 11 describes an optional component to the adding rule. If there exists a reflection symmetry about Ω_N which can be described by a $p \times p$ matrix T, then one may also wish to include the mirrored points of the working set in its updating. For example, $T = T_\ell$ may describe a reflection symmetry about x_ℓ for some $\ell = 1, \ldots, p$. Or it may be a reflection across an axis of symmetry described by a line or plane.

Remark 3 (Stopping criterion, tolerance values)**.** The tolerance δ_1 is used as the cut-off for negligible weights and δ_2 for the stopping criterion. It is not necessary that these

two values be distinct, but it may be beneficial if you do not want to end the algorithm early with too large a δ_2 value nor keep unnecessary points in the optimal design with too small a δ_1 value. We recommend that $\delta_1 \leq 10^{-4}$, as any larger may result in no updates being made to $\Omega_{(m)}$.

We find the stopping criterion in line 7 very effective as it does not end the iterations early, nor does the number of iterations reach *max.iter* using it. It may not always be possible for the optimality condition alone to be met, i.e. $d.max \leq \delta_2$. For example, say we set $\delta_2 = 10^{-7}$. For some models on certain discrete sets Ω_N, $d.max$ only ever reaches a magnitude of 10^{-5}. This could be due to the discrete set Ω_N, or rounding error from floating point arithmetic. Hence, by taking the difference between the maxima from two consecutive iterations, we avoid this issue.

Remark 4 (Clustering schemes)**.** Occasionally, there is a numerical phenomena where the Φ-optimal design has two or more support points that are exceptionally close to one another. These clusters of points emerge if the difference in information between them is negligible, or when the true Φ-optimal support point is not in Ω_N. We may choose to collapse each cluster into one point for practicality.

We use the DBSCAN program from the Statistics and Machine Learning Toolbox (The MathWorks Inc., 2023b) in the clustering scheme (line 14) to identify clusters of support points. The clusters are then merged by taking their weighted average according to their design weights. If Ω is non-convex, the user should ensure that the new merged point is still a member of Ω. Alternatively, one may choose to prioritize the support point with the largest weight in the cluster. This is ideal if $\Omega_N = \Omega$. It is good practice to run CVX one last time to ensure that the clustered design is Φ-optimal.

4.2 Numerical examples

All algorithms and examples were coded in MATLAB R2023a and executed on a Dell Latitude 9430 laptop computer, with a 12th Gen Intel(R) Core(TM) i7-1265U (1.80 GHz). The following list of initializations is our default set-up of Algorithm 4:

- $\delta_1 = \delta_2 = 10^{-6}$
- $max.iter = 100$
- $\alpha = 0.5$
- $\Omega_{(0)}$ is comprised of uniformly random sampled points from Ω_N
- A reflection matrix T is used in the adding rule
- The optimal designs are clustered using weighted averages, via the DBSCAN program (see Remark 4)

Unless stated otherwise, these default rules apply to all the numerical examples below.

4.2.1 Kite

Wynn's polygon, as it's called, is the kite defined by the vertices $(-1,-1)$, $(-1,1)$, $(1,-1)$, and $(2,2)$ and scaled by a factor of $\sqrt{2}/4$. We define our design region as the boundary and interior points of this quadrilateral:

$$\Omega = \left\{ \mathbf{x} = (x_1, x_2)^\top \,\middle|\, x_1, x_2 \geq -\frac{\sqrt{2}}{4},\ x_1 \leq \frac{1}{3}(x_2 + \sqrt{2}),\ x_2 \leq \frac{1}{3}(x_1 + \sqrt{2}) \right\}.$$

This irregular region is studied in De Castro et al. (2019) and Duan et al. (2022). Consider a second-order LM with basis vector $\mathbf{z}(\mathbf{x}) = (1,\, x_1,\, x_1^2,\, x_1x_2,\, x_2,\, x_2^2)^\top$, and suppose we seek the D- and A-optimal designs for this model.

The kite is discretized using Algorithm 3; see Figure 4.1 below. We form a lattice of

247^2 points that cover $\overline{\Omega}$, so that the accepted points are $\approx 201^2 = 40,401$ in total. To be exact, the final number of included points is $N = 40,591$, of which 492 exist on the boundary. This includes the four characterizing vertices. Using a lattice is sufficient and no additional work is needed to set up Ω_N. The symmetry of the kite across the $x_1 = x_2$ line is maintained in Ω_N.

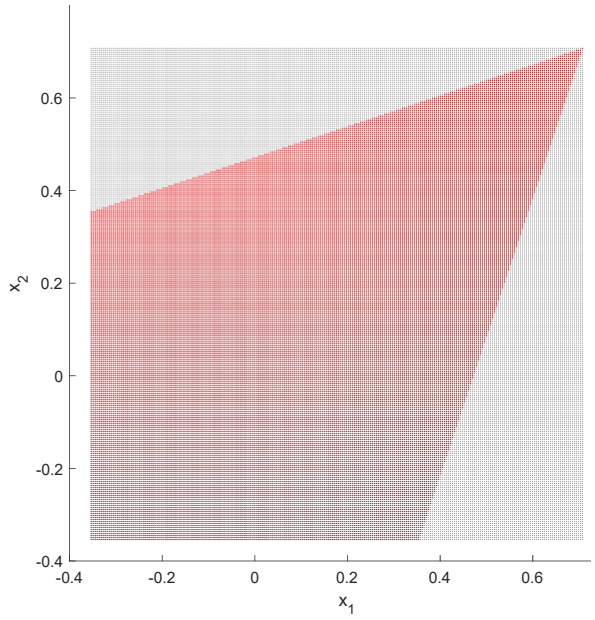


Figure 4.1: Rejection-sampling method to construct Ω_N for Wynn's polygon, with $N = 40,591$. Gray points are discarded, and the red points form Ω_N.

The initial sample size is $N_0 = 100$. The default set-up of Algorithm 4 is used with the exception of clustering, which was not necessary for these models. The reflection matrix $T = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ is used to reflect across $x_1 = x_2$ axis.

Results

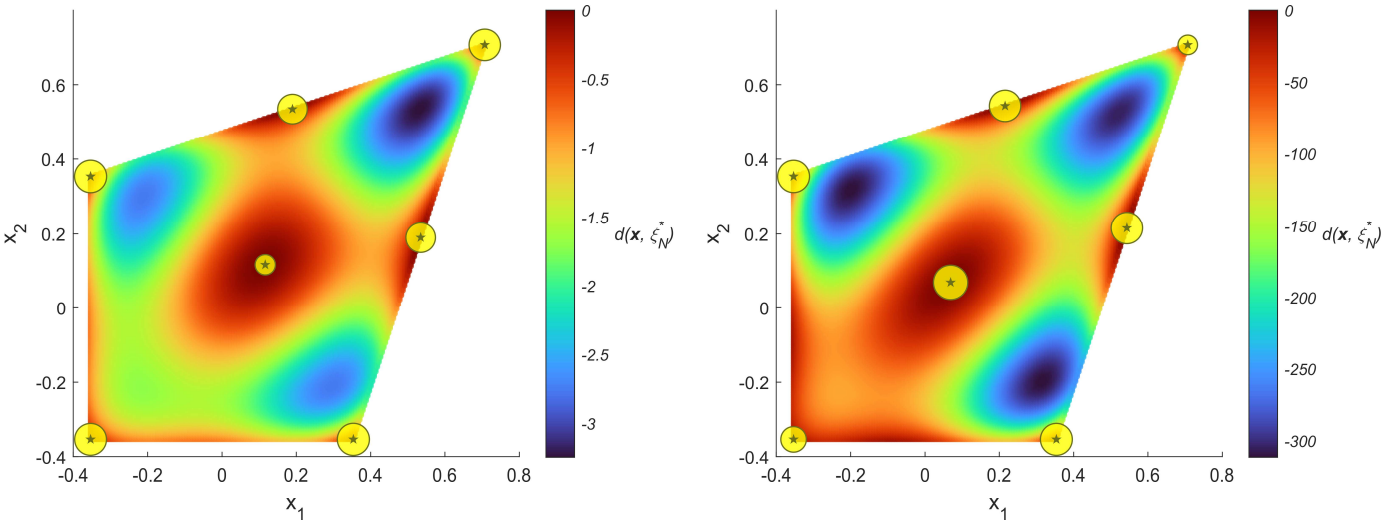
Table 4.1 presents our designs and Figure 4.2 plots the D- and A-optimal support points along with the optimality condition. As we can see, both optimal designs include seven support points. The points which are common to both designs are the four defining vertices of the polygon. These are listed in blue text in Table 4.1, but note that their weights differ per design. We evaluate the optimality conditions at the support points

D-optimal		A-optimal	
Support points $(x_1, x_2)^*$	Weights $\boldsymbol{w}^*$	Support points $(x_1, x_2)^*$	Weights $\boldsymbol{w}^*$
(-0.3536, -0.3536)	0.1627	(-0.3536, -0.3536)	0.1046
(-0.3536, 0.3536)	0.1654	(-0.3536, 0.3536)	0.1637
(0.3536, -0.3536)	0.1654	(0.3536, -0.3536)	0.1637
(0.7071, 0.7071)	0.1586	(0.7071, 0.7071)	0.0612
(0.1164, 0.1164)	0.0665	(0.0690, 0.0690)	0.1893
(0.1897, 0.5346)	0.1407	(0.2156, 0.5433)	0.1587
(0.5346, 0.1897)	0.1407	(0.5433, 0.2156)	0.1587
$-\{\det[\mathbf{I}(\xi_N)]\}^{1/6}$	-0.0553	$\mathrm{Tr}\{\mathbf{I}^{-1}(\xi_N^*)\}$	348.1304
range$(d_D(\mathbf{x}^*, \xi_N^*))$	$[-6.208, 8.166] \times 10^{-9}$	range$(d_A(\mathbf{x}^*, \xi_N^*))$	$[-1.665, 1.344] \times 10^{-6}$
run time (s)	2.0432	run time (s)	2.1653
iterations	5	iterations	5

Table 4.1: D- and A-optimal design for a second-order model on Wynn's polygon; generated using Algorithm 4 without clustering and $\alpha = 0.5$, $N_0 = 100$, $\delta_1 = \delta_2 = 10^{-6}$. The four defining vertices of the kite are in blue text.

and present the ranges. We can see that for both D- and A-optimality, $d_\Phi(\mathbf{x}^*, \xi_N^*)$ is very close to zero for all support points $\mathbf{x}^*$ in ξ_N^*. This demonstrates that the stopping criterion of Algorithm 4 is not premature.

