NORTHWESTERN UNIVERSITY

Space-Filling Designed Sampling from Databases

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Industrial Engineering and Management Science

By

Boyang Shang

EVANSTON, ILLINOIS

September 2022

Abstract

Space-Filling Designed Sampling from Databases

Boyang Shang

This thesis develops novel methods for generating space-filling designs inside a design space and subsampling from a data set. It incorporates materials from two papers by the author: Shang and Apley 2021; Shang, Apley, and Mehrotra 2022a.

Chapter 1 discusses space-filling designs of computer experiments, which is published as Shang and Apley 2021. Fully-sequential (i.e., with design points added one-at-a-time) space-filling designs are useful for global surrogate modeling of expensive computer experiments when the number of design points required to achieve a suitable accuracy is unknown in advance. We develop and investigate three fully-sequential space-filling (FSSF) design algorithms that are conceptually simple and computationally efficient and that achieve much better space-filling properties than alternative methods such as Sobol sequences and more complex batch-sequential methods based on sliced or nested optimal Latin hypercube designs (LHDs). Remarkably, at each design size in the sequence, our FSSF algorithms even achieve much better space filling properties than a one-shot LHD

optimized for that specific size. The algorithms we propose also scale well to very large design sizes. We provide the FSSF R package to implement the approaches.

Chapter 2 focuses on diversity subsampling from a data set, which is published as Shang, Apley, and Mehrotra 2022a. Subsampling from a large data set is useful in many supervised learning contexts to provide a global view of the data based on only a fraction of the observations. Diverse (or space-filling) subsampling is an appealing subsampling approach when no prior knowledge of the data is available. In this chapter, we propose a diversity subsampling approach that selects a subsample from the original data such that the subsample is independently and uniformly distributed over the support of distribution from which the data are drawn, to the maximum extent possible. We give an asymptotic performance guarantee of the proposed method and provide experimental results to show that the proposed method performs well for typical finite-size data. We also compare the proposed method with competing diversity subsampling algorithms and demonstrate numerically that subsamples selected by the proposed method are closer to a uniform sample than subsamples selected by other methods. The proposed DS algorithm is shown to be more efficient than known methods and takes only a few minutes to select tens of thousands of subsample points from a data set of size one million. Our DS algorithm easily generalizes to select subsamples following distributions other than uniform. We provide the FADS Python package to implement the proposed methods.

Acknowledgements

I am deeply indebted to many people who have supported me professionally and/or personally to complete my dissertation. I would like to express my sincere gratitude to them for making this endeavor not only possible but also enjoyable.

I could not have undertaken this journey without the professional guidance, academic inspiration, and financial support from my advisor Prof. Daniel W. Apley, and my co-advisor Prof. Sanjay Mehrotra. Words cannot express my gratitude to them for providing the precious opportunity and intensive training for me to grow as a researcher. I would like to express my deepest gratitude to Prof. Apley for introducing me to the world of statistics and industrial engineering, for the insightful and inspiring feedback that I received on a weekly basis, and for helping me do better both in research and in life. I'm also extremely grateful to Prof. Mehrotra for his motivating, practical, and knowledgeable inputs on the Diversity Subsampling project and for the opportunity to discover the impact of the developed methods in the field of healthcare. Last but not least, I deeply appreciate their unwavering support during the difficult time of the Covid-19 pandemic and visa restriction.

I would like to express my deep appreciation to Prof. Edward C. Malthouse for serving on my thesis committee; to Prof. Ajit C. Tamhane for being a reference for my job search and for teaching me so much when I worked as a teaching assistant for him; to Prof. Barry L. Nelson for his great suggestions for my professional development; to Prof.

Seyed MR Iravani, our Director of Graduate Studies, for his support when I encountered difficulties demonstrating the effectiveness of methods proposed in this thesis; to Prof. Joel L Horowitz for the knowledgeable inputs on multivariate density estimation; and to all the IEMS faculty for your devotion to both teaching and research.

I would like to extend my sincere thanks to all the IEMS staff members, especially our previous Graduate Coordinator Stephen Erik Pedersen, for his steady support when I was not able to get back to campus due to the F1 visa restriction. I am also thankful to staff members for the Quest high-performance computing facility at Northwestern and for the university library. I'd like to acknowledge the National Science Foundation, the Advanced Research Projects Agency-Energy, the U.S. Department of Energy, the Center of Engineering and Health at the NU, and the IEMS department as the funding agencies for this work.

I had the pleasure of sharing this wonderful journey with so many brilliant and warm-hearted classmates and colleagues. I am very thankful for all of them, especially, Suraj Yerramilli, Lun Yu, Kungang Zhang, Liwei Zeng, Kevin Bui, Yujing Lin, Moses Chan, Yuchen Xie, Fengqiao Luo, Shengtong Zhang, Yi Zhu, Allen Wu, Yintai Ma, and Yayu Zhou. Special thanks to Lun Yu for his kind help during my difficult time of quarantine. I am also very grateful to Virginia McMillan Carr, who has been a friend, gymmate as well as life mentor for me since I entered graduate school. I cannot thank her enough for the encouragement and personal support throughout my study at Northwestern, especially during the Covid-19 pandemic, when I had to complete my thesis remotely.

Last but not least, I would like to express my sincere gratitude to my parents, for allowing and supporting me to be myself, and to my grandparents, for their unconditional

love! I would be remiss in not mentioning my two lovely feline friends, Orange and Tiny Tiger; luck is on my side to gain your trust and friendship!

Dedication

I dedicate this thesis to everyone who pursues her/his dream, despite any obstacles in the past and the future.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Fully-Sequential Space-filling Design Algorithms for Computer Experiments

1.1. Introduction

When the objective is to obtain a globally-accurate surrogate model for an expensive computer simulation model, and the experimental design size required to achieve some desired accuracy is not known in advance (which is usually the case), a fully-sequential space-filling design is useful. By fully-sequential and space-filling, we mean that design points (i.e., simulation input locations) are added one-at-a-time, and as each new point is added the design retains good space-filling properties. Fully-sequential designs are useful because they allow the user to continue simulating the computer model response at additional design points, each time updating and improving the surrogate model, stopping when the desired accuracy level is reached. This avoids having to conduct more simulation runs of the expensive computer model code than necessary.

Minimum pairwise distance and maximum hole size are two common measures of how space-filling a design is. Minimum pairwise distance is the distance between the pair of closest design points, and the largest hole size is the distance between any location in the design region and its closest point in the design, maximized over all locations in the design region. More specifically, let $\boldsymbol{x} \in \mathbb{R}^q$ denote the vector of input variables for a design point, where q denotes the number of inputs. We assume a rectangular input design space Ω and, without loss of generality, that the inputs have been scaled so that $\Omega = [0,1]^q$ is the unit hypercube. Let $S_n = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\}$ denote a design of size n, where $\boldsymbol{x}_i = \{x_{i1}, x_{i2}, \cdots, x_{iq}\}$ for $i = 1, 2, \cdots, n$. And let $d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|$ be the distance between point $\boldsymbol{x}$ and $\boldsymbol{y}$, where $\boldsymbol{x}, \boldsymbol{y} \in \Omega$. In this chapter we only consider

Euclidean norm. Then the minimum pairwise distance of a design S_n is

$$d_{\min}(S_n) \stackrel{\Delta}{=} \min_{\substack{i,j\in\{1,\cdots,n\}\\ i<j}} d(\boldsymbol{x}_i, \boldsymbol{x}_j). \tag{1.1}$$

Similarly, the largest hole size of a design S_n is

$$h_{\max}(S_n) \stackrel{\Delta}{=} \max_{\boldsymbol{x}\in\Omega} \min_{i\in\{1,\cdots,n\}} d(\boldsymbol{x}_i, \boldsymbol{x}). \tag{1.2}$$

As a design construction criterion, for computational reasons most existing works use the minimum pairwise distance criterion (Kennard and Stone 1969; Morris and Mitchell 1995; Jin, W. Chen, and Sudjianto 2005; R. Joseph and Hung 2008; Rennen et al. 2010; S. Ba, Myers, and Brenneman 2015; V Roshan Joseph, Gul, and Shan Ba 2015) instead of the maximum hole size criterion (Tan 2013). In this chapter we will consider both the minimum pairwise distance criterion (Equation (1.1)) and a surrogate for the maximum hole size criterion (Equation (1.2)). We do not discuss the space-filling performances of designs in projected spaces, for which interested readers are referred to V Roshan Joseph, Gul, and Shan Ba 2015.

For the objective of optimizing the response of an expensive computer simulation model, sequential sampling is a well-researched area, with perhaps the most popular method being the Gaussian process (GP) based method of Jones, Schonlau, and Welch 1998. For the response surface optimization objective, each new design point cannot be determined until after the response is simulated at the previous design point. In contrast, for the globally-accurate surrogate modeling objective that we consider in this chapter, the entire sequence of design points can be determined in advance and then simulated

one-by-one, or in batches if desired. For the popular GP (aka, kriging) surrogate models, the response prediction mean square error (MSE) depends only on the input locations and not on the response itself, assuming the covariance function is specified (Sacks et al. 1989). Hence, measures of the quality of the design based on the integrated or maximum MSE can be calculated up front, given only the input locations in the design, prior to simulating the response. However, the MSE depends on the covariance function, which generally must be estimated based on the response observations to be simulated. Although there are no theoretical proofs of MSE-optimality for space-filling designs in realistic non-asymptotic scenarios, it has been observed via empirical studies that designs with good space-filling properties generally have good MSE properties. Thus space-filling designs like Latin hypercube designs (LHDs) have emerged as the most popular approach for designing computer experiments for globally-accurate surrogate modeling (Santner, Williams, and Notz 2013; V Roshan Joseph 2016). Indeed, Loeppky, Leslie M Moore, and Williams (2010) investigated various sequential designs (both batch and one-at-a-time) and found that maximin-distance designs perform competitively with sequential designs that observe the response data and update the GP model accordingly at each design stage, and then select the next design point(s) to optimize some criterion based on the predicted MSE from the GP model (also see Johnson, L. M. Moore, and Ylvisaker (1990) for related asymptotic results). Having good space-filling properties is important not only to ensure that the input space is uniformly covered and the MSE properties are good, but also because having input locations that are too closely spaced creates numerical difficulties with Gaussian process models for deterministic computer simulations.

There has been substantial recent work on batch-sequential (i.e., with design points added in batches or groups) space-filling designs for computer experiments, most of which are variations of sliced LHDs (SLHDs) or nested LHDs (NLHDs). These works include P. Z. Qian and C. J. Wu 2009; P. Qian 2009; Rennen et al. 2010; P. Qian 2012; Yang, Liu, and Lin 2014; S. Ba, Myers, and Brenneman 2015; Duan et al. 2017; Kong, Ai, and Tsui 2017; D. Chen and Xiong 2017. There also exist two much older fully-sequential (i.e., with design points added one-at-a-time) space-filling (FSSF) design approaches that have received much less attention in the computer experiment literature. Sobol sequences (Sobol 1967) are quasi-random fully-sequential sequences that are intended to uniformly cover a rectangular design region. As we demonstrate later in this chapter, Sobol sequences have much poorer space-filling properties than the other fully-sequential method Computer Aided Design of EXperiments (CADEX), introduced in Kennard and Stone 1969.

The CADEX algorithm is conceptually quite simple. It requires a set of candidate design points whose size is much larger than the size of the largest desired design. Beginning with a design comprised of the two furthest points in the candidate set, design points chosen among the remaining candidate set are added to the existing design one-at-a-time, with each new point added to satisfy a maximin criterion. That is, given S_n, a new point $\boldsymbol{x}_{n+1}$ is added to maximize $d_{\min}(S_n \cup \boldsymbol{x}_{n+1})$. The CADEX method was originally conceived for designing physical experiments with a relatively small candidate set, such as a grid of values over a low-dimensional design space in which each input can assume only a discrete set of values, and with the computational capabilities available in 1969. Methods like CADEX – that select design points one-at-a-time from some large candidate set –

appear to have been largely unexplored for the purpose of sequential design of computer experiments.

The main purpose of this chapter is to develop and investigate a number of FSSF design algorithms in the context of globally accurate surrogate modeling of computer experiments. In particular, we demonstrate that our conceptually simple FSSF design approaches have the following substantial advantages over the more heavily researched SLHD and NLHD methods (collectively, S/NLHD). First, our FSSF designs have much better space-filling properties than S/NLHD designs (and also better than Sobol sequences), in terms of both minimum pairwise distance and maximum hole size. Even more remarkably, we demonstrate that the FSSF designs at each design size in the sequence have much better space filling properties than a one-shot LHD optimized for that specific size. The second advantage over S/NLHDs is that our FSSF designs are fully-sequential, whereas the S/NLHDs are only batch-sequential. Some S/NLHDs that allow batches of equal size can be made closer to fully-sequential by reducing the batch size, but this further degrades their space-filling performance. We reiterate that by 'sequential', we mean a series of nested designs of increasing size, such that the designs at each size retain good space-filling properties. For example, the designs produced for sizes $n = 10$ and $n = 20$ should both be as space-filling as possible, and the size-10 design points must be a subset of the size-20 design points (the nested property). The goal is to ensure that the sequential design is as space-filling as possible at every stage/size. The third advantage is conceptual simplicity and low computational expense relative to S/NLHD methods optimized with respect to the $d_{\min}()$ (Equation (1.1)) or $h_{\max}()$ (Equation (1.2)) criteria.

Neither one-shot designs, nor our sequential designs, take the response observations into account while selecting the design points, but there is a fundamental difference between the two. If the experimenter decides to terminate a one-shot design early after observing the response observations collected up to that point, then the incomplete one-shot design may be poorly space-filling. Likewise, if the experimenter decides additional runs are needed after completing a one-shot design, the augmented design may not be as space-filling as possible. On the other hand, our sequential designs retain excellent space-filling properties for each sample size, regardless of whether the experimenter decides to terminate early or to augment the design.