For the D-optimal design of this model, the design in De Castro et al. (2019) is only 98.92% as efficient as ours, with an optimal function value of -0.0547. Furthermore, the D-optimal design in Duan et al. (2022) is only 97.65% as efficient, with an optimal function value of -0.054. Duan et al. (2022) also found an A-optimal design for this model with an average variance of 359.185 (96.92% efficiency compared to ours.) Thus, we have demonstrated that the iterative CVX algorithm performs well for this model and design region. The relative efficiencies are less than 1, indicating our optimal designs are better. Without including the T-reflected points in our adding rule, the



(a) D-optimal (b) A-optimal

Figure 4.2: Optimal design points (stars) and optimality conditions for the designs in Table 4.1. Sizing of the circles at the design points is proportional to their optimal weights.

algorithm takes 6-7 iterations to converge to the D- and A-optimal designs, respectively. With the standard adding rule, the algorithm takes 12-14 iterations, respectively. All three adding rules in the algorithm lead to the reported optimal designs in Table 4.1. Our algorithm essentially cuts the number of iterations in half. This is not a big influence on the run-times here. However, it is indicative of the drastic influence it can make for even larger N, which will be discussed in the examples below.

From Figure 4.2, we can compare the two optimal designs. Given that $d_\Phi(\mathbf{u}_j, \xi_N^*) \lessapprox 10^{-6}$ for all $\mathbf{u}_j \in \Omega_N$ and that equality is satisfied at the support points, we can conclude that the two designs are D- and A-optimal, respectively. It is interesting to note that the D-optimal weights are heavier on the boundary but the opposite is true for the A-optimal weights. Furthermore, both designs are symmetric across the $x_1 = x_2$ line. Had we not made sure that Ω_N maintained this natural axis of symmetry via the lattice, the optimal points and weights could not have had this property.

4.2.2 Folium

In this example, we consider a non-convex tri-folium design region in $\mathbb{R}^2$. This is a self-intersecting shape with a triple root at $(0, 0)$, defined in polar coordinates as

$$\Omega = \Big\{\mathbf{x} = (r\cos\theta, r\sin\theta)^\top \mid r \leq \cos\theta\big(3\sin^2\theta - 1\big),\ \theta \in [-\pi/2, \pi/2]\Big\}.$$

In Cartesian coordinates, this region is described as

$$\Omega = \Big\{\mathbf{x} = (x_1, x_2)^\top \;\Big|\; (x_1^2 + x_2^2)^2 + x_1(x_1^2 - 2x_2^2) \leq 0,\ \mathbf{x} \in \mathbb{R}^2\Big\}.$$

The equalities in $r \leq \cos\theta\big(3\sin^2\theta - 1\big)$ and $(x_1^2+x_2^2)^2+x_1(x_1^2-2x_2^2) \leq 0$ are attained on the boundary of Ω. We again use Algorithm 3 to form a lattice covering the minimum bounding box of Ω, $\overline{\Omega} \approx [-1, 0.5611] \times [0.3333, 0.5611]$. With $M = 351^2$, The number of accepted points is $N = 40,183 \approx 201^2$. However, only one of these points lies on the boundary of Ω and it is $(-1, 0)$. This is an example of when sampling the boundary explicitly is beneficial. We evaluate r at $3,000$ values of θ, equally-spaced in the interval $[-\pi/2, \pi/2]$ and convert the (r, θ) pairs to Cartesian coordinates. Thus Ω_N is comprised of $N = 43,182$ points in total. The discrete region Ω_N is symmetric about x_2. Figure 4.3 below presents a view of Ω_N around the intersecting point of the folium. Here, we can see that similarly to the arbelos example of Chapter 3, the lattice alone (indicated by the red points) does not cover the boundary of Ω. We would need to use a very fine-meshed lattice to approximate the boundary points (in black). By explicitly sampling the boundary, we avoid having to use a much larger candidate set in our optimization problem.

We seek the D- and I-optimal designs for a third-order LM with basis vector $\mathbf{z}(\mathbf{x}) = (1,\ x_1,\ x_1^2,\ x_1^3,\ x_2,\ x_2^2,\ x_2^3,\ x_1x_2,\ x_1^2x_2,\ x_1x_2^2)$. We set $N_0 = 100$ and use the default set-

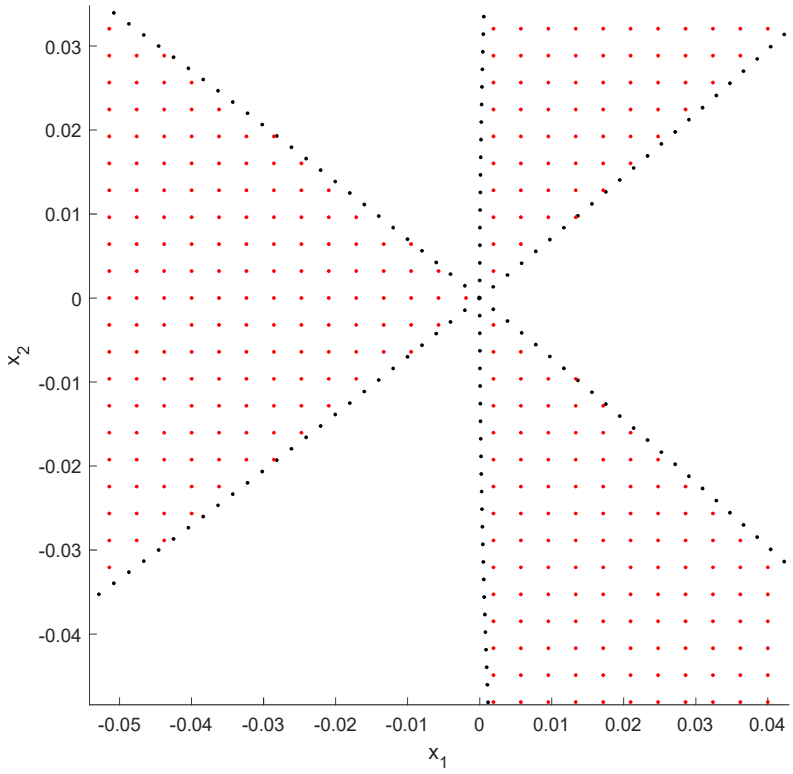


Figure 4.3: Zoomed-in view of Ω_N for a tri-folium region. Interior points (red) generated via Algorithm 3. Boundary points (black) are sampled independently using polar coordinates.

up for the algorithm. Points satisfying the adding rule in the m^{th} iteration are reflected across $x_2 = 0$ using $T = \left[\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right]$. For the I-optimal design, we estimate matrix $\mathbf{A}$ using Algorithm 2 with $1,000,000$ semi-random points as follows. To preserve the symmetry of the region, $500,000$ randomly sampled points in the positive x_2 half-plane were

reflected across the $x_2 = 0$ line. Thus, $\mathbf{A}$ is found as

$$\mathbf{A} \approx \begin{bmatrix} 1.0000 & -0.2754 & 0.2355 & -0.1556 & 0 & 0.0508 & 0 & 0 & 0 & 0.0054 \\ -0.2754 & 0.2355 & -0.1556 & 0.1226 & 0 & 0.0055 & 0 & 0 & 0 & 0.0045 \\ 0.2355 & -0.1556 & 0.1226 & -0.0966 & 0 & 0.0045 & 0 & 0 & 0 & -0.0011 \\ -0.1556 & 0.1226 & -0.0966 & 0.0794 & 0 & -0.0011 & 0 & 0 & 0 & 0.0013 \\ 0 & 0 & 0 & 0 & 0.0508 & 0 & 0.0078 & 0.0054 & 0.0045 & 0 \\ 0.0508 & 0.0054 & 0.0045 & -0.0011 & 0 & 0.0078 & 0 & 0 & 0 & 0.0016 \\ 0 & 0 & 0 & 0 & 0.0078 & 0 & 0.0016 & 0.0016 & 0.0005 & 0 \\ 0 & 0 & 0 & 0 & 0.0054 & 0 & 0.0016 & 0.0045 & -0.0011 & 0 \\ 0 & 0 & 0 & 0 & 0.0045 & 0 & 0.0005 & -0.0011 & 0.0013 & 0 \\ 0.0054 & 0.0045 & -0.0011 & 0.0013 & 0 & 0.0016 & 0 & 0 & 0 & 0.0005 \end{bmatrix}. \quad (4.1)$$

Results

De Castro et al. (2019) report a D-optimal design with $\Phi_D(\xi_N^*) = -0.0093$, which matches the efficiency of our design presented in Table 4.2. Figure 4.4 below presents the optimality condition function for both designs. Notice that our I-optimal design is also symmetric. If the approximation error in A was large enough that the '0' entries were instead small values, the I-optimal design would not be symmetric in x_2.

For both objective functions, our relaxed adding rule improves the convergence time of the iterative procedure compared to the standard one. Updating $\Omega_{(m)}$ one point at a time takes 65 iterations (≈ 54 seconds) and 67 iterations (≈ 57 seconds) for D- and I-optimal designs, respectively. Both designs found under the standard rule attain the same objective values as in Table 4.2.

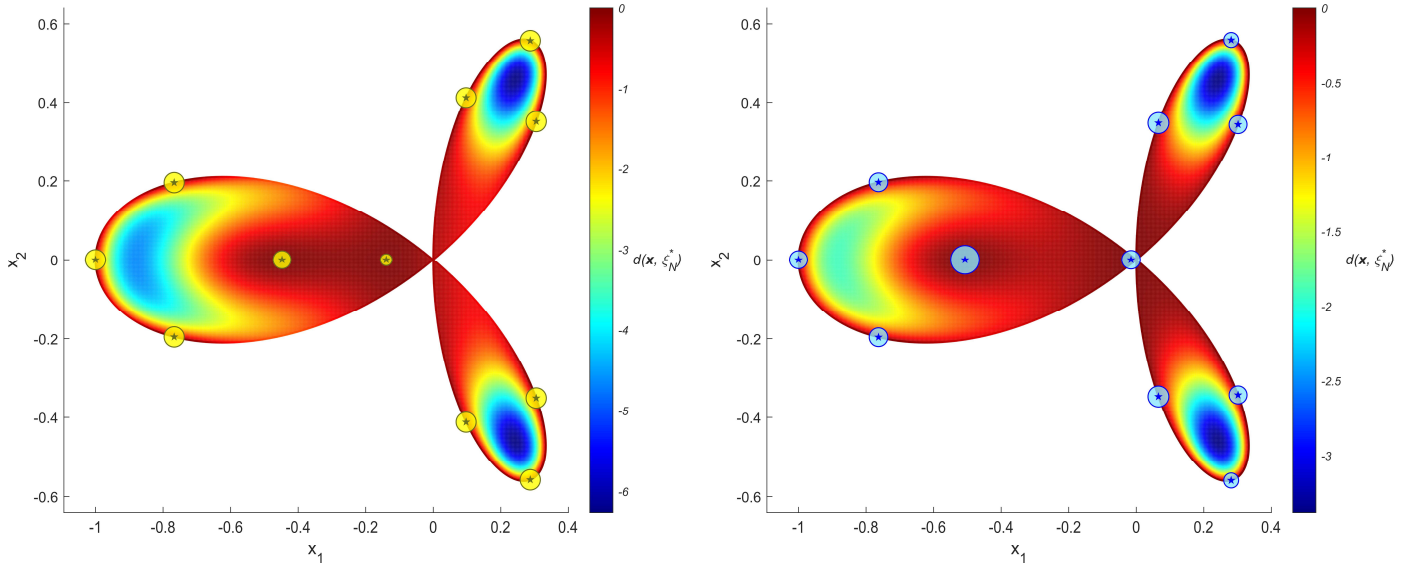
We can see that the optimality condition is satisfied for both designs, as d_Φ attains near-zero values at the support points of ξ_N^*. Without a collapsing scheme, there are 34 D-optimal points and 35 I-optimal points for this model. Both designs can be collapsed into 11 support points, as in Table 4.2, with unchanged efficiency. Hence, the D- and I-optimal designs for this model and region are not unique. This is common

D-optimal		I-optimal	
Support points $(x_1, x_2)^*$	Weights $\boldsymbol{w}^*$	Support points $(x_1, x_2)^*$	Weights $\boldsymbol{w}^*$
(-1, 0)	0.0995	(-1, 0)	0.0778
(-0.7670, $\pm$0.1956)	0.0985	(-0.7631, $\pm$0.1964)	0.0837
(-0.4476, 0)	0.0773	(-0.5070, 0)	0.1978
(-0.1390, 0)	0.0332	(-0.0154, 0)	0.0830
(0.0974, $\pm$0.4121)	0.0975	(0.0658, $\pm$0.3469)	0.1056
(0.2874, $\pm$0.5581)	0.0992	(0.2811, $\pm$0.5598)	0.0536
(0.3049, $\pm$0.3507)	0.0998	(0.3016, $\pm$0.3429)	0.0778
$-\{\det[\mathbf{I}(\xi_N)]\}^{1/10}$	-0.0093	$\mathrm{Tr}\{\mathbf{A}\mathbf{I}^{-1}(\xi_N^*)\}$	6.3786
# of unmerged pts	34	# of unmerged pts	35
range$(d_D(\mathbf{x}^*, \xi_N^*))$	$[-7.178, 6.540] \times 10^{-8}$	range$(d_I(\mathbf{x}^*, \xi_N^*))$	$[-9.919, 7.125] \times 10^{-9}$
run time (s)	27.5582	run time (s)	23.1106
iterations	11	iterations	20

Table 4.2: D- and I-optimal designs for a third-order model on a tri-folium; Generated using Algorithm 4 with clustering and $\alpha = 0.5$, $N_0 = 100$, $\delta_1 = \delta_2 = 10^{-6}$

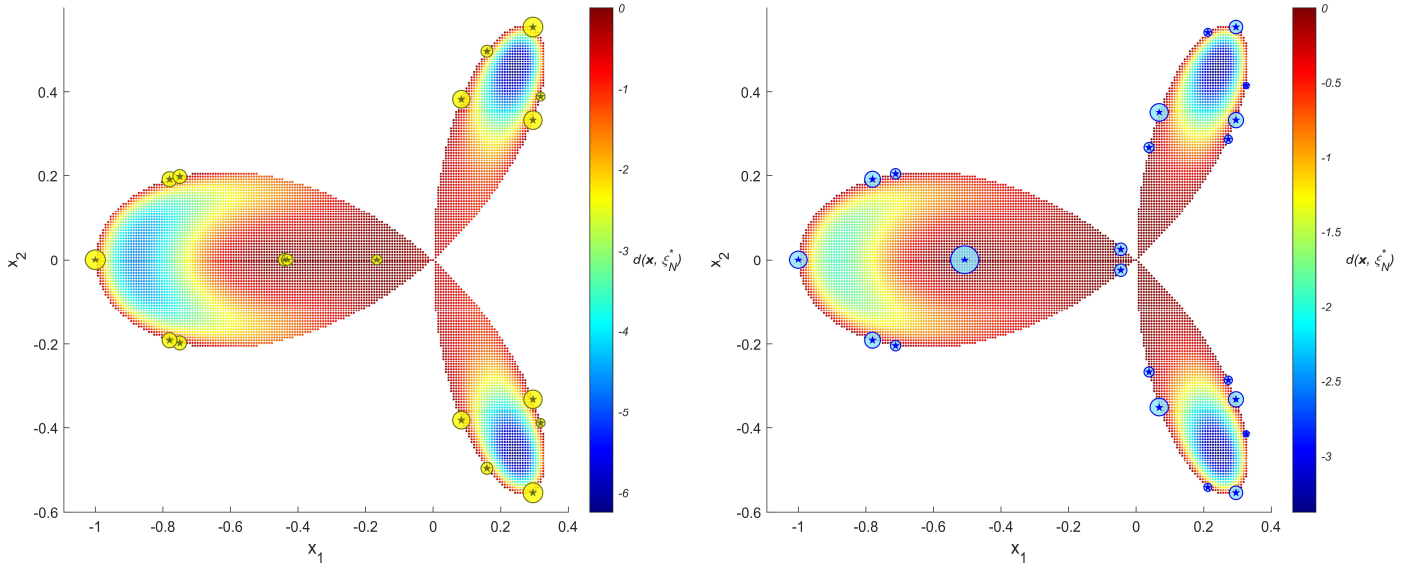
for many optimal designs on elliptically-shaped regions. When the surface of $d_\Phi(\mathbf{x}, \xi_N^*)$ does not have strict maxima, the adding and deleting rules will influence the number of (unmerged) support points. If we increase δ_2 from 10^{-6} to 10^{-4} and keep $\delta_1 = 10^{-6}$, for example, the deletion rule of the algorithm is stricter. This results in D- and I-optimal designs with 19 and 25 unmerged support points, respectively. The initial design points, $\Omega_{(0)}$, may also affect the final optimal design if it is not unique.

To further demonstrate the significance of adequate discretization and iterative procedures, we use Algorithm 1 to compute the D- and I-optimal designs for this model. To do so, we require a smaller Ω_N. We do not sample the boundary, and our lattice is generated via rejection-sampling such that $N = 10,127 \approx 101^2$ points are



(a) D-optimal (b) I-optimal

Figure 4.4: D- and I-optimal design points and optimality conditions for the third-order LMs in Table 4.2. The sizes of the circles are proportional to their weights.



(a) D-optimal (b) I-optimal

Figure 4.5: D- and I-optimal results of the Folium example with a smaller candidate set ($N = 10,127$); generated via Algorithm 1. Circles denote the support points and the sizes of the circles are proportional to their weights.

in the design region. The $\mathbf{A}$ matrix in (4.1) is used for this small I-optimal design problem. Figure 4.5 presents the results. It is clear to see the shortcomings of using a regular lattice here. The roundest parts of the tri-folium are not sampled, so Ω_N looks "clipped" at the edges. Furthermore, the larger patches of white-space between the points highlight the difference in sample size from the original candidate set in Figure 4.4.

For both cases with Algorithm 1, the run time is around 132 seconds. The small I-optimal design achieves an objective value of 6.4027 with 22 support points and the small D-optimal design an objective of -0.0092 with 18 support points. The optimal designs in Table 4.2 are clearly better. Thus, Algorithm 4 is more effective than Algorithm 1. Notice in Figure 4.5 that the optimal designs on the small candidate set are still symmetric about x_2.