The format of the remainder of the chapter is as follows. In Section 1.2 we introduce three FSSF algorithms, denoted FSSF-f, FSSF-fr, and FSSF-b. The FSSF-f algorithm is similar to the CADEX algorithm proposed by Kennard and Stone 1969 but with choice of candidate set and computational improvements tailored to the situation of computer experiments with modern computational capabilities. Here 'f' stands for 'forward', since FSSF-f adds points to the design one-at-a-time. The FSSF-fr algorithm (Section 1.2.2), where 'fr' stands for 'forward reflected', uses a modified version of the minimum pairwise distance that also considers points reflected across the design region boundary. The purpose of considering reflected points is to improve the maximum hole size (Equation (1.2)) performances of the designs. The FSSF-b algorithm (Section 1.2.3), where 'b' stands for 'backward', uses backward removal (as opposed to forward addition) of the design points from the candidate set, which improves computational expense for large designs. In Section 1.2.3.2 we provide algorithm details and discuss computational strategies, such as using an approximate nearest neighbors algorithm to quickly identify the points to

remove. Section 1.3 discusses choice of the candidate set, including size of the candidate set, and choice of criterion for removing candidate points. Section 1.4 numerically demonstrates (1) the space-filling advantages of the three FSSF design algorithms, relative to existing S/NLHD methods, Sobol sequences, and even one-shot LHDs optimized for each design size and (2) the computational advantages of the FSSF-b algorithm relative to FSSF-f and FSSF-fr. Section 1.6 concludes the chapter.

1.2. Three Fully-Sequential Space-filling (FSSF) Design Algorithms

We first introduce notation that will be used frequently throughout this section. Then we present the FSSF-f algorithm (a slightly modified version of CADEX with Sobol sequence as the candidate set) and discuss its potential drawbacks in the context of modern computer experiment design. The FSSF-fr and FSSF-b algorithms (Sections 1.2.2 and 1.2.3, respectively) are designed to improve the maximum hole size and computational expense properties, respectively, relative to the FSSF-f algorithm.

Let $n_{\max}$ denote the user-chosen maximum number of design points that one would need in the design space Ω. Let $\mathcal{C} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\} \subset \Omega$ denote the candidate set with size denoted by N. The candidate set $\mathcal{C}$ should be chosen to cover Ω fairly densely, and N should be chosen substantially larger than $n_{\max}$. We use a Sobol sequence for $\mathcal{C}$ (see Section 1.3 for choices of $\mathcal{C}$ and N). Let $\{i_1, i_2, \cdots, i_{n_{\max}}\}$ be the indices of candidate points selected for the sequential design (i.e., $\boldsymbol{x}_{i_1}$ is the first design point, $\boldsymbol{x}_{i_2}$ is the second design point, etc.) and note that $\{i_1, i_2, \cdots, i_{n_{\max}}\}$ is a subset of $\{1, 2, \cdots, N\}$. Denote the design at stage-n by $S_n = \{\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \cdots, \boldsymbol{x}_{i_n}\}$ for $n = 1, 2, \cdots, n_{\max}$, so that

$S_1 \subset S_2 \subset \cdots \subset S_{n_{max}} \subset \mathcal{C}$ is the nested design sequence from size 1 to size n_{max}. Denote the set of indices of the points in S_n by $I_n = \{i_1, i_2, \cdots, i_n\}$.

1.2.1. The FSSF-f Algorithm

The FSSF-f algorithm is a modified version of the CADEX algorithm (Kennard and Stone 1969) that uses a Sobol sequence as the candidate set, the motivation for which is to improve space-filling properties while being computationally manageable for large design size. As in the CADEX algorithm, we begin with no points in the design and sequentially add points one-at-a-time, with each new point added according to a maximin criterion, i.e., to maximize $d_{\min}(S_n \cup \boldsymbol{x}_{i_{n+1}})$. We summarize the FSSF-f algorithm in Algorithm 1.

Algorithm 1 FSSF-f Algorithm

Input: $n_{\max}$, N, q

1: Generate candidate set $\mathcal{C} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$ as a Sobol sequence

2: Randomly select one point $i_1 \in \{1, 2, \cdots, N\}$

3: $\boldsymbol{D} \leftarrow$ an array of size N

4: **for** $j = 1, 2, \cdots, N$ **do**

5: $\quad D_j \leftarrow d(\boldsymbol{x}_{i_1}, \boldsymbol{x}_j)$

6: **end for**

7: **for** $n = 2, 3, \cdots, n_{\max}$ **do**

8: $\quad i_n \leftarrow \arg\max_{j \in \{1,2,\cdots,N\} \setminus \{i_1, i_2, \cdots, i_{n-1}\}} D_j$

9: $\quad$ **for** $j = 1, 2, \cdots, N$ **do**

10: $\quad\quad D_j \leftarrow \min\{D_j, d(\boldsymbol{x}_{i_n}, \boldsymbol{x}_j)\}$ $(= \min_{\boldsymbol{x} \in S_n} d(\boldsymbol{x}, \boldsymbol{x}_j))$

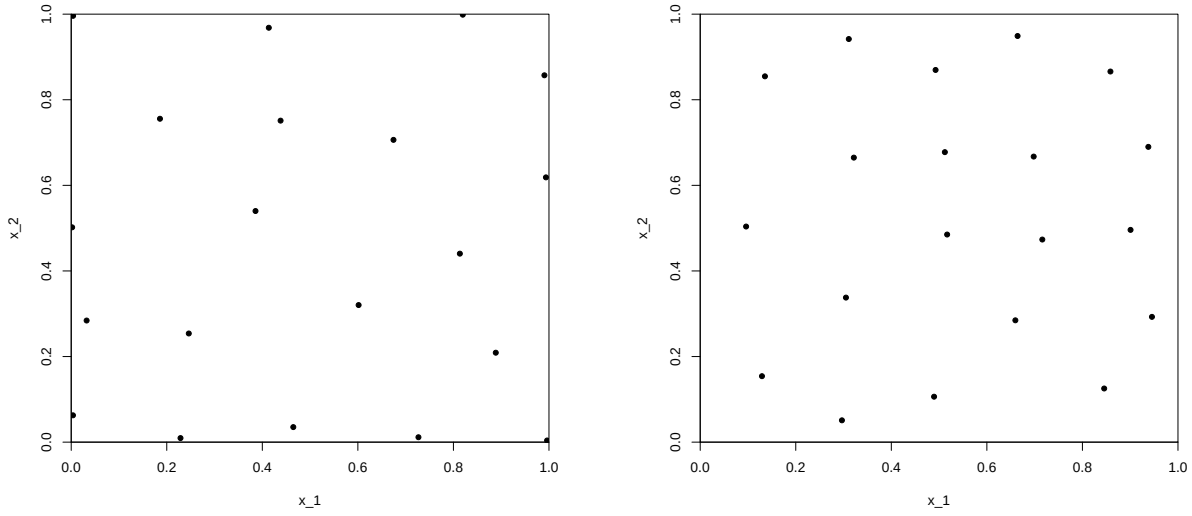
11: $\quad$ **end for**

12: **end for**

Output: Ordered FSSF design sequence $S_{n_{\max}} = \{\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \cdots, \boldsymbol{x}_{i_{n_{\max}}}\}$

The use of the N-length array $\boldsymbol{D}$ and its updating in line 10 of Algorithm 1 substantially improves the computational expense of finding the point in S_n closest to each of the remaining candidate points, relative to a brute force search over all n points on each iteration of the outer loop. In spite of this computational improvement, when $n_{\max}$ is large, the Algorithm 1 computational expense may be prohibitive, as its complexity is $O(N^2)$ under our choice of $N = 1000q + 2n_{\max}$ (see Section 1.3 for this choice) and q fixed. Our FSSF-b algorithm presented in Section 1.2.3 is more efficient for large n_{max} and has complexity $O(N \log N)$ with $N = 1000q + 2n_{\max}$ and q fixed.

Another potential drawback of the FSSF-f algorithm is that it tends to select design points near the boundary of Ω at the early stages, i.e., when n is small. Figure 1.1a shows a typical design at stage $n = 20$ using the FSSF-f algorithm for $q = 2$, from which we see that almost half of the design points are very near the boundary. Having many points on or very near the boundary does indeed does indeed improve the d_{min} behavior of the design. However, the drawback is that it also worsens the h_{max} behavior by leaving large areas in the interior region under-represented, especially in high dimension, as we will demonstrate in Section 1.4. This may negatively affect the accuracy of the computer model, especially for smaller n. For situations in which users prefer more emphasis on good h_{max} and less on good d_{min}, in Section 1.2.2 we propose the FSSF-fr algorithm.



(a) FSSF-f typical design

(b) FSSF-fr typical design

Figure 1.1. Comparison of typical designs at stage $n = 20$ for (a) FSSF-f (better d_{min} but worse h_{max}) vs. (b) FSSF-fr (worse d_{min} but better h_{max}). Both designs are for $n_{\max} = 1000$ and $q = 2$. The candidate set for both designs are Sobol sequences with size $N = 1000q + 2n_{\max}$.

1.2.2. The FSSF-fr Algorithm

The goal in this section is to develop an algorithm that has better h_{max} performance than the FSSF-f algorithm but that remains computationally efficient and retains good d_{min} performance. Since directly optimizing h_{max} is computationally prohibitive, we still use a criterion that is based on minimum pairwise distance and achieve our goal via a simple modification of the FSSF-f algorithm that avoids selecting points too close to the boundary, thereby encouraging smaller hole sizes in the interior regions.

Our approach is motivated by the observations from Figure 1.1a. Namely, the FSSF-f algorithm results in larger-than-optimal h_{max} because it tends to select design points very close to the boundary. Hence, for the FSSF-fr algorithm, we seek to avoid selecting design points too close to the boundary (unless n is large). We achieve this as follows. When deciding whether or not to add a candidate point, say $\boldsymbol{x}$, to the current design S_n, we pretend that its "reflected" point $R(\boldsymbol{x})$ with respect to the boundary $\partial\Omega$ of the design region is also included in S_n. We define $R(\boldsymbol{x})$ as the closest mirror image of $\boldsymbol{x}$ with respect to all hyperplanes that form $\partial\Omega$. Note that the distance between $\boldsymbol{x}$ and $R(\boldsymbol{x})$ is $2\min_{\boldsymbol{y}\in\partial\Omega} d(\boldsymbol{x},\boldsymbol{y}) = 2d(\boldsymbol{x},\boldsymbol{x}_\partial)$, as illustrated in Figure 1.2, where $\boldsymbol{x}_\partial = (\boldsymbol{x} + R(\boldsymbol{x}))/2$ is the point on the boundary bisecting the line segment between $\boldsymbol{x}$ and $R(\boldsymbol{x})$. By doing this, candidate points that fall too close to the boundary will not be added until the later stages, when n is large enough that interior hole sizes (i.e., h_{max} but restricted to points in the interior regions) are small enough to be on par with exterior holes (i.e., $d(\boldsymbol{x},\boldsymbol{x}_\partial)$ for design points $\boldsymbol{x}$ near the boundary). See Figure 1.2.

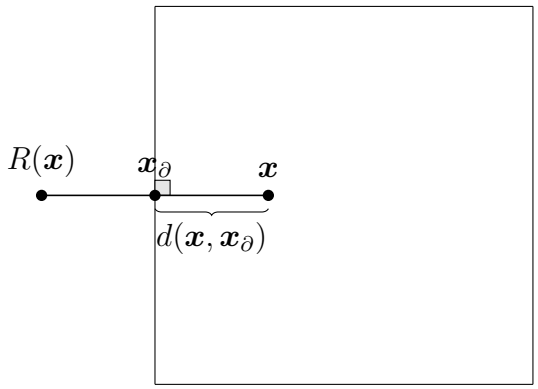


Figure 1.2. Illustration of the closest boundary point $\boldsymbol{x}_\partial$ and the reflected point $R(\boldsymbol{x})$ for a point $\boldsymbol{x}$.

One potential problem with the idea mentioned above is that in higher dimensions $d(\boldsymbol{x}, R(\boldsymbol{x}))$ may be inappropriately small compared to pairwise distances between design points. This is because the vector $R(\boldsymbol{x}) - \boldsymbol{x}$ always has a nonzero component in only one of its q coordinates, by definition of the distance from $\boldsymbol{x}$ to the hypercube boundary $\partial\Omega$. The result would be that even when n is relatively large, the algorithm would still tend to place design points in the interior of Ω and leave larger holes near $\partial\Omega$. To see this, consider the design point $\boldsymbol{x} = (\epsilon, \cdots, \epsilon) \in \mathbb{R}^q$ for some small ϵ, so that $\boldsymbol{x}$ is near $\partial\Omega$. Regardless of the value of q, the distance between $\boldsymbol{x}$ and $R(\boldsymbol{x})$ is always 2ϵ, whereas the hole size at the origin is $d(\boldsymbol{x}, \mathbf{0}) = \sqrt{q}\epsilon$ (assuming $\boldsymbol{x}$ is the closest design point to the origin), which increases with q. Likewise, distances between pairs of design points will tend to increase with q. In light of this, it may be helpful to weight $d(\boldsymbol{x}, R(\boldsymbol{x}))$ by a factor that increases with q. Generally speaking, using a larger factor will result in more points selected near $\partial\Omega$, which improves d_{min} performance at the expense of h_{max} performance. We have found through trial-and-error over various experiments (the results

of which are omitted for brevity) that using $\sqrt{2q}$ as the weighting factor provides a good balance overall.