4.2.3 Cube

Our final example in this chapter focuses on a real-world logistic regression model studied in Li and Deng (2021) and Woods et al. (2006) about potato packing. The aim of the experiment was to investigate the use of protected atmosphere packing to give potatoes a shelf-life of one week. The three predictors are the vitamin concentration in the pre-packaging dip and the levels of two gases in the packing atmosphere. Let x_1, x_2, and x_3 denote these variables, respectively, standardized such that $\Omega = [-1, 1]^3$. Several binary responses (denoted y) were recorded, one being the presence or absence of liquid in the pack after 7 days. A second-order logistic model is considered, with basis vector $\mathbf{z}(\mathbf{x}) = (1,\, x_2,\, x_3,\, x_2x_3,\, x_1^2,\, x_2^2,\, x_3^2)^\top$.

The design region Ω is regular so a regular lattice will be sufficient in covering it. So long as we choose an odd number of points along each coordinate axis, the symmetry

in all three variables will be preserved. We let Ω_N be a regular lattice of $N = 101^3$ points in $[-1, 1]^3$.

We aim to find the I-optimal design for this model. Recall that the information matrix for the logistic model takes the form

$$\mathbf{I}(\xi_N, \hat{\boldsymbol{\beta}}) = \sum_{i=1}^{N} w_i \left[\frac{e^{\hat{\eta}_i}}{(1 + e^{\hat{\eta}_i})^2} \right] \mathbf{z}(\mathbf{u}_i)\mathbf{z}(\mathbf{u}_i)^\top,$$

where $\hat{\boldsymbol{\beta}} = (-2.93, -0.52, -0.79, -0.66, 0.94, 0.79, 1.82)^\top$ is the estimated parameter vector in Woods et al. (2006). Algorithm 2 is used with 151^3 randomly sampled points in Ω to numerically compute $\mathbf{A}$:

$$\begin{aligned}
\mathbf{A} &= \int_\Omega \mathbf{z}(\mathbf{x})^\top \mathbf{z}(\mathbf{x}) \left(\frac{\partial h^{-1}}{\partial \eta} \right)^2 dF_{\mathsf{UNIF}}(\mathbf{x}) \\
&\approx 10^{-2} \times \begin{bmatrix}
2.092 & -0.342 & -0.575 & -0.142 & 0.842 & 0.846 & 1.051 \\
-0.342 & 0.846 & -0.142 & -0.180 & -0.134 & -0.218 & -0.135 \\
-0.575 & -0.142 & 1.051 & -0.135 & -0.194 & -0.180 & -0.360 \\
-0.142 & -0.180 & -0.135 & 0.400 & -0.052 & -0.088 & -0.093 \\
0.842 & -0.134 & -0.194 & -0.052 & 0.543 & 0.331 & 0.397 \\
0.846 & -0.218 & -0.180 & -0.088 & 0.331 & 0.546 & 0.400 \\
1.051 & -0.135 & -0.360 & -0.093 & 0.397 & 0.400 & 0.718
\end{bmatrix}.
\end{aligned}$$

The initial sample size is $N_0 = 1000$ and we reflect the points which meet the adding rule across the $x_1 = 0$ plane using $T = \left[\begin{smallmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{smallmatrix} \right]$. Although Ω is symmetric along all the coordinate axes, the basis vector $\mathbf{z}(\mathbf{x})$ seems to exhibit a symmetry across this plane. Additionally, the only change to the default algorithm set-up is that the cut-off for importance is changed to $\delta_1 = 10^{-4}$ while the stopping tolerance remains $\delta_2 = 10^{-6}$.

Results

The I-optimal design is presented in Table 4.3. Without clustering, the I-optimal design

contains 24 points. The point $(0, -0.0008, -1)$ is the weighted average of two nearby points: $(0, -0.02, -1)$ with weight 0.0042 and $(0, 0, -1)$ with weight 0.1025. Our design compared with that in Li and Deng (2021) has a relative efficiency of less than 1. This indicates that our design is better. Furthermore, there is indeed a reflection-symmetry across $x_1 = 0$ plane. The stopping criterion was appropriate, as the support points in Table 4.3 satisfy the numerical I-optimality condition.

Support points			Weights
x_1^*	x_2^*	x_3^*	$\boldsymbol{w}^*$
± 1	-1	-1	0.0226
± 1	-1	0.02	0.0257
± 1	-1	1	0.0265
± 1	-0.14	1	0.0083
± 1	-0.04	-0.42	0.0830
± 1	0	-1	0.0116
± 1	1	-0.94	0.0280
± 1	1	1	0.0316
0	-1	-1	0.0650
0	-1	-0.06	0.1076
0	-1	1	0.0412
0	-0.36	1	0.0833
0	-0.0008	-1	0.1068
0	1	-1	0.0357
0	1	-0.6	0.0859
		$\mathrm{Tr}\{\mathbf{AI}^{-1}(\xi_N^*)\}$	0.5046
		$\mathrm{range}(d_I(\mathbf{x}^*, \xi_N^*))$	$[-2.521, 2.386] \times 10^{-10}$
		Rel. Eff.	0.9424
		run time (s)	122.9740
		iterations	15

Table 4.3: I-optimal design for a logistic model in 3D; Generated via Algorithm 4.

Figure 4.6 shows the optimality condition function for some cross-sections of Ω_N. Five equally spaced cross-sections are taken along the x_1 axis to showcase the symmetry of the design. Notice that the optimality condition is satisfied since all the design points are on the darkest red parts of the cross-sections. Furthermore, the locations and sizes of the design points are identical across the $x_1 = 0$ cross-section. That is, the slices are $x_1 = 1$ and $x_1 = -1$ are identical, and so are the ones at $x_1 = 0.5$ and $x_1 = -0.5$.

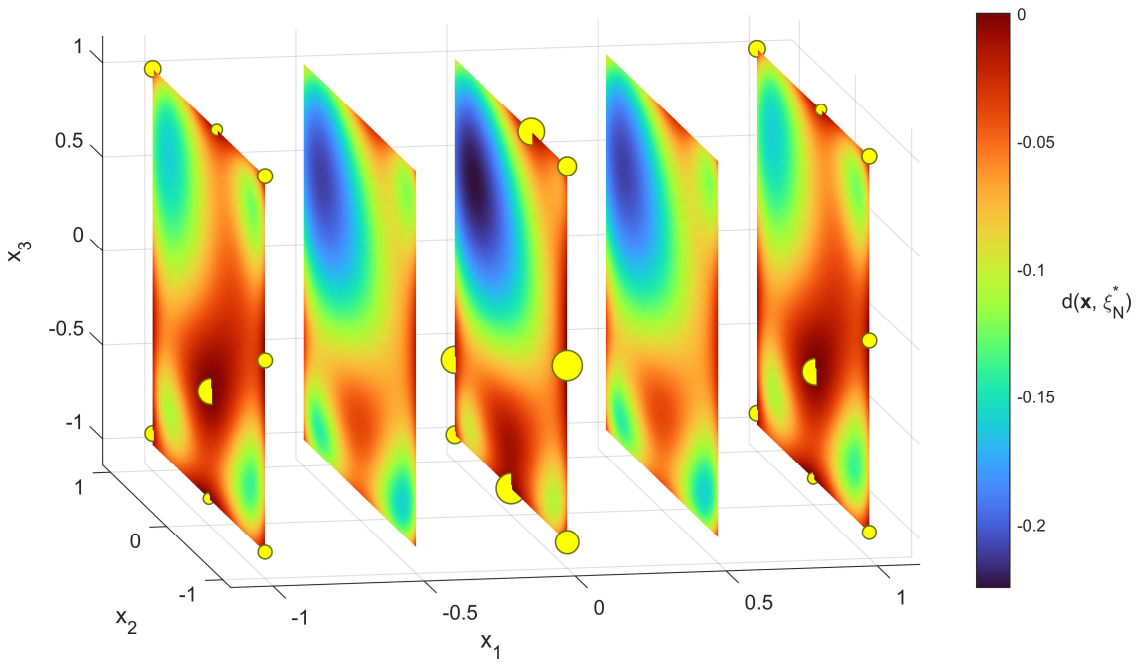


Figure 4.6: I-optimal design points (yellow circles) and cross-sections of the optimality condition function at $x_1 = \{0, \pm 0.5, \pm 1\}$ for the design presented in Table 4.3; Generated via Algorithm 4. Sizes of circles are proportional to the weights.

Setting $\alpha = 1$ and using the standard adding rule, the algorithm takes 38 iterations to converge and a run time of ≈ 291 seconds. Our relaxed adding rule takes less than half of this time to converge to the same design. Hence, our algorithm's updating procedure is better than the standard.

4.3 Summary

The numerical examples demonstrate various aspects of Algorithm 4. Firstly, it is better than Algorithm 1. Because the iterative procedure can handle large candidate sets, we can use a large N value to construct Ω_N to cover Ω well. This leads to more precise optimal designs. The way in which we discretize Ω can help reduce N, as well, using the guidelines from Chapter 3. Additionally, Algorithm 4 is faster because it uses CVX iteratively with small subsets of Ω_N at a time.

Secondly, there are benefits to clustering. Consider the case when the true Φ-optimal support points are not in Ω_N. Usually, two or more candidate points nearest to them are identified as support points. Taking weighted averages of these clusters can give a better approximation of the true support points to include in the final optimal design. This is seen in the logistic cube example. Also, in the case where $d_\Phi(\mathbf{x}, \xi_N^*)$ has non-strict maxima, several adjacent candidate points at those maxima may be included in the optimal design. Clustering by weighted average will collapse them and reduce the number of support points overall. This is seen in the folium example.

Finally, the neighbourhood size $\alpha < 1$ as well as the use of a transformation matrix T in the adding rule improves the convergence speed of the algorithm, compared to the adding rule typically used in the literature. It can affect the size of the optimal design, i.e. the number of support points, when the design is not unique. We recommend trying different values of α in implementation, as each model may vary. From our experience, $\alpha = 0.5$ is a great value to start with.