Algorithm 2 FSSF-fr Algorithm

Input: $n_{\max}$, N, q

Generate candidate set $\mathcal{C} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$ as a Sobol sequence
$\boldsymbol{D} \leftarrow$ an array of size N
for $j = 1, 2, \cdots, N$ **do**
    $D_j \leftarrow 2\sqrt{2q} \min_{\boldsymbol{y} \in \partial\Omega} d(\boldsymbol{x}_j, \boldsymbol{y})$
end for
for $n = 1, 2, \cdots, n_{\max}$ **do**
    $i_n \leftarrow \arg\max_{j \in \{1,2,\cdots,N\} \setminus \{i_1, i_2, \cdots, i_{n-1}\}} D_j$
    for $j = 1, 2, \cdots, N$ **do**
        $D_j \leftarrow \min\{D_j, d(\boldsymbol{x}_{i_n}, \boldsymbol{x}_j)\}$
    end for
end for

Output: Ordered FSSF design sequence $S_{n_{\max}} = \{\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \cdots, \boldsymbol{x}_{i_{n_{\max}}}\}$

Algorithm 2 summarizes the FSSF-fr algorithm, including the concept just discussed. Since computing $\min_{\boldsymbol{y} \in \partial\Omega} d(\boldsymbol{x}_j, \boldsymbol{y})$ for any $\boldsymbol{x}_j \in \mathcal{C}$ is of $O(q)$, we can see Algorithm 2 is of the same computation time complexity with Algorithm 1. For $q = 2$, Figure 1.1b shows a typical design produced by the FSSF-fr algorithm at stage $n = 20$. Comparing this to the results of the FSSF-f algorithm in Figure 1.1a, the FSSF-fr algorithm selects fewer design points near the boundary. We defer more quantitative comparisons of the h_{max} and d_{min} performances and computational expense to Section 1.4.

1.2.3. The FSSF-b Algorithm

The FSSF-b algorithm is intended to have lower computational expense than the FSSF-f/fr algorithms for large N. We first give an overview of the major steps of the FSSF-b design algorithm with pseudo-code (Section 1.2.3.1), followed by computational and other technical details (Section 1.2.3.2).

1.2.3.1. Overview of the FSSF-b Algorithm. Our FSSF-b design algorithm removes points one-at-a-time from $\mathcal{C}$, so that the remaining points at each stage form a sequence of nested designs. Now, $\{i_N, i_{N-1}, \cdots, i_2, i_1\}$ is the sequence of indices of points removed in order (i.e., i_N is removed first, and i_1 is removed last). We still denote the design at stage-n by

$S_n = \{\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \cdots, \boldsymbol{x}_{i_n}\}$ and its index set by $I_n = \{i_1, i_2, \cdots, i_n\}$, so that the nested sequence of designs from size 1 to size n_{max} is $S_1 \subset S_2 \subset \cdots \subset S_{n_{max}} \subset \mathcal{C}$.

Given the candidate set $\mathcal{C}$, the main idea behind the FSSF-b design algorithm is the following. At each stage-n (sequentially, for $n = N-1, N-2, \cdots, 0$), given I_{n+1} and the corresponding design S_{n+1} from the previous stage-$(n+1)$, the goal is to find the index $i_{n+1} \in I_{n+1}$ of the next point $\boldsymbol{x}_{i_{n+1}} \in S_{n+1}$ to remove in order to form $S_n = S_{n+1} \setminus \boldsymbol{x}_{i_{n+1}}$. Our criterion is the maximin distance criterion (other criteria are discussed in Section 1.3), i.e., we seek

$$i_{n+1} = \arg\max_{i \in I_{n+1}} d_{\min}(S_{n+1} \setminus \boldsymbol{x}_i), \tag{1.3}$$

where $d_{\min}()$ is defined in Equation (1.1).

Algorithm 3 summarizes the FSSF-b algorithm. Notice that a naive search procedure in the optimization problems of Equation (1.3) and Equation (1.1) would result in excessive computational expense (even lager than for FSSF-f) when N is large, and N must generally be chosen large to achieve the best space-filling properties, especially when $n_{\max}$ is large. Section 1.2.3.2 discusses the major computational strategies that we use to substantially speed up the approach so that it is faster than FSSF-f for large N, which are based on fast nearest neighbor search algorithms and on appropriate choices of data structures. Full details of the algorithm are provided in Algorithm 3 Details in Appendix A.1.

Algorithm 3 Overview of FSSF-b Algorithm

Input: $n_{\max}$, N, q

1: Generate candidate set $\mathcal{C} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$ as a Sobol sequence

2: Initialize $I_N \leftarrow \{1, 2, \cdots, N\}$

3: Initialize $S_N \leftarrow \mathcal{C}$

4: **for** $n = N-1, N-2, \cdots, 1$ **do**

5: Find $i_{n+1} = \underset{i \in I_{n+1}}{\arg\max}\, d_{\min}(S_{n+1} \setminus \boldsymbol{x}_i)$

6: Update $I_n \leftarrow I_{n+1} \setminus i_{n+1}$

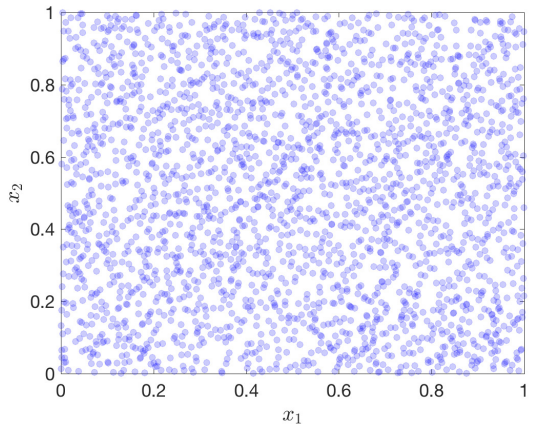
7: Update $S_n = S_{n+1} \setminus \boldsymbol{x}_{i_{n+1}}$

8: **end for**

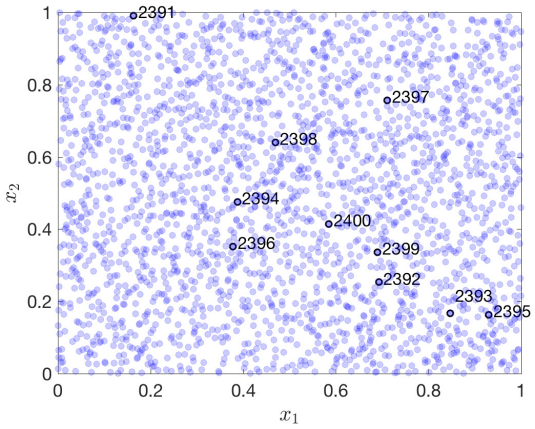
Output: Ordered FSSF design sequence $S_{n_{\max}} = \{\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \cdots, \boldsymbol{x}_{i_{n_{\max}}}\}$.

Before discussing the details of the algorithm, Figure 1.3 illustrates the operation of the FSSF-b algorithm when sequentially removing points to produce the design $S_{n_{max}} = \{\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \cdots, \boldsymbol{x}_{i_{n_{\max}}}\}$ for the case $n_{max} = 200$, $q = 2$ and $N = 1000q + 2n_{\max} = 2400$

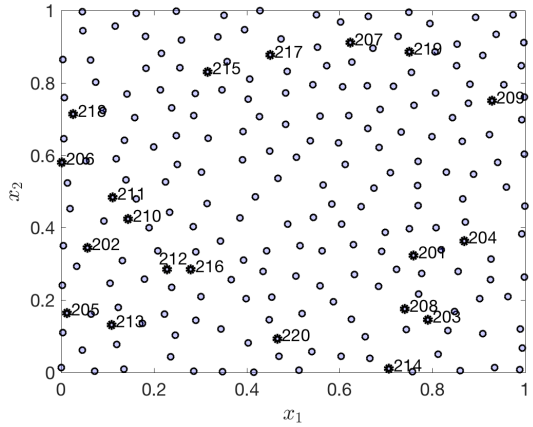
(the rule for choosing N is discussed in Section 1.3). Figure 1.3a shows the candidate set. Figure 1.3b shows the candidate set and the first 10 points that are removed (the point labeled 2400 was removed first, 2399 was removed second, etc.). Although it is difficult to discern from Figure 1.3b, the points having the closest neighbors are removed first. Figure 1.3c shows the last 220 candidate points after removing 2180 candidate points in total. Removing $\boldsymbol{x}_{i_{220}}, \boldsymbol{x}_{i_{219}}, \cdots, \boldsymbol{x}_{i_{201}}$ yields the final design S_{200}. From Figure 1.3c one sees that candidate points of which the nearest neighbor is closer to it gets removed earlier than points of which the nearest neighbor is further apart. Figure 1.3d shows the FSSF design $S_{200} = \{\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \cdots, \boldsymbol{x}_{i_{200}}\}$ and the first 20 design points in the nested design sequence. The latter are labeled in the order in which they are added to the design (i.e., the reverse order in which they were removed).



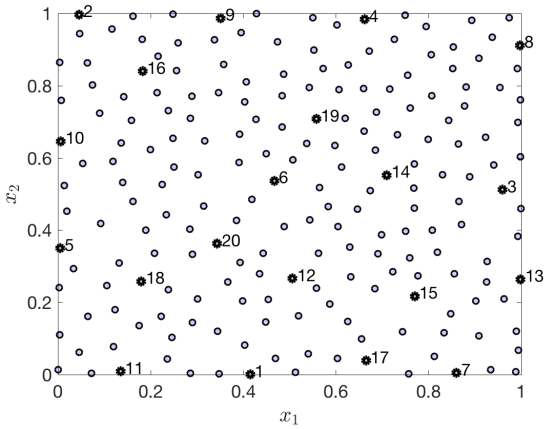
(a) Candidate set



(b) Candidate set showing the first 10 removed points



(c) Remaining FSSF-b points S_{220} at stage-220 with the sequence $\{\boldsymbol{x}_{i_{201}}, \boldsymbol{x}_{i_{202}}, \cdots, \boldsymbol{x}_{i_{220}}\}$ labeled



(d) Final FSSF-b design S_{200} with the sequence $\{\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \cdots, \boldsymbol{x}_{i_{20}}\}$ labeled

Figure 1.3. Illustration of the FSSF-b design algorithm operation for $n_{max} = 200$, $q = 2$ and $N = 1000q + 2n_{\max}$.

1.2.3.2. Computational Strategies and Algorithm Details. In order to drastically reduce computational expense of the optimization searches in Equation (1.3) and Equation (1.1), we use a number of computational strategies discussed in this section. First notice that if $\{i^1_{n+1}, i^2_{n+1}\} \triangleq \underset{\{i,j\} \in I_{n+1}}{\arg\min} d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ denotes the indices of the pair of nearest

neighbor points in S_{n+1} (or a pair, if the nearest neighbor pair is not unique), then it must be the case that $i_{n+1} \in \{i^1_{n+1}, i^2_{n+1}\}$. That is, the minimizer of Equation (1.3) is

$$i_{n+1} = \underset{i \in \{i^1_{n+1}, i^2_{n+1}\}}{\arg\max} \; d_{\min}(S_{n+1} \setminus \boldsymbol{x}_i). \tag{1.4}$$

Thus, the optimization goal in Equation (1.3) and Equation (1.1) reduces to finding the nearest neighbor pair $\{i^1_{n+1}, i^2_{n+1}\} \in I_{n+1}$ at each stage $n = N-1, N-2, \cdots, 1$, and then removing one of the two. It will often be the case that $d_{\min}(S_{n+1} \setminus \boldsymbol{x}_i)$ does not depend on whether $i_{n+1} = i^1_{n+1}$ or $i_{n+1} = i^2_{n+1}$, in which case either point from the pair $\{\boldsymbol{x}_{i^1_{n+1}}, \boldsymbol{x}_{i^2_{n+1}}\}$ can be removed. Regardless, the criterion is always optimized if we remove from the pair the point whose distance to its second nearest neighbor in S_{n+1} is the smaller. If the nearest neighbor pair $\{i^1_{n+1}, i^2_{n+1}\}$ is not unique (i.e, if multiple pairs of points in S_{n+1} have the same minimum pairwise distance), then we also remove the point from among all the nearest neighbor pairs, whose distance to its second nearest neighbor in S_{n+1} is the smallest.

On the surface, it may appear that in order to find the nearest neighbor pair $\{i^1_{n+1}, i^2_{n+1}\}$ at each stage $n = N-1, N-2, \cdots, 1$, it requires finding the $N \times N$ distance matrix, which would be computationally prohibitive. However, this is unnecessary, and a much faster K-nearest neighbors (K-NN) algorithm (Hickernell, Bentley, and Finkel 1977) can be used up front at stage N, before the algorithm begins removing points. We use a k-d tree based algorithm, which is particularly fast (computational expense details are discussed in Section 1.4.3). K-NN algorithms find only the K nearest neighbors for each point in $\mathcal{C}$, for some suitably chosen K. Our strategy is to choose K relatively small (in the FSSF R package, we use $K = 20$ if $q < 8$ and $K = 40$ otherwise) and then rerun

K-NN on the remaining points at some intermediate stage(s), if needed, which we discuss in more details below. In general, choosing K too large incurs unnecessary computational expense and computer memory issues, whereas choosing K too small incurs the risk of having to rerun K-NN search too frequently, which would not be efficient.