The MATLAB code for this chapter is presented in Section A.4 of the Appendix, including a user-defined function implementing Algorithm 4.

Chapter 5

Conclusion

This thesis has demonstrated that CVX is an excellent tool for solving convex optimal design problems on discrete design spaces. The reflection-symmetry and scale-invariance properties were derived for the EI-optimality criterion. Then, CVX was used to solve for EI-optimal designs, which were better than those presented in the literature and satisfied said properties.

The relationship between optimal designs on discrete and continuous design regions was investigated. Optimal designs found on discrete spaces were assessed for their optimality in the original continuous design space. An upper bound on the difference in optimality was derived for the c-, D-, A-, and I-optimality criteria. We give useful guidelines to discretize continuous design spaces, in particular when design spaces are irregular, there are several design variables, or the optimal designs have certain features such as symmetric properties.

We presented an iterative CVX-based algorithm that can handle large candidate sets. Our innovation included finding efficient ways to update the discrete design space in each iteration and clustering the support points in optimal designs at the end of iteration. Several applications were presented to illustrate discretizing irregular design

spaces as well as applying the numerical method for finding optimal designs for linear and generalized linear models. The resulting optimal designs were better than those found in the literature.

Future research problems include improving the upper bound in Theorem 3.2, investigating the neighbourhood size α in more detail, implementing and comparing different clustering schemes, and providing systematic methods of discretizing design regions. Algorithms for more complicated design problems may be developed as well, such as designs with multi-objective optimality criteria, or multivariate regression models.

Appendix

A.1 Installing CVX in MATLAB

Here are the steps to installing the CVX program.

1. Go to webpage `cvxr.com/cvx/download/`. Click on the correct version of CVX for your computer. A zip file will be downloaded. Save it in a file folder.

2. Extract the contents of the zip file.

3. Start MATLAB. Change the working directory in MATLAB to that which contains the CVX file and run the script titled `cvx_setup.m` (you can right-click on it from the left-side view and select 'Run' from the drop-down menu). The output in the command window should indicate if there were any errors in the set up. If there are no errors, then you may use CVX properly.

A.1.1 Add-on toolboxes in MATLAB

For the code developed in this thesis, we rely on two add-ons in MATLAB. The first is the Deep Learning toolbox (The MathWorks Inc., 2023a), from which we use the `combvec(...)` function. The second is the Statistics and Machine Learning toolbox

(The MathWorks Inc., 2023b) to use the `dbscan(...)` function. Here are the steps to installing toolboxes in MATLAB.

1. Open MATLAB and select the 'Home' tab in the toolstrip at the top of the page.
2. Click on the 'Add-ons' button in the Environment section of the toolstrip. Select the 'Get Add-ons' item from the drop-down menu.
3. A new window will open for the Add-on Explorer. Search the name of the desired toolbox in the top-right search bar.
4. Click the 'install' button. The add-on will be installed and any function from it may be used thereafter.

A.1.2 MATLAB File Exchange

The plot in Figure 2.6 is called a ternary plot and was made using the Ternary Plots program submitted on the MATLAB Central File Exchange (Theune, 2005). To download any submission from the file exchange, click the 'Download' button on the right-side of the program page. A zip file will be downloaded. Extract the files into your working directory and use as desired.

A.2 Selected code from Chapter 2

We present the complete code for the linear model example. For the remaining two examples, we only present the relevant set-up and matrix computations, as the simple CVX algorithm is unchanged from the linear model example.

A.2.1 LM example

We set up the design region Ω_N as vector `x` and initialize the elementary matrices, `Zi`, using the basis vector $\mathbf{z}(\mathbf{x})$. The $\mathbf{A}$ matrices using F_{UNIF} and F_{ASIN} are saved into a 3D matrix, `AA`.

```
%% design region set-up
x1=linspace(-1, 1, 101);  %setting up the two variables
x2=linspace(0, 1, 101);
x = combvec(x1,x2); % 2 by N matrix, each column is a design point
N=length(x);

%% model and Info matrix set-up
z = @(x) [1 x(1) x(1)^2 x(2) x(1)*x(2)]';  %basis vector
q = 5; % dim(z(x))

Zi = arrayfun(@(i) z(x(:,i))*z(x(:,i))', 1:N, "UniformOutput", false);
Zi = reshape([Zi{:}],q,q,[]); %(qxq)xN elementary matrices for Info matrix

AA = zeros(q,q,2);
%A matrix - using F(x) = uniform dist.
AA(:,:,1) = [1 0 1/3 1/2 0; 0 1/3 0 0 1/6;
   1/3 0 1/5 1/6 0; 1/2 0 1/6 1/3 0; 0 1/6 0 0 1/9];

%A matrix - using arcsine dist.
AA(:,:,2) = [1 0 1/2 1/2 0; 0 1/2 0 0 1/4;
    1/2 0 3/8 1/4 0; 1/2 0 1/4 3/8 0; 0 1/4 0 0 3/16];
```

The tolerance value δ is saved into the variable `tol`. Lines 30-43 describe the simple CVX algorithm. Lines 50-52 are where we compute $d_{EI}(\mathbf{u}_j, \xi_N^*)$ for all $\mathbf{u}_j \in \Omega_N$, and save the vector of values into `d_ei`. Main results are stored in the cell, `outcvx`.

```
outcvx = cell(2,6); %store our results
tol = 1e-7; %set the delta value
for m=1:2
    A = AA(:,:,m);
    A2= sqrtm(A);
    A2inv=inv(A2); %needed for CVX objective function

    tic
    cvx_begin quiet
        cvx_precision high
        variable w(N) nonnegative; %we want optimal weights
        expression I(q,q); %Info matrix is an "expression" of the weights
```

```
        for j=1:N       %computing the Info matrix
           I = I + Zi(:,:,j)*w(j);
        end

        AI=A2inv*I*(A2inv)';
        AI = 0.5*(AI + AI'); % re-casting AI to be a symmetric matrix that CVX
           will recognize.
        minimize (trace_inv(AI)); % equiv to trace(A*inv(I))
        sum(w) == 1;
    cvx_end

    runtime=toc;
    design = sortrows([w(w>tol), x(:,w>tol)'], 2:3);

    d_ei = zeros(1,N);
    invI = inv(I);
    for ii=1:N %optimality condition
        d_ei(ii) = trace(A2*(I\Zi(:,:,ii)/I - invI)*A2');
    end

    outcvx{m,1} = runtime; %run time
    outcvx{m,2} = cvx_optval; %objective function value
    outcvx{m,3} = cvx_status; %status (solved, failed, etc) of CVX solver
    outcvx{m,4} = design; %optimal design
    outcvx{m,5} = I; %information matrix
    outcvx{m,6} = d_ei; %optimality condition function
end
```

Finally, Figures 2.2 and 2.3 are plotted. Note that the custom colours `myyellow`, `myblue`, `mygray` are RGB colour codes, and are used throughout the code presented in this Appendix.

```
%% PLOTTING
%setting up some custom colours
myyellow = [108 114 27]/255; %myyellow
myblue = [137 232 255]/255; %myblue
mygray = [155 155 155]/255; %mygray
f = figure(1);
tiledlayout(1,2,"Padding","loose","TileSpacing","loose")
colormap(parula);
    for i=1:2
     nexttile(i);
     h = reshape(outcvx{i,6}, S+1, S+1)';
     design = outcvx{i,4};
     surf(x1, x2, h, 'Edgecolor','none'); hold on
     scatter3(design(:,2), design(:,3), zeros(1,length(design)),...,
         design(:,1)*500,'k','LineWidth',0.75, 'MarkerFaceColor','y',...
         'MarkerEdgeColor',myyellow, 'Marker','o','MarkerFaceAlpha',0.5);
     scatter3(design(:,2), design(:,3), zeros(1, length(design)), 10,...
              'filled','LineWidth',1,'Marker','pentagram',...
```

```
                'MarkerFaceColor',myyellow,'MarkerEdgeColor',myyellow,);
     hold off
     xlabel("x_1", "FontSize",12); ylabel("x_2", "FontSize",12);
     zlabel("d({\bf{x}}, \xi^*_N)")
     axis padded square
    end
linkaxes

points_unif= [0.944,0.460; % design points from Li & Deng (2021)
        -0.340,0.410; -0.1360,0.7580; 0.4920,0.8860; 0.568,0.098;
        -1,1;-1,0;1,1;1,0; -0.064,0; 0.0920,1; -0.108,1; 0.0880,0];

points_arcsi = [0.348,0.93; % design points from Li & Deng (2021)
    0.096,0.416;-0.272,0.69; 1,0;-1,0;
    -1,1;1,1; -0.052,0;-0.116,1; 0.108,1;0.008,1];

figure(2)
scatter(design(:,2), design(:,3), 'red', 'filled','^',...
    'MarkerEdgeColor','none','MarkerFaceAlpha',0.5, 'SizeData',150); hold on;
scatter(points_unif(:,1), points_unif(:,2), 'b', 'pentagram', ...
    LineWidth=1, SizeData=100);
scatter(points_arcsi(:,1), points_arcsi(:,2), 'go', 'filled',...
    LineWidth=1, SizeData=50, MarkerEdgeColor='k', MarkerEdgeAlpha=0.5, ...
    MarkerFaceAlpha=0.75);
legend(["CVX Alg.", "Uniform", "Arcsine"], "Location","southwest","FontSize
   ",10)
hold off
```

A.2.2 GLM example

The design region and regression model are set-up.

```
%design region set up
S=101; % # of discretized points per dimension
r = linspace(-1,1,S);
x = combvec(r, r); N=length(x);

%model set-up
beta = [2 1 -2.5];
z = @(x) [1 x]';       %bases for predictors
v = @(x) exp(beta*[1 x]')/(1+exp(beta*[1 x]'))^2;
        %v(x) is used in both the Info matrix and the A matrix

q = length(z(x(:,1)'));
Zi = zeros(q,q,N); %for information matrix

for i=1:N
    Zi(:,:,i) = z(x(:,i)')*z(x(:,i)')'*v(x(:,i)');
end
```

Then, we implement Algorithm 2 to compute $\mathbf{A}$ numerically.

```
% Numerically computing 'A' matrix
n=1000000;
seed=1234; rng(seed);
a=0; b=1; %the design interval [0,1]x[0,1]
G = @(x) z(x)*z(x)'*v(x)^2; % integrand of the 'A' matrix (area=1)
xy = a + (b-a).*rand(n, 2);
A0=0;
for jj=1:n
    A0=A0+G(xy(jj,:));
end
A = (A0/n); %area=1

A2= sqrtm(A);
A2inv=inv(A2); %needed for CVX objective function
```

The simple CVX algorithm and the optimality condition code is the same as in A.2.1.