We now discuss our strategy for using and updating the nearest neighbors information on the remaining points, when implementing Equation (1.4) to determine which point to remove at each stage. We represent the nearest neighbor information via the two arrays $\boldsymbol{D}$ and $\boldsymbol{J}$, which which are initially of size $N \times K$ and contain the distances and indices, respectively, of the K nearest neighbors of each of the N points. That is, the jth row, ith column elements of $\boldsymbol{J}$ and $\boldsymbol{D}$ (denoted by J_{ji} and D_{ji}, respectively) are the index of the ith nearest neighbor of $\boldsymbol{x}_j$ and the distance between this neighbor and $\boldsymbol{x}_j$, respectively. The rows and cells of $\boldsymbol{D}$ and $\boldsymbol{J}$ are updated each stage as points are removed, to reflect the nearest neighbor structure of the remaining points.

We illustrate the concepts with the following simple toy example for $q = 1$, $N = 10$, $K = 3$, $n_{max} = 5$, and candidate set $\mathcal{C} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_{10}\} = \{0.40, 0.68, 0.14, 0.31, 0.79, 0.94, 0.43, 0.83, 0.69, 0.37\}$. Table 1.1 and Table 1.2 show the initial $\boldsymbol{D}$ and $\boldsymbol{J}$ for the candidate set $\mathcal{C}$. By scanning the first column of $\boldsymbol{D}$ to find its smallest element and then finding the corresponding element of $\boldsymbol{J}$, the nearest neighbor index pair at stage $n = 9$ is identified as $\{i_{10}^1, i_{10}^2\} = \{2, 9\}$. Since $\boldsymbol{x}_9$'s second-closest neighbor (which is $\boldsymbol{x}_5$, at a distance of 0.10) is closer than $\boldsymbol{x}_2$'s second-closest neighbor (which is also $\boldsymbol{x}_5$, at a distance of 0.11), we have $i_{10} = 9$, and the first point removed is $\boldsymbol{x}_9$, so that $S_9 = \mathcal{C} \setminus x_9$. Its row is shaded in Tables 1.1 and 1.2 to indicate that it will be ignored when searching for nearest neighbors in subsequent stages. All other cells corresponding to points for which $\boldsymbol{x}_9$ was

among the three nearest neighbors are also shaded, since they must also be ignored in subsequent stages. The deletion of an entire row, as opposed to only a cell or set of cells, is indicated in Tables 1.1 to 1.6 by shading the entire row, including its first cell (containing the label for the point).

Tables 1.1 and 1.2 show the updated $\boldsymbol{D}$ and $\boldsymbol{J}$ at the end of the first stage $n = 9$. Notice that when searching for the nearest neighbor pair in stage $n = 8$, the matrices in Tables 1.1 and 1.2 produced at the end of the previous stage must be searched, excluding any shaded cells and rows. This action can be accelerated because the nearest neighbor for each remaining point is the first nonshaded cell in its row. Hence, to find the nearest neighbor pair at each stage, we only need to search the first column of the $\boldsymbol{D}$ array, assuming that in the coded implementation, the shaded cells are deleted and the remaining cells in each row are shifted to the left. From searching the nonshaded entries of Tables 1.1 and 1.2, the nearest neighbor index pairs at stage $n = 8$ are $\{i_9^1, i_9^2\} = \{1, 7\}$ and $\{i_9^1, i_9^2\} = \{1, 10\}$, which both share the same minimum distance 0.03. Of the three points $\{\boldsymbol{x}_1, \boldsymbol{x}_7, \boldsymbol{x}_{10}\}$, the one having the closest second-nearest neighbor is $\boldsymbol{x}_1$ (at a distance of 0.03 from $\boldsymbol{x}_{10}$, versus distances of 0.06 between the other two points and their second-nearest neighbors). Hence, $i_9 = 1$, and the second point removed is $\boldsymbol{x}_1$. The shaded rows and cells in Tables 1.3 and 1.4 indicate those associated with the two removed points $\{\boldsymbol{x}_9, \boldsymbol{x}_1\}$, which will be ignored when searching for nearest neighbors in subsequent stages.

By searching Table 1.3 and Table 1.4 similarly at stage $n = 7$, we have $\{i_8^1, i_8^2\} = \{5, 8\}$. This pair coincidentally have the same distance to their second-nearest neighbors, and so we arbitrarily remove the first point $\boldsymbol{x}_5$. The remaining points and their corresponding unshaded cells are shown in Table 1.5 and Table 1.6. At this point, we must re-do K-NN,

because there exists remaining points ($\{x_2, x_6, x_8\}$ in this case) that have only a single nonshaded cell in their row, which means their second-nearest neighbors (which are used to determine which one of the nearest-neighbor points are removed) are unavailable in the current K-NN results. In general, our strategy is to re-do K-NN on S_n after a stage n whenever the second-nearest neighbor of any remaining point in S_n is not contained in the table produced by the previous application of K-NN. With our choice of K (20 if $q < 8$ and K otherwise), we typically must re-do K-NN 3 to 5 times to produce one design. Notice that each time K-NN is re-done it is for substantially fewer points than the original N candidate points, and hence it is computationally faster than the initial K-NN on the entire candidate set.

Tables 1.7 and 1.8 show the updated $\boldsymbol{D}$ and $\boldsymbol{J}$ arrays at the end of stage $n = 7$ after re-doing K-NN on the remaining points in S_7. At the beginning of $n = 6$, there are two nearest neighbor pairs $\{i_7^1, i_7^2\} = \{4, 10\}$ and $\{i_7^1, i_7^2\} = \{7, 10\}$ having the same distance 0.06, of which point $\boldsymbol{x}_{10}$ was removed according to our second nearest neighbor rule. This sequential procedure is continued until only a single pair remains, re-doing K-NN whenever needed, as described above. In this example, the remain points are removed in the order of $\{\boldsymbol{x}_8, \boldsymbol{x}_4, \boldsymbol{x}_2, \boldsymbol{x}_7, \boldsymbol{x}_6, \boldsymbol{x}_3\}$ with the last two removed in arbitrary order. Thus, the ordered sequence of indices is $\{i_1, i_2, \cdots, i_{10}\} = \{3, 6, 7, 2, 4, 8, 10, 5, 1, 9\}$, so that the largest desired FSSF design is $S_{n_{max}} = \{\boldsymbol{x}_3, \boldsymbol{x}_6, \boldsymbol{x}_7, \boldsymbol{x}_2, \boldsymbol{x}_4\}$, and the order in which we have listed the points in $S_{n_{max}}$ indicates the entire FSSF design sequence up to size n_{max}.

point index j	D_{j1}	D_{j2}	D_{j3}
1	0.03	0.03	0.09
2	0.01	0.11	0.15
3	0.17	0.23	0.26
4	0.06	0.09	0.12
5	0.04	0.10	0.11
6	0.11	0.15	0.25
7	0.03	0.06	0.12
8	0.04	0.11	0.14
9	0.01	0.10	0.14
10	0.03	0.06	0.06

Table 1.1. Array $\boldsymbol{D}$ at end of stage $n = 9$

point index j	J_{j1}	J_{j2}	J_{j3}
1	7	10	4
2	9	5	8
3	4	10	1
4	10	1	7
5	8	9	2
6	8	5	9
7	1	10	4
8	5	6	9
9	2	5	8
10	1	7	4

Table 1.2. Array $\boldsymbol{J}$ at end of stage $n = 9$

point index j	D_{j1}	D_{j2}	D_{j3}
1	0.03	0.03	0.09
2	0.01	0.11	0.15
3	0.17	0.23	0.26
4	0.06	0.09	0.12
5	0.04	0.10	0.11
6	0.11	0.15	0.25
7	0.03	0.06	0.12
8	0.04	0.11	0.14
9	0.01	0.10	0.14
10	0.03	0.06	0.06

Table 1.3. Array $\boldsymbol{D}$ at end of stage $n = 8$

point index j	J_{j1}	J_{j2}	J_{j3}
1	7	10	4
2	9	5	8
3	4	10	1
4	10	1	7
5	8	9	2
6	8	5	9
7	1	10	4
8	5	6	9
9	2	5	8
10	1	7	4

Table 1.4. Array $\boldsymbol{J}$ at end of stage $n = 8$

point index j	D_{j1}	D_{j2}	D_{j3}
1	0.03	0.03	0.09
2	0.01	0.11	0.15
3	0.17	0.23	0.26
4	0.06	0.09	0.12
5	0.04	0.10	0.11
6	0.11	0.15	0.25
7	0.03	0.06	0.12
8	0.04	0.11	0.14
9	0.01	0.10	0.14
10	0.03	0.06	0.06

Table 1.5. Array $\boldsymbol{D}$ at end of stage $n = 7$

point index j	J_{j1}	J_{j2}	J_{j3}
1	7	10	4
2	9	5	8
3	4	10	1
4	10	1	7
5	8	9	2
6	8	5	9
7	1	10	4
8	5	6	9
9	2	5	8
10	1	7	4

Table 1.6. Array $\boldsymbol{J}$ at end of stage $n = 7$

point index j	D_{j1}	D_{j2}	D_{j3}
2	0.15	0.25	0.26
3	0.17	0.23	0.29
4	0.06	0.12	0.17
6	0.11	0.26	0.51
7	0.06	0.12	0.25
8	0.11	0.15	0.40
10	0.06	0.06	0.23

Table 1.7. Array $\boldsymbol{D}$ at end of stage $n = 7$, after re-doing K-NN

point index j	J_{j1}	J_{j2}	J_{j3}
2	8	7	6
3	4	10	7
4	10	7	3
6	8	2	7
7	10	4	2
8	6	2	7
10	7	4	3

Table 1.8. Array $\boldsymbol{J}$ at end of stage $n = 7$, after re-doing K-NN

1.3. Choice of Candidate Set and Design Criterion

For all of our FSSF algorithms (FSSF algorithms refer to FSSF-f (Algorithm 1), FSSF-fr (Algorithm 2), and FSSF-b (Algorithm 3) algorithms), we have used Sobol sequences for the candidate set $\mathcal{C}$ and a maximin-like criterion when selecting or removing points from $\mathcal{C}$. In this section we discuss the rationale behind these choices, compared with alternatives for $\mathcal{C}$ and for space-filling (and briefly for other non-distance-based) criteria, and we also discuss the choice of N.

1.3.1. Choice of Candidate Set

The general requirement for $\mathcal{C}$ is that it should cover Ω as evenly and densely as possible, without leaving large holes. We have investigated both Sobol sequences and LHDs optimized with respect to some space-filling criteria (S. Ba, Myers, and Brenneman 2015; Rennen et al. 2010). We compared the space-filling performances of the FSSF design sequences resulting from these two choices of $\mathcal{C}$ (results are not shown, for brevity), and we did not observe any significant performance differences. Because Sobol sequences are much faster to generate than optimized LHDs, we henceforth use Sobol sequences exclusively as the candidate set. By themselves, Sobol sequences are undesirable as space-filling designs for deterministic computer experiments, because they tend to place some points very close to each other (see, e.g., Figure 1.7c). However, when used as the candidate set in the FSSF algorithms, this drawback disappears. For instance, the backward removal portion of the FSSF-b algorithm removes points that are too closely spaced.

1.3.2. Choice of N

After q and $n_{\max}$ are specified, one must also choose N. Using larger N generally provides better space-filling performance of the FSSF design, but the obvious drawback of larger N is increased computational expense. We have found that for a fixed q and $n_{\max}$, when N increases beyond a certain value there is little further improvement in the space-filling properties. Therefore, our rule is to choose N as small as possible (for computational reasons), but just larger than the value after which there is little further space-filling performance improvement. This value depends on q and $n_{\max}$, and through extensive investigations we have found that using $N \geq 1000q + 2n_{\max}$ usually suffices. As a default value, we use $N = 1000q + 2n_{\max}$ in our FSSF R package, which was used in all the examples in this chapter.

1.3.3. Choice of Space-filling Criterion

We have also investigated different space-filling criteria for sequentially adding (FSSF-f/fr) or removing (FSSF-b) points, analogous to Equation (1.1). As discussed earlier Equation (1.2) is not a practical design criterion because of excessive computational expense. In addition to the minimum pairwise distance criterion (Equation (1.1)), we have also investigated the average reciprocal distance criterion (Morris and Mitchell 1995) defined as follows. The average reciprocal distance of a set $S_n = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\}$ is

$$\Big[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{1}{\Big(d(\boldsymbol{x}_i,\boldsymbol{x}_j)\Big)^p}\Big]^{\frac{1}{p}}, \tag{1.5}$$

where p is some positive integer. The average reciprocal distance criterion with $p = 1$ is sometimes referred to as the energy criterion (V. R. Joseph et al. 2015). We chose to use the minimum pairwise distance criterion, instead of an average reciprocal distance criterion, for the FSSF algorithms because the latter can lead to designs that are not suitably space-filling. Figure 1.4 demonstrates this by comparing designs from the FSSF-b algorithm using the minimum pairwise distance criterion (Figure 1.4a) and the average reciprocal distance criterion with $p = 1$ (Figure 1.4b). For the designs in both Figures 1.4a and 1.4b, $q = 2$, $n_{\max} = 50$ and the candidate set is Sobol sequence with $N = 1000q + 2n_{\max}$. The FSSF-b design using the average reciprocal distance criterion has much poorer space-filling behavior with too many design points located on the boundary of the design space Ω. This undesirable characteristic of the average reciprocal distance criterion was also observed in Currin et al. 1988. Other choices of p could lead to better space-filling property. However, the best choice of p is not clear, and this may depend on q. Moreover, minimizing the average reciprocal distance would be substantially more computationally expensive than maximizing the minimum pairwise distance criterion (Equation (1.1)). In light of this, we have chosen the computationally simpler and parameter-free criterion Equation (1.1), which turned out to work quite well in our performance comparisons (see Section 1.4).

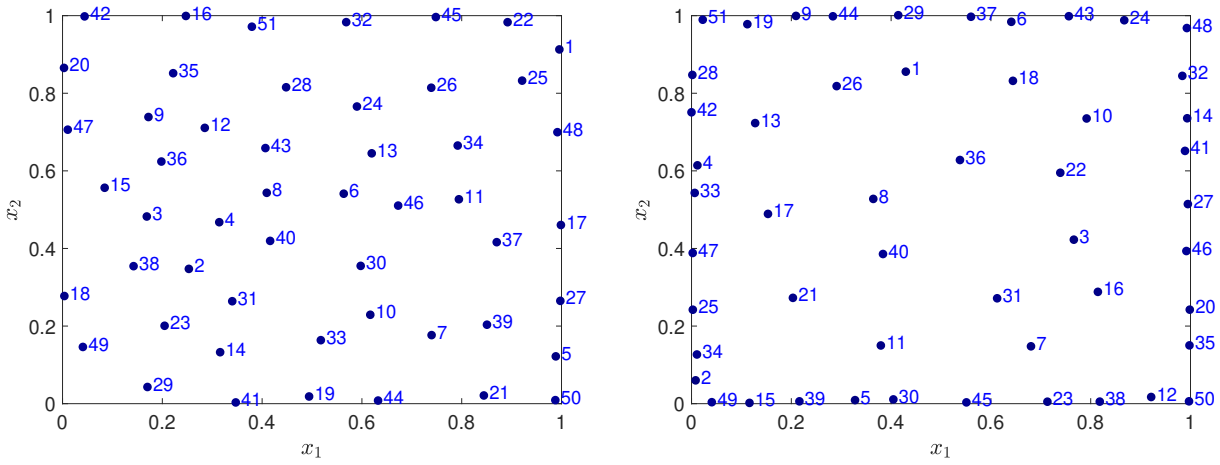


Figure 1.4. Example FSSF-b designs comparing (a) the minimum pairwise distance criterion versus (b) the average reciprocal distance criterion with $p = 1$. The numbers beside each point indicate the order in which the points are added in the final FSSF-b design sequence.

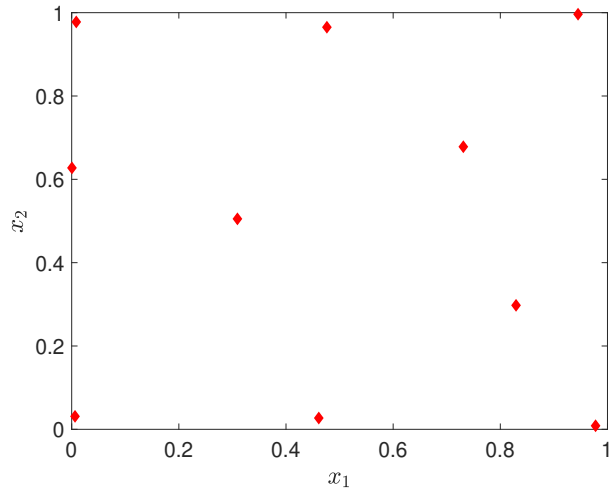
1.4. Sequential Space-Filling Performance Comparisons

We first (Section 1.4.1) illustrate qualitatively the sequential space-filling behavior of the FSSF design sequences (FSSF-f, FSSF-fr, FSSF-b) relative to each other and relative to alternatives including the standard non-sequential maximin LHD (S. Ba, Myers, and Brenneman 2015), optimized SLHD (S. Ba, Myers, and Brenneman 2015), multi-layer NLHD (P. Qian 2009), and Sobol sequence (Sobol 1967). For this, we plot the produced designs at low ($q = 2$) dimension. We then provide more extensive numerical comparisons of space-filling performances (Section 1.4.2) and computation times (Section 1.4.3) of the various FSSF and alternative methods. We note that the standard non-sequential maximin LHD is included only as a benchmark, since it is not a sequential method; and the SLHD and NLHD methods are block-sequential and not one-at-a-time sequential.

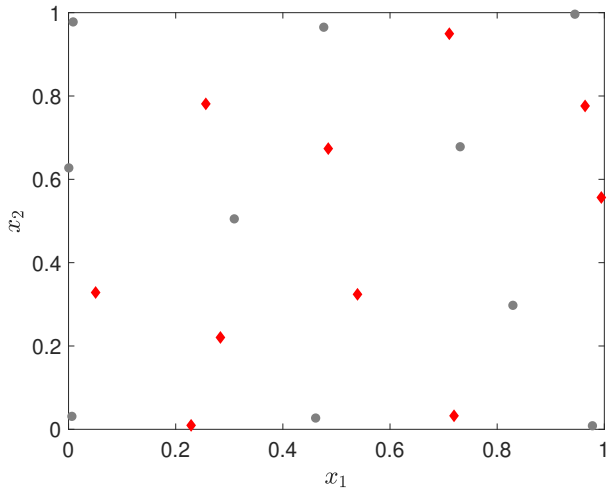
1.4.1. Qualitative Comparisons via Example Designs

Figures 1.5 and 1.6 show typical design sequence examples for the FSSF-b and FSSF-fr algorithms, respectively, for $q = 2$, $n_{\max} = 320$, and $N = 1000q + 2n_{\max}$ at stages $n = 10, 20, 40, 80, 160,$
and 320. Typical results for the FSSF-f algorithm are very similar to those shown in Figure 1.5 for the FSSF-b algorithm and are omitted for brevity. In each panel, the red diamonds are the additional points added to the smaller nested design shown in the previous panel, and the latter is indicated by gray circles. For example, in Figure 1.5d the gray circles represent the same FSSF design S_{40} shown in Figure 1.5c, and the red diamonds are the additional 40 points that, together with the gray S_{40} circles, constitute the FSSF design S_{80}. Although the designs are only shown as batch-sequential in Figures 1.5 and 1.6, the sample points were actually chosen fully-sequentially (i.e., one-at-a-time).

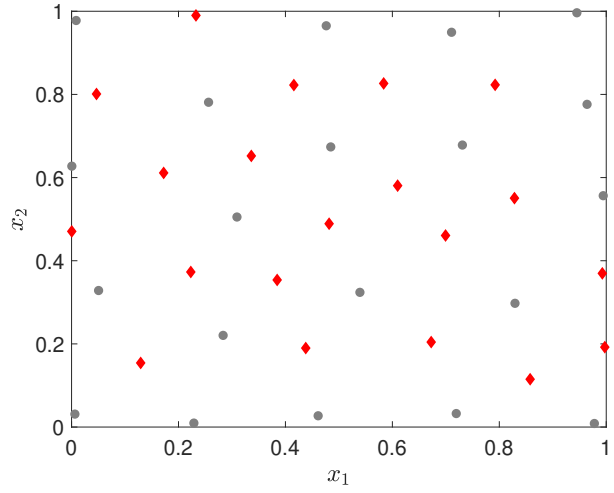
Visually, the design sequences in Figures 1.5 and 1.6 have good space-filling properties at each stage $n = 10, 20, 40, 80, 160, 320$, and the design sequences tend to fill the design space gradually and evenly as sample size increases. Moreover, as intended, the FSSF-fr designs avoid selecting points too close to the boundary, especially for smaller n, whereas the FSSF-b (and FSSF-f) algorithms tend to do this (see for example Figures 1.5a and 1.6a). As demonstrated more quantitatively in the next section, this generally results in better h_{max} performance for the FSSF-fr designs, at the expense of worse d_{min} performance.