A.2.3 Mixture models

The variable `DD` holds the number of variables, which in the script below is set to three. Note that the basis vector is saved into `g`, and the special cubic and second-order basis functions are defined respectively by `ord3sp(x)` and `ord2f(x)` at the end of the script. The $\mathbf{A}$ matrix is generated in lines 25-29. The optimality condition function is checked at 1000 random points on the simplex in lines 54-61. Figure 2.6 is plotted in lines 63-71.

```
%% Mixture model examples

DD=3; %number of ingredients
tol = 1e-7; %for stopping criterion
g = @(x) ord3sp(x(:)')'; %see functions below. special cubic
%g = @(x) ord2f(x(:)')'; %scheffe second order

q = length(g(ones(DD,1))); %for info matrix, size is k*k
GI = @(x) g(x)*g(x)'; %for info matrix

V= eye(DD);
t = 2; xy = V; %start with the pure components
```

```
while t <= DD %generate the {DD, DD} simplex-centroid points
    t1 = perms([repelem(1/t, t), repelem(0, DD-t)]);
    xy = union(xy, t1, "rows","sorted");
    t = t+1;
end
x = xy';
N = length(x);

vol = 1/factorial(DD-1); %this is the volume formula, per Goos et. al. (2016)
% each entry in A matrix is equal to:
% prod_i=1^DD(factorial(p_i))/factorial(DD-1 + sum_i=1^DD{p_i})
% where p_i is the exponent of the i^th variable in the term
temp1 = arrayfun(@(i) log(abs(GI(1-0.5*V(i,:))))/log(0.5), 1:DD,...
    'UniformOutput', false);
temp2 = reshape([temp1{1:end}],q,q,[]);
A0 = prod(factorial(temp2),3)./factorial(DD-1 + sum(temp2,3));
A= A0/vol;

A2= sqrtm(A);
A2inv=inv(A2); %needed for CVX objective function

tic
cvx_begin quiet
    cvx_precision high
    variable w1(N) nonnegative; %we want optimal weights
    expression I(q,q); %Info matrix is an "expression" of the weights

    for j=1:N       %computing the Info matrix
        I = I + GI(x(:,j)')*w1(j);
    end

    AI=A2inv*I*(A2inv)';
    AI = 0.5*(AI + AI'); % symmetric matrix that CVX will recognize.
    minimize (trace_inv(AI)); % equiv to trace(A*inv(I))
    sum(w1) == 1;
cvx_end

runtime=toc;
design = sortrows([w1(w1>tol), x(:,w1>tol)'], 2:DD+1);

rng(12345) %set seed
N1=1000;
xi = exprnd(1, N1-length(xy), DD);
int_pts = union(xi./sum(xi, 2), x', 'rows', 'sorted')';
d_ei = zeros(1,N1);
invI = inv(I);
for ii=1:N1
    d_ei(ii) = trace(A2*(I\GI(int_pts(:,ii))/I - invI)*A2');
end

if DD==3
    figure(2)
    colormap(turbo);
    tersurf(int_pts(1,:), int_pts(2,:), int_pts(3,:), d_ei); hold on
```

```
    h = ternaryc(design(:,2), design(:,3), design(:,4));
    arrayfun(@(i) set(h(i), 'SizeData', design(i,1)*1000), 1:length(design));
    set(h, 'MarkerFaceColor', 'y', 'Marker','o', 'MarkerEdgeColor',myyellow, '
       LineWidth',1, 'MarkerFaceAlpha',0.5);
    hlabels=terlabel('x_1', 'x_2', 'x_3');
end

%% functions for mixture models
function g = ord2f(x)
% Make a 2nd order model, x is a row vector
    g=x; p=length(x);
    for i=1:(p-1)
        g = [g (x(i))*(x(i+1:p))];
    end
end

function g=ord3sp(x)
%make a special cubic mixture model, x is row vector
    p=length(x);
    g = ord2f(x);
    for i=1:(p-2)
        for j=(i+1):(p-1)
            g = [g x(i)*x(j)*x(j+1:p)];
        end
    end
end
```

A.3 Selected code from Chapter 3

The two sampling methods for the arbelos design region are presented.

```
%% arbelos

%%%%%% design region set-up %%%%%%
%%with boundary sampling:
nt=1001;
theta = flip(linspace(0, pi, nt));
t1 = (linspace(0, pi, 0.6*nt));
t2 = (linspace(0, pi, 0.4*nt));
b1 = [cos(theta); sin(theta)]; %outside
b2 = [0.6*cos(t1)+0.4, 0.4*cos(t2)-0.6;
    0.6*sin(t1), 0.4*sin(t2)]; %inside
V = union(b1', b2', 'rows', 'stable')';

n=ceil(80^2*2/0.7540);
xx = combvec(linspace(-1, 1,ceil(2*sqrt(n/2)) ), linspace(0, 1, ceil(sqrt(n/2))
    ) );
```

```
[in, on] = inpolygon(xx(1,:), xx(2,:), V(1,:), V(2,:));
x =union(xx(:,in)', V', 'rows')';

%%just regular lattice:
n=ceil(101^2*2/0.7540);
xx = combvec(linspace(-1, 1,ceil(2*sqrt(n/2))), linspace(0, 1, ceil(sqrt(n/2)))
    );
[in, on] = inpolygon(xx(1,:), xx(2,:), V(1,:), V(2,:));
x=xx(:,in);
```

The model is set-up and the CVX solver is used to find the D-optimal design. Optimality condition is checked at the end of the script.

```
%model set-up
g =  @(x) [1 x(1) x(1)^2 x(2) x(2)^2 x(1)*x(2)]';
q = 6; %dimensions of info matrix, k*k
beta=ones(1,q);
v = @(x) exp(beta*g(x));%/(1+exp(beta*g(x)))^2;
GI = @(x) g(x)*g(x)'*v(x); %for info matrix

N=length(x); %candidate set size
tol=1e-6; %delta value

tic
cvx_begin
    cvx_precision high
    variable w(N) nonnegative; %we want optimal weights
    expression I(q,q); %Info matrix is an "expression" of the weights

    for j=1:N       %computing the Info matrix
       I = I + GI(x(:,j))*w(j);
    end

    minimize -det_rootn(I);
    sum(w) == 1;
cvx_end
runtime=toc;

design = sortrows([w(w>tol), x(:,w>tol)'], 2:3); %D-optimal design

d_phi = zeros(1,N);
invI = inv(I);
for ii=1:N
    d_phi(ii) = trace(invI*GI(x(:,ii)))-q; %optimality condition
end
```

A.4 Selected code from Chapter 4

A MATLAB implementation of Algorithm 4 as a function is presented, followed by the set-up for the three numerical examples in Chapter 4.

A.4.1 Iterative CVX algorithm

```
function [design, x, h_ti, cvx_results, I, N] = iterative_cvx(x, x1, GI, phi,
    suff, opts)
%ITERATIVE_CVX performs the iterative CVX method in Ch.4 of thesis
%   x - DDxN matrix containing all points in Omega_N
%   x1 - initial sample
%   GI - function for elementary moment matrices (input: 1xk row vector,
%       output: kxk matrix)
%   phi - MINIMIZABLE optimality criterion of interest (MUST BE
%       CVX-COMPATIBLE)
%       i.e. @(I) trace_inv(I) for A-optimality
%            @(I) -det_rootn(I) for D-optimality
%           @(I) trace_inv(inv(A)*I) for I-optimality (need to make sure
%           inv(A)*I is ***symmetric***!!!
%   suff - optimality condition function
%   opts: the customizable settings of the algorithm, additional inputs.
%   alf - size factor for the updating rule, scalar in [0,1].
%   maxiter - maximum number of iterations
%   tol - 2 small positive numbers for weight cut-off & stopping criterion
%   clstr - boolean. Do you want to perform clustering of the design via
%           dbscan at the end?
%   T - reflection matrix for the update rule in the Algorithm. default is
%       the identity matrix
    arguments
        x double {mustBeDims(x)};
        x1 double {mustBeEqualSize(x1,x), mustBeIn(x1,x)};
        GI function_handle {mustBeSquare(GI, x1)};
        phi function_handle;
        suff function_handle;

        opts.alf (1,1) double =0.8;
        opts.maxiter (1,1) double =100;
        opts.tol (1,:) double =[1e-6 1e-6];
        opts.clstr logical =false;
        opts.T double {mustBeSquare(opts.T,1)} = eye(size(x,1));
    end
    DD = size(x,1); N=size(x,2); s0 = size(x1,2);
    k = length(GI(ones(DD,1)'));
    alf = opts.alf;
```

```
    maxiter = opts.maxiter;
    tol = opts.tol(length(opts.tol)); %stopping criterion
    tol2 = opts.tol(1); %weight cut-off
    clstr = opts.clstr;
    T = opts.T;

    %%% iterative method

    hmax = 0; %the value of max(h_ti) from the previous iteration, because
    % sometimes max(h_ti) does not reach tol.
    for m=1:maxiter
        cvx_begin
            cvx_precision high
            variable w1(s0) nonnegative;
            expression I(k,k);
            for j=1:s0       %computing the Info matrix, which depends on the
               weights
               I = I + GI(x1(:,j)')*w1(j);
            end
                minimize phi(I);
            sum(w1) == 1;
        cvx_end

        design = [w1(w1>tol2), x1(:,(w1>tol2))'];

        invI = inv(I);
        h_ti = arrayfun(@(n) suff(invI, x(:,n)'), 1:N);

        if abs(hmax-max(h_ti)) <=tol %abs(max(h_ti)-hmax)<=tol %if the max and
           previous max are reallly close to zero, end.
            break
        else
            hmax = max(h_ti); %update maximum value

            %add all the points within alf*100% of maximum condition value
            x2a = union(design(:,2:(DD+1)), x(:, h_ti>=alf*hmax)',"rows")';
            x2b = (x2a'*T)'; %reflect using T matrix
            x1 = union(x2a', x2b', 'rows', 'sorted')';
            s0=length(x1); %update size of sample.
        end

    end %of iterations

    %clustering points together (optional):
    if clstr
        design1 = [w1((w1>tol2)), x1(:,(w1>tol2))'];
        idx = dbscan(design1(:,2:(DD+1)),0.05*(DD-1),1); %clustering region,
           epsilon=0.05 is usually good.
        design = zeros(max(idx), DD+1);
        for c=1:max(idx)
            tst = design1(idx==c,:);
            pt_tst = tst(:,1)'*tst(:,2:(DD+1))/sum(tst(:,1)); %take weighted
               average of points per cluster
            design(c,:) = [sum(tst(:,1)), pt_tst]; %add their weights together
```

```
        end
        x1=design(:,2:(DD+1))'; s0 = size(x1,2);
        x = union(x', x1', "rows", 'sorted')'; N=size(x,2);

        cvx_begin quiet %one last run of CVX to see if weights need re-
            adjusting after clustering
            cvx_precision high
            variable w1(s0) nonnegative;
            expression I(k,k);
            for j=1:s0
               I = I + GI(x1(:,j)')*w1(j);
            end
                minimize phi(I);
            sum(w1) == 1;
        cvx_end
        invI = inv(I);
        design = [w1(w1>tol2), x1(:,(w1>tol2))'];

        h_ti = arrayfun(@(n) suff(invI, x(:,n)'), 1:N);
    end
    cvx_results = dictionary(["cvx_optval" "cvx_status" "num_iter"],...
        {cvx_optval cvx_status m});
end %of function

function mustBeDims(x)
    % Test for equal size
    if size(x,1)>size(x,2)
        eid = 'size:WrongDimensions';
        msg = ['Transpose Input. Candidate set must be a ',num2str(size(x,2)),'
            x', num2str(size(x,1)),' matrix.'];
        error(eid,msg)
    end
end

function mustBeEqualSize(a,b)
    % Test for equal size
    if all([~isequal(size(a,1),size(b,1)), size(a,2)<=size(b,2)])
        eid = 'Size:notEqual';
        msg = 'Initial sample does not have correct dimensions.';
        error(eid,msg)
    end
end

function mustBeIn(value, S)
    % Test for equal size
    if prod(ismember(value', S', 'rows'))==0
        eid = 'size:WrongDimensions';
        msg = ['Initial sample must be a subset of the candidate set'];
        error(eid,msg)
    end
end

function mustBeSquare(GI, x)
    %test that the output for function GI is a square matrix
```

```
    if isUnderlyingType(GI, 'function_handle')
        r = randi(length(x));
        s = size(GI(x(:,r)));
        msg = 'Output of GI(x) must be square';
    else
        s = size(GI);
        msg = 'A must be square';
    end
    if s(1) ~= s(2)
        eid = 'size:MatrixNotSquare';
        error(eid,msg)
    end
end
```

A.4.2 Wynn's polygon

```
%% wynns polygon code
% requires 'iterative_cvx.m' in the same directory!

%% design region set-up
%the vertices of the design region
xv = [-1 -1 1 2 -1]*sqrt(2)/4;
yv = [1 -1 -1 2 1]*sqrt(2)/4;
V=[xv; yv]';
omega = polyshape(xv, yv); %continuous region

a = ([min(xv), min(yv)]); b=([max(xv), max(yv)]); %bounding box

%%% discretization %%%
n=ceil(201*sqrt(prod(b-a)/area(omega))); % number of pts/axis
xy = combvec(linspace(a(1), b(1), n), linspace(a(2), b(2), n))';

[in, on] = inpolygon(xy(:,1), xy(:,2), xv, yv);
% in = interior and boundary points
% on = just boundary points

omg = union(xy(in,:), V, 'rows')'; %FINAL DISCRETE SET

%% model and optimality set-up
g = @(x) [1 x(1) x(1)^2 x(2) x(2)^2 x(1)*x(2)]';%2nd order
k=6;
GI = @(x) g(x)*g(x)'; %for info matrix

optfunc = {'D_optim', 'A_optim'};
opt = optfunc{2}; %choose index of which model you want to run

if strcmpi(opt, "D_optim")
    phi= @(I) -det_rootn(I); %objective function, D-optimality
    suff = @(invI, x) trace(invI*GI(x))-k;
else
```

```
    phi = @(I) trace_inv(I); %A-optimality
    suff = @(invI, x) trace(invI*GI(x')*invI - invI);
end


rng(123)
x1 = datasample(omg,100, 2,'Replace',false); %INITIALIZE SAMPLE

tic
[design, x, h_ti, outcvx, I, N] = iterative_cvx(omg, x1, GI, phi, suff,...
    alf=0.5, clstr=false, T=[0 1; 1 0], tol=[1e-6, 1e-6])
disp('END OF METHOD')
runtime=toc;
```

A.4.3 Folium

```
%% folium, third order model, D and I-optimality
% requires 'iterative_cvx.m' in same directory!

%% design region set up
% start with boundary of omega (polar coordinate eq. is used here)
nt = 3001;
theta=linspace(-pi/2,pi/2, nt);
r = round(cos(theta).*(3*sin(theta).^2-1), 13);
[xv, yv] = pol2cart(theta, r);
xv = round(xv, 13); yv = round(yv, 13);
V = unique([xv; yv]', "rows");

vol = 0.490871; %volume of folium, computed numerically

% create lattice to generate pts inside omega
a = ([min(xv), min(yv)]); b=([max(xv), max(yv)]); %bounding box
n=ceil(201*sqrt(prod(b-a)/vol)); % number of pts/axis
xy = combvec(linspace(a(1), b(1), n), linspace(a(2), b(2), n))';

% cartesian coord. eq. is easier to use to reject pts in xy:
fol = @(x) -x(1)*(x(1)^2-2*x(2)^2)-(x(1)^2+x(2)^2)^2;
in = arrayfun(@(n) fol(xy(n,:))>=0, 1:length(xy));
on = arrayfun(@(n) fol(xy(n,:))==0, 1:length(xy));

omg = union(xy(in,:), V, 'rows')'; %%% FINAL DISCRETE SET %%%

%% model set up and optimality function

% g(x) = basis functions
g = @(x) [1 x(1) x(1)^2 x(1)^3 x(2) x(2)^2 x(2)^3 x(1)*x(2) x(1)^2*x(2) x(1)*x
    (2)^2]';
k = 10; %dimensions of info matrix, k*k
GI = @(x) g(x)*g(x)'; %for info matrix
```

```
rng(23); %set seed

optfunc = {'D_optim', 'I_optim'};
opt = optfunc{1}; %choose between D and I optimality

if strcmpi(opt, "D_optim")
    A=1;
    phi= @(I) -det_rootn(I); %objective function, D-optimality
    suff = @(invI, x) trace(invI*GI(x))-k;
else
    % generate random pts (symmetric across x_2=0):
    xa = [a(1) 0] + (b-a).*rand(ceil(1000000*prod(b-a)/vol),2);
    int_f = arrayfun(@(n) fol(xa(n,:)), 1:length(xa));
    xa = unique([xa(int_f>=0, :); xa(int_f>=0,:)*[1 0; 0 -1]], 'rows');

    %%% NUMERICALLY COMPUTE 'A' %%%
    tmpA = arrayfun(@(n) GI(xa(n,:)')./vol, 1:length(xa), 'UniformOutput',false
       );
    tmp2A = reshape([tmpA{1:end}],k,k,[]);
    A = round(mean(tmp2A,3)*vol,13);
    A2 = sqrtm(A); A2inv = inv(A2);

    AI = @(I) 0.5*(A2inv*I*(A2inv') + (A2inv*I*(A2inv'))');
    phi = @(I) trace_inv(AI(I)); %I-optimality
    suff = @(invI, x) trace(A2*(invI*GI(x')*invI - invI)*A2);
end

%% iterative alg
x1 = datasample(omg,100, 2,'Replace',false); %random

tic
[design, x, h_ti, outcvx, I, N] = iterative_cvx(omg, x1, GI, phi, suff,...
    alf=0.5, clstr=true, T=[1 0; 0 1], tol=[1e-6, 1e-6]);
disp('END OF METHOD')
runtime=toc;

%optimality condition at design points (range):
[min(arrayfun(@(n) suff(inv(I), design(n,2:3)), 1:length(design)))...
max(arrayfun(@(n) suff(inv(I), design(n,2:3)), 1:length(design)))]
```