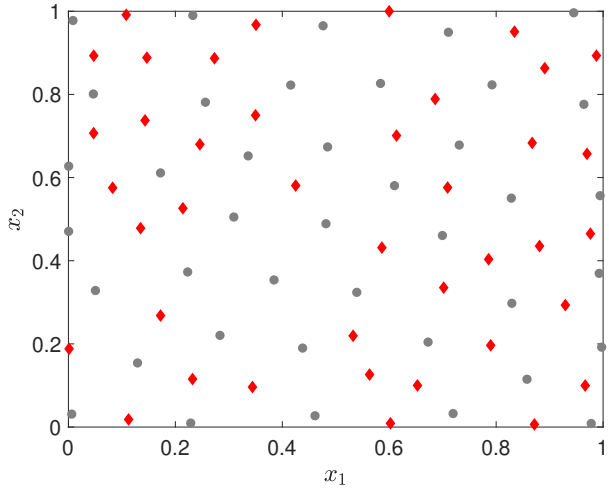
(a) $n = 10$



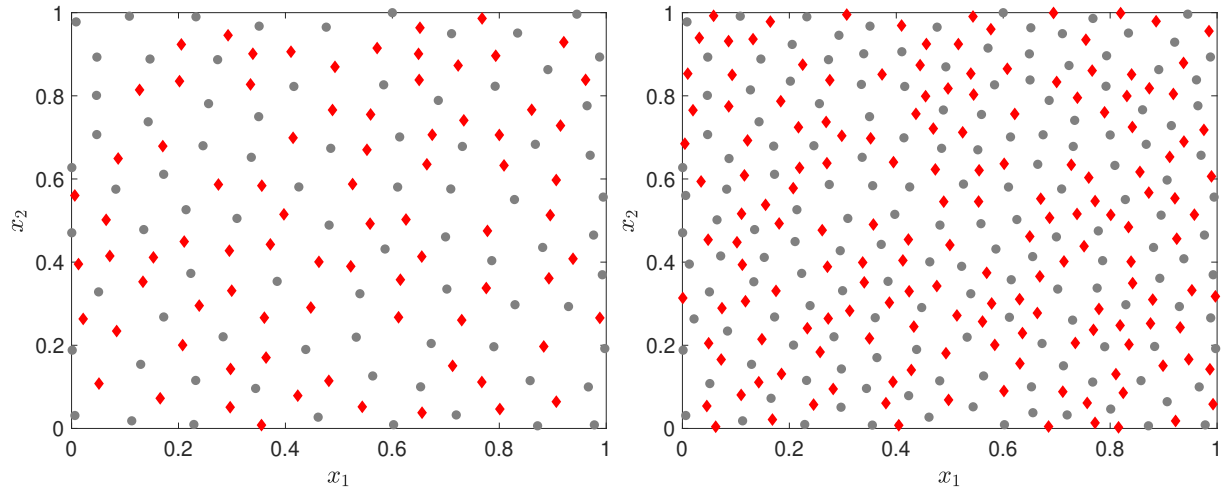
(b) $n = 20$



(c) $n = 40$

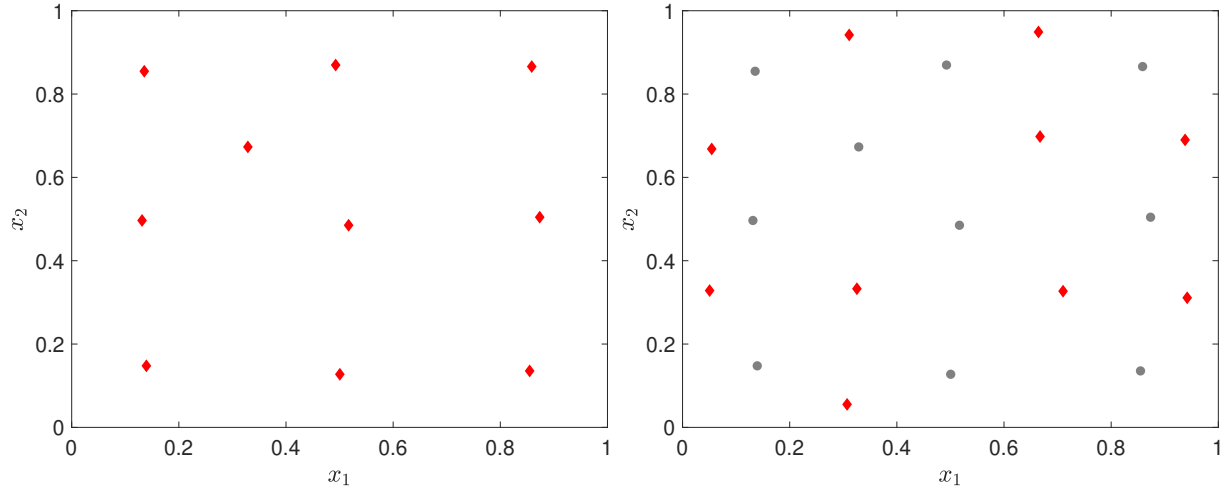


(d) $n = 80$

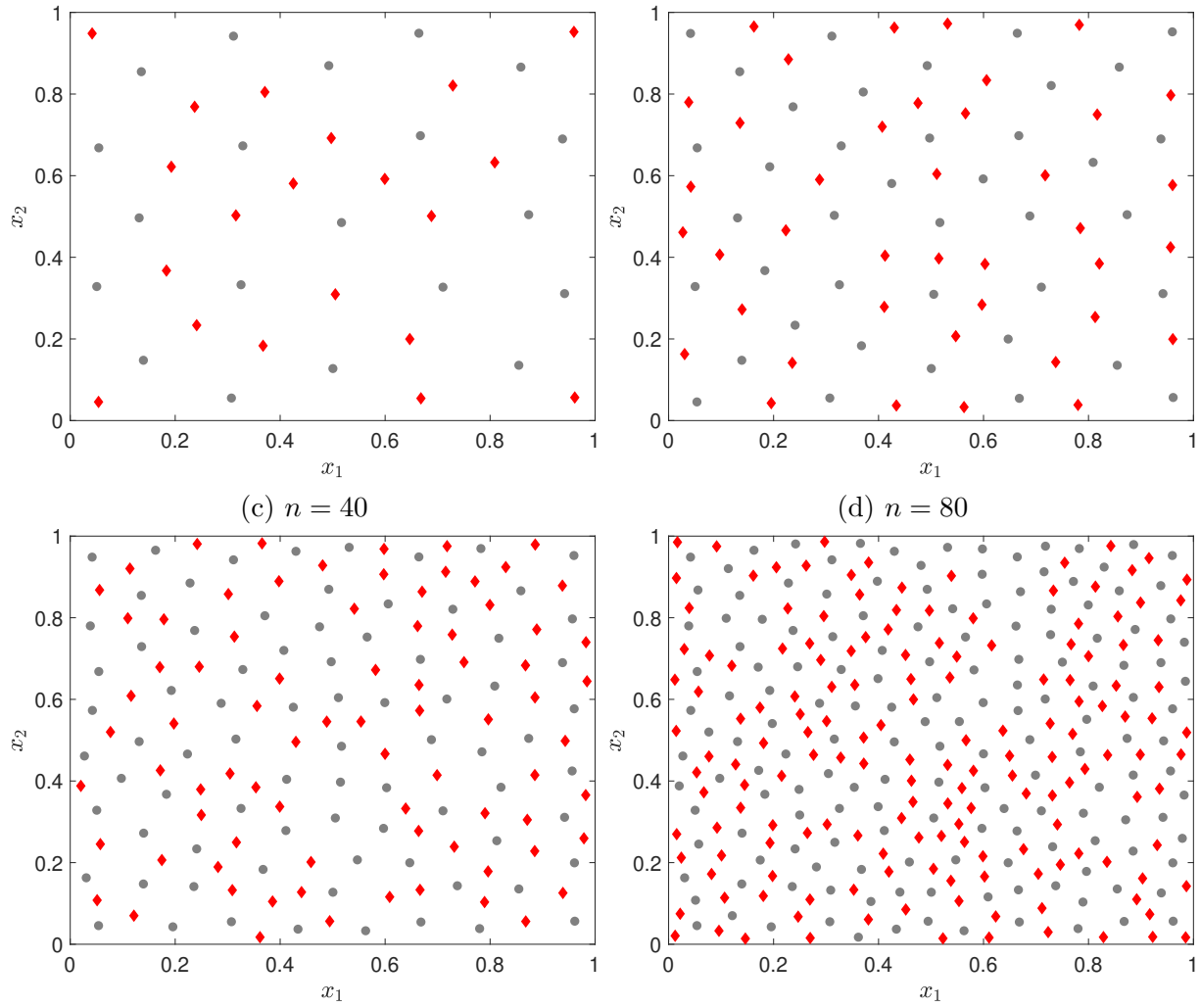


(e) $n = 160$ (f) $n = 320$

Figure 1.5. Qualitative visual illustration of the space-filling behavior of a typical FSSF-b design sequence for $q = 2$ and $n_{\max} = 320$, showing designs from the nested FSSF sequence at sizes $n = 10, 20, 40, 80, 160, 320$. The gray circles in each panel are the smaller nested design from the previous panel, and the red diamonds are the additional points added to the smaller design to comprise the larger design.



(a) $n = 10$ (b) $n = 20$



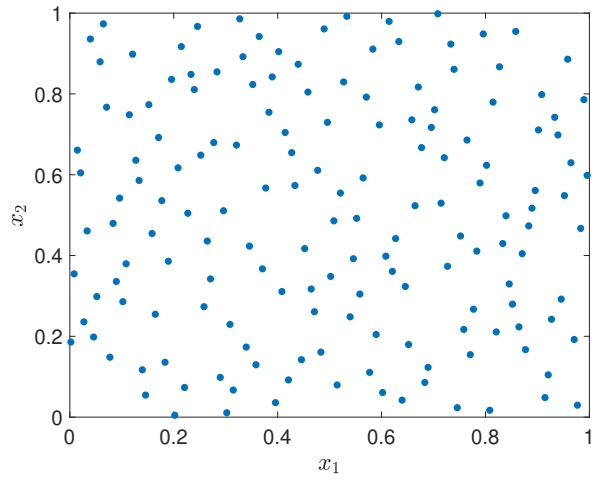
(c) $n = 40$ (d) $n = 80$

(e) $n = 160$ (f) $n = 320$

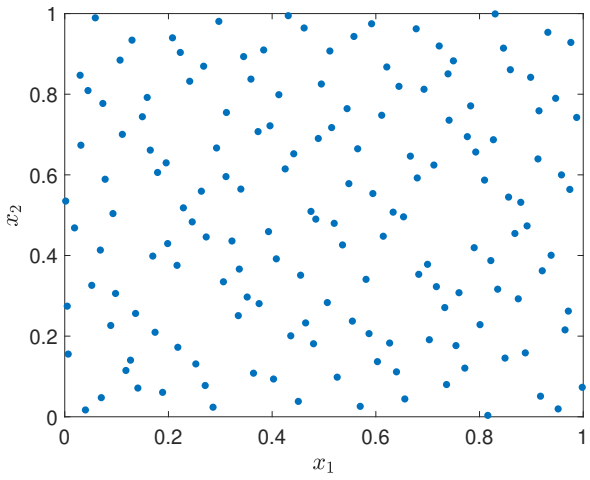
Figure 1.6. Qualitative visual illustration of the space-filling behavior of a typical FSSF-fr design sequence for $q = 2$ and $n_{\max} = 320$, showing designs from the nested FSSF sequence at sizes $n = 10, 20, 40, 80, 160, 320$. The gray circles in each panel are the smaller nested design from the previous panel, and the red diamonds are the additional points added to the smaller design to comprise the larger design.

Figure 1.7 visually compares the space-filling performances of typical realizations of a FSSF-b design, a FSSF-f design, a FSSF-fr design, a NLHD of P. Qian 2009, a Sobol

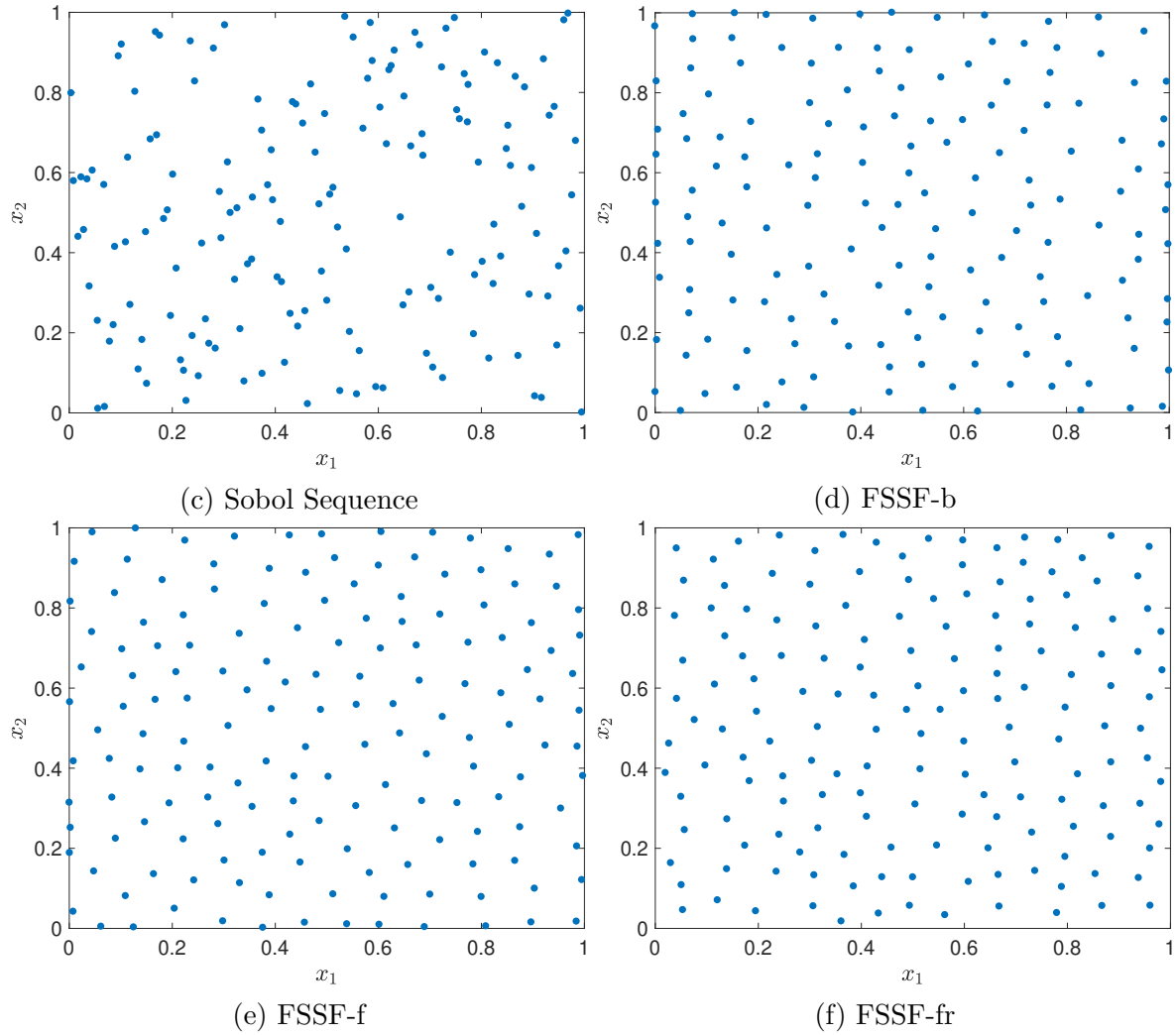
sequence, and a one-shot (non-sequential) maximin LHD of S. Ba, Myers, and Brenneman 2015 (using 1000 iterations; using more iterations may lead to better space-filling performance at the cost of computation efficiency) for $n = n_{\max} = 160$ and $q = 2$. As in all the examples in this chapter, we have used $N = 1000q + 2n_{\max}$. The FSSF designs (Figures 1.7d to 1.7f) clearly have visually superior space-filling behavior than other designs, which is consistent with the numerically-superior performances demonstrated in the next section. One should keep in mind that among the six designs in Figure 1.7, only the FSSF designs and Sobol sequence are fully-sequential. The NLHD is batch-sequential (we used batches of size $20, 40, 80, 160$), and the one-shot maximin LHD is not sequential at all.



(a) One-shot maximin LHD



(b) NLHD



(c) Sobol Sequence (d) FSSF-b

(e) FSSF-f (f) FSSF-fr

Figure 1.7. Comparison of typical design results for six different methods at size $n = n_{\max} = 160$. The FSSF designs appear more evenly space-filling.

1.4.2. Numerical Performance Comparisons in Terms of d_{min} and h_{max}

Figure 1.8 shows the performances of FSSF-b, FSSF-f, FSSF-fr, Sobol sequence, NLHD, SLHD with slice sizes 2 and 10, and one-shot maximin LHD, in terms of the minimum pairwise distance criterion d_{min} (defined in Equation (1.1), the larger the better) and the

maximum hole size criterion h_{max} (defined in Equation (1.2), the smaller the better) in dimensions $q = 2, 8, 16$ respectively. The d_{min} and h_{max} values plotted in Figure 1.8 were average values across 10 independent replicates. The maximum hole size criterion was computed by generating 10^6 test points from a uniform distribution over Ω, calculating the minimum distance between each test point and the design points in S_n, and then calculating h_{max} as the maximum (across all test points) of these minimum distances. The FSSF designs, used $n_{\max} = 320$ for $q = 2, 8, 16$ as well as $n_{\max} = 1280$ for $q = 16$, with the candidate set being a Sobol sequence of size $N = 1000q + 2n_{\max}$. The NLHDs were generated using batch sizes $20, 40, 80, 160, 320$ for the first three rows in Figure 1.8 and batch sizes $20, 40, 80, 160, 320, 640, 1280$ for the last row in Figure 1.8. The one-shot maximin LHDs and SLHDs were generated using the package provided by S. Ba, Myers, and Brenneman 2015 with 1000 iterations. We have again included the one-shot LHD only as a benchmark, since it is not sequential. The "One-shot LHD" curves plotted in each panel are not truly a nested sequence, but rather a set of independently generated designs of size $20, 40, 80, 160, 320$ for the first three row in Figure 1.8 and of size $20, 40, 80, 160, 320, 640, 1280$ for the last row in Figure 1.8. Moreover, when comparing performance results, readers should keep in mind that the NLHD and SLHD designs are not fully-sequential; for instance, NLHD did not produce designs of sizes in between the values $20, 40, 80, 160, 320$.