A.4.4 Potato packing

```
%% logistic model (potato packing)
% requires 'iterative_cvx.m' in the same directory!

%% setting up design region
n=101; % # of points/ unit interval
```

```
x = combvec(linspace(-1,1,n), linspace(-1,1,n), linspace(-1, 1, n)); %uniform
    grid
vol=2^3;

%%% MODEL SET-UP %%%
g = @(x) [1 x(2) x(3) x(2)*x(3) x(1)^2 x(2)^2 x(3)^2]';
k = 7;

beta = [-2.93, -0.52, -0.79, -0.66, 0.94, 0.79, 1.82];
v = @(x) exp(beta*g(x))/(1+exp(beta*g(x)))^2;
GI = @(x) g(x)*g(x)'*v(x); %for info matrix

%%% COMPUTING 'A' NUMERICALLY: %%%
rng(15) %set seed
n1=151; %number of points/axis
N1 = n1^3;
xa = (2*rand(N1,3) -1)'; %generate random points

tmpA = arrayfun(@(n) GI(xa(:,n))*v(xa(:,n))/vol, 1:N1, 'UniformOutput', false);
tmp2A = reshape([tmpA{1:end}],k,k,[]);
A = mean(tmp2A,3)*vol;

A2 = sqrtm(A); A2inv = inv(A2);
AI = @(I) 0.5*(A2inv*I*(A2inv') + (A2inv*I*(A2inv')))';

%optimality criteria and condition
phi = @(I) trace_inv(AI(I)); %I-optimality
suff = @(invI, x) trace(A2*(invI*GI(x')*invI - invI)*A2);

%iterative algorithm
x1 = datasample(x,1000, 2,'Replace',false); %random
tic;
[design, x, h_ti, outcvx, I, N] =  iterative_cvx(x,x1,GI,phi, suff,...
    alf=0.5, clstr=true, T=[-1 0 0; 0 1 0; 0 0 1], tol=[1e-4, 1e-6]);
runtime=toc;
```

Bibliography

Abousaleh, H., & Zhou, J. (2023). Minimax A-, c-, and I-optimal regression designs for models with heteroscedastic errors. *Canadian Journal of Statistics*, *51*, 258–274. https://doi.org/10.1002/cjs.11674

Atkinson, A., Donev, A., & Tobias, R. (2007). *Optimum experimental design, with SAS*. Oxford University Press, Inc.

Atkinson, A., & Woods, D. (2015). Designs for generalized linear models. In A. Dean, M. Morris, J. Stufken, & D. Bingham (Eds.), *Handbook of design and analysis of experiments* (pp. 471–514). Chapman; Hall/CRC. https://doi.org/10.1201/b18619

Berger, M. P., & Wong, W. K. (2009). *An introduction to optimal designs for social and biomedical research*. John Wiley & Sons Ltd.

Bose, M., & Mukerjee, R. (2015). Optimal design measures under asymmetric errors, with application to binary design points. *Journal of Statistical Planning and Inference*, *159*, 28–36. https://doi.org/10.1016/j.jspi.2014.10.006

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.

Chen, R. B., Chang, S. P., Wang, W., Tung, H. C., & Wong, W. K. (2015). Minimax optimal designs via particle swarm optimization methods. *Statistics and Computing*, *25*, 975–988. https://doi.org/10.1007/s11222-014-9466-0

Cook, R. D., & Nachtsheim, C. J. (1980). A comparison of algorithms for constructing exact D-optimal designs. *Technometrics*, *22*(3), 315–324. https://doi.org/10.2307/1268315

De Castro, Y., Gamboa, F., Henrion, D., Hess, R., & Lasserre, J.-B. (2019). Approximate optimal designs for multivariate polynomial regression. *Annals of Statistics*, *47*(1), 127–155. https://doi.org/10.1214/18-AOS1683

Duan, J., Gao, W., Ma, Y., & Ng, H. K. T. (2022). Efficient computational algorithms for approximate optimal designs. *Journal of Statistical Computation and Simulation*, *92*(4), 764–793. https://doi.org/10.1080/00949655.2021.1974439

Duan, J., Gao, W., & Ng, H. K. T. (2019). Efficient computational algorithm for optimal continuous experimental designs. *Journal of Computational and Applied Mathematics*, *350*, 98–113. https://doi.org/10.1016/j.cam.2018.09.046

Fedorov, V. V. (1972). *Theory of optimal experiments* (W. Studden & E. Klimko, Trans.). Academic Press.

Fedorov, V. V., & Leonov, S. L. (2014). *Optimal design for nonlinear response models* (1st ed.). CRC Press.

Gao, L. L., & Zhou, J. (2017). D-optimal designs based on the second-order least squares estimator. *Statistical Papers*, *58*(1), 77–94. https://doi.org/10.1007/s00362-015-0688-9

Goos, P., Jones, B., & Syafitri, U. (2016). I-optimal design of mixture experiments. *Journal of the American Statistical Association*, *111*(514), 899–911. https://doi.org/10.1080/01621459.2015.1136632

Grant, M., & Boyd, S. (2020, March). CVX: Matlab software for disciplined convex programming, version 2.2. http://cvxr.com/cvx

Haines, L. M. (1987). The application of the annealing algorithm to the construction of exact optimal designs for linear-regression models. *Technometrics*, *29*(4), 439–447. https://doi.org/10.2307/1269455

Idais, O., & Schwabe, R. (2021). Equivariance and invariance for optimal designs in generalized linear models exemplified by a class of gamma models. *Journal of Statistical Theory and Practice*, *15*(93). https://doi.org/10.1007/s42519-021-00221-z

Kiefer, J. (1974). General equivalence theory for optimum designs (approximate theory). *Ann. Stat.*, *2*(5), 849–879. https://doi.org/10.1214/aos/1176342810

Kiefer, J., & Wolfowitz, J. (1959). Optimum designs in regression problems. *Annals of Mathematical Statistics*, *30*(2), 271–294. https://doi.org/10.1214/aoms/1177706252

Li, Y., & Deng, X. (2021). An efficient algorithm for elastic I-optimal design of generalized linear models. *Canadian Journal of Statistics*, *49*(2), 438–470. https://doi.org/10.1002/cjs.11571

Mitchell, T. J. (1974). An algorithm for the construction of "D-optimal" experimental designs. *Technometrics*, *16*(2), 203–210. https://doi.org/10.2307/1267940

Molchanov, I., & Zuyev, S. (2002). Steepest descent algorithms in a space of measures. *Statistics and Computing*, *12*(2), 115–123. https://doi.org/10.1023/A:1014878317736

Papp, D. (2012). Optimal designs for rational function regression. *Journal of the American Statistical Association*, *107*, 400–411. https://doi.org/10.1080/01621459.2012.656035

Pukelsheim, F., & Rieder, S. (1992). Efficient rounding of approximate designs. *Biometrika*, *79*(4), 763–770. https://doi.org/10.2307/2337232

Rankin, I., & Zhou, J. (2023). Bayesian and maximin A-optimal designs for spline regression models with unknown knots. *Statistical Papers*. https://doi.org/10.1007/s00362-023-01469-2

Rempel, M. F., & Zhou, J. (2014). On exact K-optimal designs minimizing the condition number. *Communications in Statistics - Theory and Methods*, *43*, 1114–1131. https://doi.org/10.1080/03610926.2012.670352

Rinne, H. (2010). Location–scale distributions: Linear estimation and probability plotting using MATLAB [Retrieved June 22, 2023]. http://geb.uni-giessen.de/geb/volltexte/2010/7607/pdf/RinneHorst_LocationScale_2010.pdf

Silvey, S., Titterington, D., & Torsney, B. (1978). An algorithm for optimal designs on a design space. *Communications in Statistics - Theory and Methods*, *7*(14), 1379–1389. https://doi.org/10.1080/03610927808827719

Stufken, J., & Yang, M. (2012). Optimal designs for generalized linear models. In *Design and analysis of experiments: Special designs and applications* (pp. 137–164, Vol. 3). John Wiley & Sons Ltd. https://doi.org/10.1002/9781118147634.ch4

The MathWorks Inc. (2023a). *Deep learning toolbox version: 14.6 (r2023a)*. Natick, Massachusetts, United States. https://www.mathworks.com/products/deep-learning.html

The MathWorks Inc. (2023b). *Statistics and machine learning toolbox version: 12.5 (r2023a)*. Natick, Massachusetts, United States. https://www.mathworks.com/products/statistics.html

Theune, U. (2005). *Ternary plots*. MATLAB Central File Exchange. https://www.mathworks.com/matlabcentral/fileexchange/7210-ternary-plots

Wong, W. K., & Zhou, J. (2019). CVX-based algorithms for constructing various optimal regression designs. *Canadian Journal of Statistics*, *47*(3), 374–391. https://doi.org/10.1002/cjs.11499

Woods, D. C., Lewis, S. M., Eccleston, J. A., & Russell, K. G. (2006). Designs for generalized linear models with several variables and model uncertainty. *Technometrics*, *48*(2), 284–292. https://doi.org/10.1198/004017005000000571

Yang, M., Biedermann, S., & Tang, E. (2013). On optimal designs for nonlinear models: A general and efficient algorithm. *Journal of the American Statistical Association*, *108*(504), 1411–1420. https://doi.org/10.1080/01621459.2013.806268

Ye, J. J., Zhou, J., & Zhou, W. (2017). Computing A-optimal and E-optimal designs for regression models via semidefinite programming. *Communications in Statistics - Simulation and Computation*, *46*(3), 2011–2024. https://doi.org/10.1080/03610918.2015.1030414

Yeh, C., & Zhou, J. (2021). Properties of optimal regression designs under the second-order least squares estimator. *Statistical Papers*, *62*, 75–92. https://doi.org/10.1007/s00362-018-01076-6

Yu, Y. (2011). D-optimal designs via a cocktail algorithm. *Statistics and Computing*, *21*, 475–481. https://doi.org/10.1007/s11222-010-9183-2