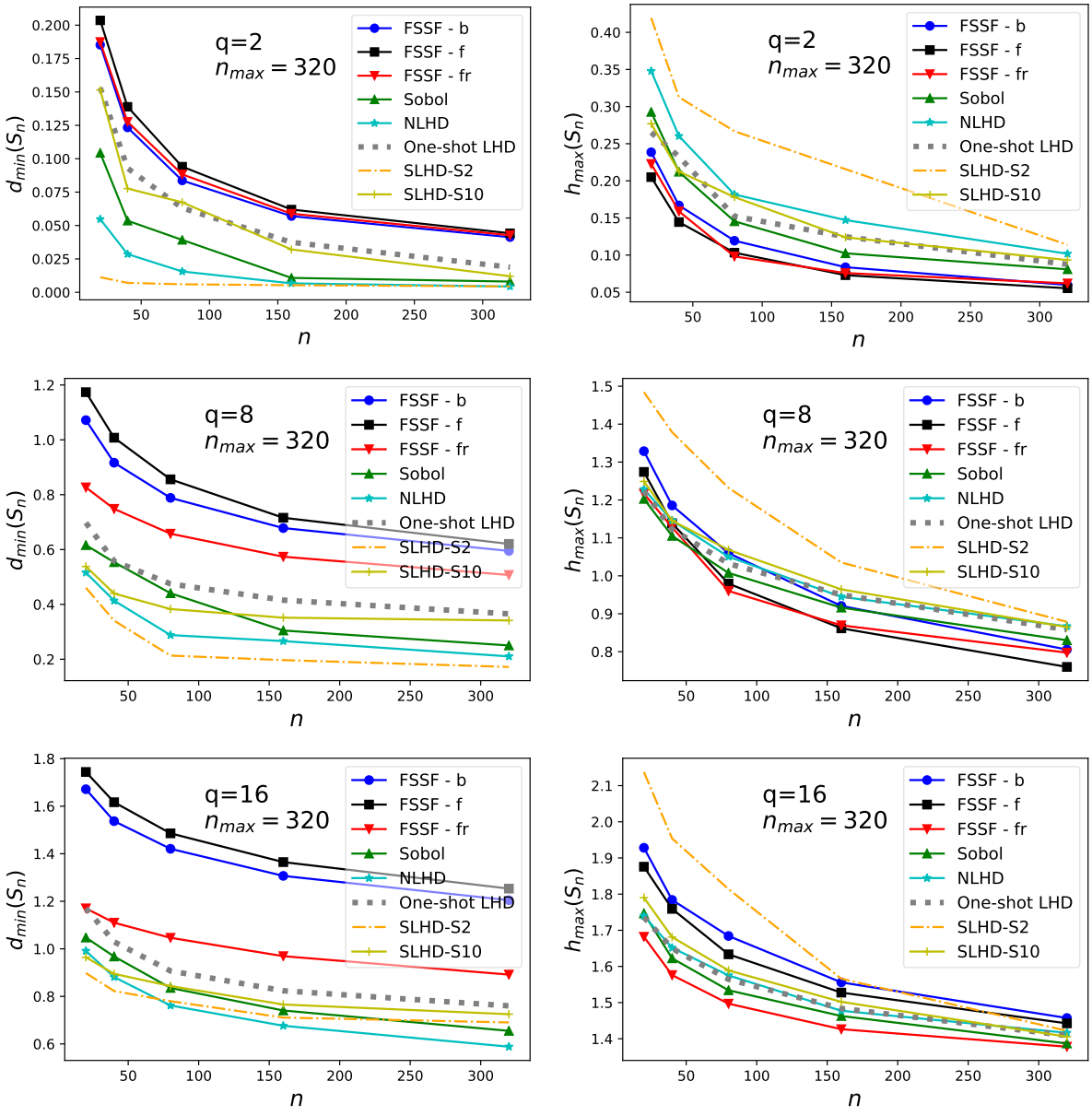
Comparing the FSSF algorithms against the benchmark (one-shot maximin LHD) and other designs, we draw the following conclusions based on the results shown in Figure 1.8. First, all FSSF designs performed better than the benchmark (the one-shot maximin LHD) in terms of the $d_{\min}$ criterion (Equation (1.1)) for all tested dimensions $q = 2, 8, 16$

and sample sizes n. In contrast, the NLHD and Sobol sequences performed consistently worse than the benchmark for all tested q and n values (see the four figures in the left panel of Figure 1.8). The SLHD with slice size 10 performed comparably with the benchmark, whereas the SLHD with slice size 2 consistently performed worse than the benchmark. Note that the SLHD with smaller slice size is closer to being fully sequential like our FSSF algorithm, whereas with larger slice size it is only block-sequential. Figure 1.8 shows that the space-filling performance of SLHD degrades as slice size decreases and it becomes closer to fully sequential. In addition, the FSSF designs are superior to the benchmark in terms of $d_{\min}$ not only for small values of n but also for relatively large ones. For instance, in the bottom-left panel of Figure 1.8, the $d_{\min}$ values for the FSSF-b and FSSF-fr design are larger than the benchmark $d_{\min}$ value by around 0.6 for all n values ranging from 20 to 1280.

The performances in terms of $h_{\max}$ of Equation (1.2) are somewhat different. The FSSF designs show better (smaller) $h_{\max}$ than the benchmark when n is beyond a reasonably large value. For example, when $q = 2$, all the FSSF designs have better $h_{\max}$ values than the benchmark for all tested n values (see the top-right panel of Figure 1.8). For $q = 8$, all the FSSF designs performed consistently better than the benchmark when n is larger than around 100 (see the middle-right panel of Figure 1.8). The FSSF-f and FSSF-fr designs were consistently better than the benchmark for n larger than around 50. For $q = 16$, the the FSSF-f and FSSF-b designs had worse $h_{\max}$ than the benchmark (but better $d_{\min}$), whereas the FSSF-fr designs had better $h_{\max}$ and $d_{\min}$ for all n values. In contrast, relative to the benchmark, the NLHD and Sobol sequence had comparable $h_{\max}$ but substantially worse $d_{\min}$ for all q and n values considered. The SLHD performance

relative to the benchmark is similar to what was found for the $d_{\min}$ criterion. SLHD with slice size 10 performed comparably with the benchmark but is only batch-sequential. In contrast, SLHD with slice size 2 is closer to fully-sequential but has $h_{\max}$ performance that is much worse than the benchmark.

Figure 1.8 also confirms that the three FSSF designs (FSSF-f, FSSF-fr and FSSF-b) have different relative advantages. The FSSF-f and FSSF-b designs generally have better (larger) $d_{\min}$ values than the FSSF-fr designs and substantially better $d_{\min}$ values than the other designs considered, and the difference is more pronounced in higher dimensions. The FSSF-f designs performed slightly better than FSSF-b designs in terms of $d_{\min}$, but as will be shown in Section 1.4.3, FSSF-b can be orders of magnitude faster than FSSF-f. As expected, the space-filling performance of FSSF-f and FSSF-b relative to FSSF-fr, is reversed for the $h_{\max}$ criterion, since the FSSF-fr algorithm was designed to have better $h_{\max}$ at the expense of worse $d_{\min}$. The FSSF-fr designs have better (smaller) $h_{\max}$ values than the FSSF-f and FSSF-b designs, especially in higher dimensions. In lower dimensions ($q \leq 8$) the $h_{\max}$ performances of the FSSF-fr, FSSF-f, and FSSF-b designs are all roughly comparable.


q=2
$n_{max} = 320$
$d_{min}(S_n)$
$h_{max}(S_n)$
q=8
$n_{max} = 320$
$d_{min}(S_n)$
$h_{max}(S_n)$
q=16
$n_{max} = 320$
$d_{min}(S_n)$
$h_{max}(S_n)$
FSSF - b
FSSF - f
FSSF - fr
Sobol
NLHD
One-shot LHD
SLHD-S2
SLHD-S10
n

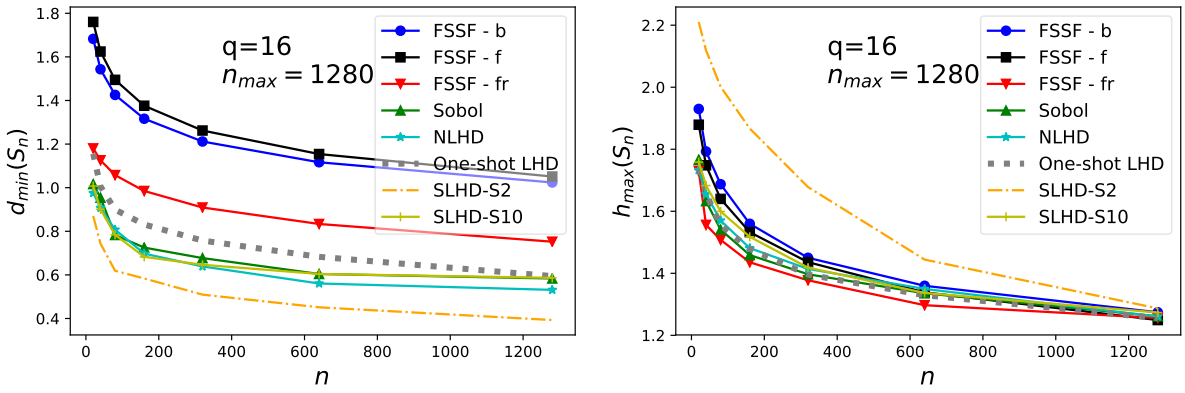


Figure 1.8. Comparisons of d_{min} (left column) and h_{max} (right column) space-filling performances of eight algorithms (FSSF-f, FSSF-fr, FSSF-b, NLHD, Sobol sequence, one-shot LHD, SLHD with slice size 2, and SLHD with slice size 10).

1.4.3. Runtime Comparisons

In this section we analyze the theoretical time complexity and memory complexity of the FSSF algorithms and also investigate their computation time in practice. We used a computer with 16GB memory and 3.3 GHz Intel Core i7 processor to test the computation time.

In short, the FSSF-b algorithm has better time complexity than FSSF-f/fr algorithms, $O(qN \log N)$ as opposed to $O(qN^2)$. It is well-known that by utilizing the k-d tree data structure (Bentley 1975), the time complexity of the K-NN algorithm for finding the K nearest neighbors of every point in a set of size N in dimension q is $O(qN \log N)$. By using the computational strategies discussed in Section 1.2.3.2 and choosing appropriate data structures and strategies as described in Algorithm 3 Details in Appendix A.1, the theoretical time complexity of the FSSF-b algorithm is also $O(qN \log N)$. For example, when searching the remaining rows and cells in the K-NN tables $\boldsymbol{D}$ and $\boldsymbol{J}$ (analogous

to those shown in Tables 1.1 to 1.8), we use a balanced binary tree. This results in, in the worst case, an $O(\log n)$ search at stage n. Considering that the K-NN algorithm only needs to be called a few times during the FSSF-b algorithm, the entire backward elimination portion of the FSSF-b algorithm is $O(qN \log N)$, the same as the K-NN portion of the algorithm. For a fixed q, since we use $N = 1000q + 2n_{\max}$ (see Section 1.3), i.e., $N = O(n_{\max})$, the theoretical time complexity becomes $O(n_{\max} \log n_{\max})$. When q is fixed, the memory complexity of the FSSF algorithms are all linear with respect to N (and thus to $n_{\max}$).

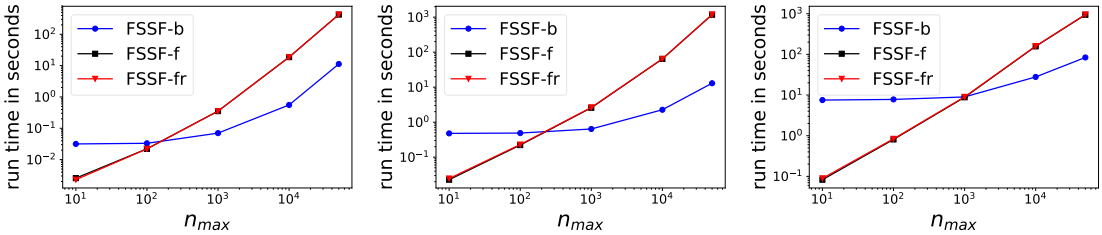
We should point out that the major contributor to the computational expense of the FSSF-b algorithm is the K-NN portion, even though it has the same theoretical $O(N \log N)$ complexity as the backward removal portion. The additional expense of the Sobol sequence candidate set generation and the backward removal portion was less than 10% of the overall expense of the FSSF-b algorithm in all experiments shown in this chapter. In the provided FSSF R package, we used an approximate nearest neighbor (ANN) searching algorithm (the ANN library that we used is Mount and Arya 2010), which allows some small potential inaccuracies in finding the exact nearest neighbors in exchange for faster speed and less memory cost.

Figure 1.9 shows the actual computation time of the FSSF-b/f/fr algorithms versus $n_{\max}$ for different dimensions $q = 2, 8, 16$. The FSSF designs in each plot are for $n_{\max} = 10, 100, 1000, 10,000, 50,000$, with the candidate set being a Sobol sequence of size $N = 1000q + 2n_{\max}$. The statistics in Figure 1.9 are based on a single replicate, because we have observed in separate experiments (which we omit, for brevity) that the runtime variation between different replicates of the FSSF algorithms is negligible. Notice that the

runtimes of the FSSF-f and FSSF-fr algorithms are almost identical, and hence the red and black curves overlap in Figure 1.9. As before, the candidate set was a Sobol sequence of size $N = 1000q + 2n_{\max}$. We note that the computation times shown in Figure 1.9 are for the entire algorithm. For instance, the computation time reported in Figure 1.9 of the FSSF-b algorithm includes generating the candidate set, conducting K-NN, and conducting the backward elimination sequence. Based on the theoretical computational expense results, and since the axes in Figure 1.9 are in log scale, one might expect the runtime curves in Figure 1.9 to be nearly straight lines, with the lines for the FSSF-f/fr algorithms having larger slopes than for the FSSF-b algorithm. This was not the case for the FSSF-b algorithm, and this may be due to the fact that the actual run time of the FSSF-b algorithm is largely a function of N, and when $n_{\max} <= 10^3$, our choice of $N = 1000q + 2n_{\max}$ is dominated by q and is not proportional to $n_{\max}$.

Figure 1.9 shows that the FSSF algorithms, and especially the FSSF-b algorithm, are computationally efficient. For instance, even when $n_{\max} = 50,000$ and $q = 16$, the FSSF-b algorithm only took around one minute. From Figure 1.9, we observe that in any tested dimension ($q = 2, 8, 16$), when $n_{\max}$ is large, the FSSF-b algorithm can be much faster than the FSSF-f/fr algorithms, which is consistent with the above theoretical time complexity analysis. The value of $n_{\max}$ beyond which FSSF-b is faster than FSSF-f/fr increases with q. For instance, when $q = 2$, FSSF-b is faster than FSSF-f/fr when $n_{\max}$ is larger than around 10^2, while when $q = 16$, FSSF-b is faster than FSSF-f/fr when $n_{\max}$ is larger than around 10^3. One also observes from Figure 1.9 that the efficiency advantage of the FSSF-b algorithm, relative to the FSSF-f/fr algorithms, is more significant as $n_{\max}$

increases. For example, for $q = 8$, FSSF-b is around 5 times faster than FSSF-f/fr when $n_{\max} = 10^3$, versus around 100 times faster when $n_{\max} = 50,000$.



(a) $q = 2$ (b) $q = 8$ (c) $q = 16$

Figure 1.9. Runtime comparisons of the FSSF algorithms. The runtimes of the FSSF-f and FSSF-fr algorithms are virtually identical, so their curves overlap.

1.5. FSSF Design Extensions

This section briefly discusses potential extensions of the FSSF algorithms to a weighted Euclidean distance (Section 1.5.1), which may be desirable if the response smoothness is different in different input coordinates, and to non-distance based design criteria (Section 1.5.2).

1.5.1. FSSF Designs for Inputs with Different Lengthscale Parameters

The FSSF designs discussed so far are generated without taking into account any response observations or any underlying statistical model for the response, as they are based entirely on space-filling criteria. In this context, response observations and modeling would only be used to decide when to stop experimentation. However, even for the purpose in this chapter of obtaining a globally accurate surrogate model (as opposed to response optimization), one might consider incorporating response observations at each stage when deciding the remaining sequence of design points. When using GP surrogate models, a

potential advantage of this is to detect if the lengthscale parameters for each input differ and to incorporate this into the design accordingly. In this section, we extend the FSSF designs to unequal scaling in the inputs (via changing the standard distance to a weighted distance), investigate its effect on the integrated mean square error (IMSE) performance for GP response surfaces, and compare this to the GP-based maxMSE sequential design strategy (defined shortly) that automatically incorporates lengthscale information via the GP covariance model.

1.5.1.1. Scaled FSSF Designs. Suppose the response surface $Y(\boldsymbol{x})$ is modeled as a GP with constant prior mean and squared-exponential covariance function $Cov(Y(\boldsymbol{x}), Y(\boldsymbol{x}')) = \sigma^2 R(\boldsymbol{x}, \boldsymbol{x}')$, where σ^2 is the prior variance,

$$R(\boldsymbol{x}, \boldsymbol{x}') = \prod_{j=1}^{q} \exp\left(-\left(\frac{x_j - x_j'}{\theta_j}\right)^2\right), \tag{1.6}$$

$\boldsymbol{x} = (x_1, x_2, \cdots, x_q)$, $\boldsymbol{x}' = (x_1', x_2', \cdots, x_q')$, and θ_j represents the lengthscale parameter for the jth input coordinate. Intuitively, a larger θ_j means the response depends on x_j in a smoother manner, and the best design may have points that are less densely spaced in the x_j direction than in other input directions.

We accomplish this within the FSSF algorithms by replacing the regular Euclidean distance by the weighted distance (Johnson, L. M. Moore, and Ylvisaker (1990))

$$d_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{\sum_{j=1}^{q} \left(\frac{x_j - x_j'}{\theta_j}\right)^2}. \tag{1.7}$$

We refer to designs generated using the weighted and unweighted distance as scaled and non-scaled designs, respectively. The reflected distance in line 4 of the FSSF-fr algorithm (Algorithm 2) should be modified when using the weighted distance. Otherwise, if (1.7) is

used in line 4 without modification, we have found that design points tend to be clustered too much towards the center of the design space in x_j coordinate directions having large θ_j. We have found that replacing line 4 of the FSSF-fr algorithm (Algorithm 2) by the following avoids this: Set $D_j = 2\sqrt{2q}\Big(\prod_{i=1}^{q} \frac{\theta_{i^*(j)}}{\theta_i}\Big)^{\frac{1}{q}} \min_{\boldsymbol{y}\in\partial\Omega} d_\theta(\boldsymbol{x}_j, \boldsymbol{y})$, where $i^*(j) = \arg\min_{i\in\{1,\cdots,q\}} \min\{\frac{x_{ji}}{\theta_i}, \frac{1-x_{ji}}{\theta_i}\}$.

For $n_{\max} = 100$, $q = 2$ and $\boldsymbol{\theta} = (1, 20) \times 0.5$, Figure 1.10 shows the designs generated by the FSSF-f algorithm (Algorithm 1), the FSSF-fr algorithm (Algorithm 2) and the maxMSE algorithm. The latter is a heuristic that attempts to reduce the MSE in general by selecting the design point at stage $n + 1$ as the candidate point having the largest predicted MSE after stage n, i.e.

$$\boldsymbol{x}_{n+1} \stackrel{\text{def}}{=} \arg\max_{\boldsymbol{x}\in\mathcal{C}\setminus S_n} \text{MSE}(\boldsymbol{x}, S_n), \tag{1.8}$$

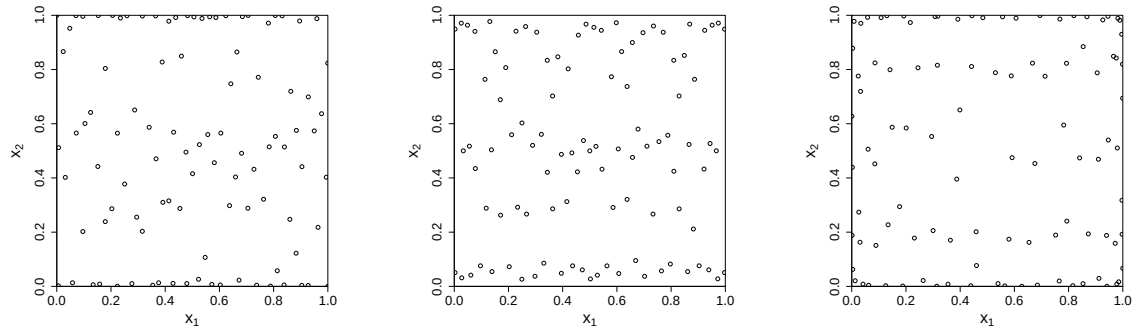
where

$$\text{MSE}(\boldsymbol{x}, S_n) = \sigma^2\Big[1 - \begin{pmatrix} 1 & \boldsymbol{r}^T(\boldsymbol{x}) \end{pmatrix} \begin{pmatrix} 0 & \mathbf{1}_{1\times n} \\ \mathbf{1}_{n\times 1} & \boldsymbol{R} \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ \boldsymbol{r}(\boldsymbol{x}) \end{pmatrix}\Big], \tag{1.9}$$

is the predicted MSE of $Y(\boldsymbol{x})$ given the hypothetical response observations at design S_n (Sacks et al. (1989)). In the preceding, $\boldsymbol{r}^T(\boldsymbol{x}) = \Big(R(\boldsymbol{x}_1, \boldsymbol{x}) \quad R(\boldsymbol{x}_2, \boldsymbol{x}) \quad \cdots \quad R(\boldsymbol{x}_n, \boldsymbol{x})\Big)$, and $\boldsymbol{R}$ is the $n \times n$ correlation matrix whose (i, j)-th element is $R(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

We see from Figure 1.10 that because the response dependence in the x_2 coordinate is much smoother than in the x_1 coordinate (i.e., $\frac{\theta_2}{\theta_1} = 20$), the maxMSE design (Figure 1.10c) tends to form horizontal bands of points and essentially place points more densely in the rougher x_1 direction than in the smoother x_2 direction. The scaled FSSF-f

and FSSF-fr designs in Figures 1.10a and 1.10b show similar patterns as the maxMSE design. As the numerical IMSE results in Section 1.5.1.2 will demonstrate, this does indeed tend to give a smaller IMSE than the non-scaled designs. Figure 1.10 also indicates that the scaled FSSF-fr design does not put as many design points on the boundary as the scaled FSSf-f and maxMSE designs do, and we will further demonstrate in Section 1.5.1.2 that this also helps to reduce the IMSE.



(a) Scaled FSSF-f design (b) Scaled FSSF-fr design (c) Scaled maxMSE design

Figure 1.10. Designs produced by FSSF-f, FSSF-fr and maxMSE algorithms for $\boldsymbol{\theta} = (1, 20) \times 0.5$ and $q = 2$ with design size $n = 100$.

1.5.1.2. IMSE Performances of Scaled and Non-scaled FSSF Designs. To compare the performances of the scaled and non-scaled FSSF designs and the maxMSE design, we use the IMSE (see Appendix A.2 for discussions about the link between IMSE and the space-filling property). For a design S_n of size n, the IMSE is defined as

$$\text{IMSE}(S_n) = \int_{\Omega} \text{MSE}(\boldsymbol{x}, S_n)\, \mathrm{d}\boldsymbol{x}. \tag{1.10}$$

Tables 1.9 and 1.10 compare the IMSE results for various n and $\boldsymbol{\theta}$ for $q = 2$ and $q = 8$, respectively. The IMSE in Equation (1.10) is computed via Monte Carlo integration with

10^6 Sobol sequence points inside $[0,1]^q$. In all cases we used $\sigma^2 = 1$, and all designs began with the same point $\boldsymbol{x}_{i_1} = 0.5 \times \mathbf{1}_q$, where $\mathbf{1}_q$ denotes a q-length vector of ones. We averaged the IMSE values across ten Monte Carlo replicates, each using a different Sobol sequence as the candidate set, and the square root of the averaged IMSE values are reported in the tables. The sample standard deviations of the IMSE values across the ten replicates were negligible and are omitted for brevity.

From Tables 1.9 and 1.10 we see that FSSF-fr algorithm usually achieves better IMSE than the maxMSE and FSSF-f algorithms for either $q = 2$ or $q = 8$. The highlighted cells in both Tables 1.9 and 1.10 are the IMSE results when one uses the correct lengthscale parameter (i.e., $\boldsymbol{\theta}_{\text{true}} = \boldsymbol{\theta}_{\text{assumed}}$). The FSSF-fr algorithm generally achieves better accuracy not only when $\boldsymbol{\theta}_{\text{true}} = \boldsymbol{\theta}_{\text{assumed}}$, but also when $\boldsymbol{\theta}_{\text{true}} \neq \boldsymbol{\theta}_{\text{assumed}}$. We also observe that when the scaling truly is different in different input coordinates (i.e., when $\boldsymbol{\theta}_{\text{true}} \neq c\mathbf{1}_q$ for some scalar c) the scaled FSSF-f and FSSF-fr algorithms do indeed performed better than the non-scaled versions when $q = 2$. But when $q = 8$, the relative performance of the scaled vs. non-scaled FSSF-fr algorithms are mixed. This may be because of our rule described above for handling the reflected distances with different scaling. A potentially more effective rule warrants further investigation.

n	$\boldsymbol{\theta}_{\text{true}}$	Method	$\boldsymbol{\theta}_{\text{assumed}}$			
			$(1,1)\times 1$	$(1,5)\times 1$	$(1,1)\times 0.5$	$(1,5)\times 0.5$
10	$(1,1)\times 1$	FSSF-f	0.0416	0.0651	0.0416	0.0651
		FSSF-fr	0.0225	0.0301	0.0225	0.0301
		minMMSE	0.0395	0.0514	0.0385	0.0758
10	$(1,1)\times 0.5$	FSSF-f	0.2704	0.3591	0.2704	0.3591
		FSSF-fr	0.1538	0.2129	0.1538	0.2129
		minMMSE	0.2549	0.3042	0.2533	0.3797
10	$(1,5)\times 0.5$	FSSF-f	0.1182	0.0380	0.1182	0.0380
		FSSF-fr	0.0414	0.0344	0.0414	0.0344
		minMMSE	0.0979	0.0324	0.1066	0.0402
20	$(1,1)\times 1$	FSSF-f	0.0023	0.0052	0.0023	0.0052
		FSSF-fr	0.0024	0.0028	0.0024	0.0028
		minMMSE	0.0022	0.0043	0.0021	0.0044
20	$(1,1)\times 0.5$	FSSF-f	0.0558	0.0982	0.0558	0.0982
		FSSF-fr	0.0430	0.0575	0.0430	0.0575
		minMMSE	0.0653	0.0886	0.0582	0.0977
20	$(1,5)\times 0.5$	FSSF-f	0.0101	0.0028	0.0101	0.0028
		FSSF-fr	0.0033	0.0023	0.0034	0.0023
		minMMSE	0.0049	0.0029	0.0052	0.0024

Table 1.9. For $q = 2$, IMSE performance and robustness comparison of FSSF-f, FSSF-fr, and maxMSE designs. The reported values are the square root of the IMSE for various combinations of true and assumed $\boldsymbol{\theta}$. The shaded cells indicate when the assumed and true $\boldsymbol{\theta}$ coincide.

n	$\boldsymbol{\theta}_{\text{true}}$	Method	$\boldsymbol{\theta}_{\text{assumed}}$			
			$(\mathbf{1}_6, \mathbf{1}_2) \times 4$	$(\mathbf{1}_6, \mathbf{5}_2) \times 4$	$(\mathbf{1}_6, \mathbf{1}_2) \times 2$	$(\mathbf{1}_6, \mathbf{5}_2) \times 2$
40	$(\mathbf{1}_6, \mathbf{1}_2) \times 4$	FSSF-f	0.0301	0.0345	0.0301	0.0345
		FSSF-fr	0.0259	0.0319	0.0259	0.0319
		minMMSE	0.0273	0.0302	0.0280	0.0318
40	$(\mathbf{1}_6, \mathbf{1}_2) \times 2$	FSSF-f	0.1502	0.1625	0.1502	0.1625
		FSSF-fr	0.1251	0.1492	0.1251	0.1492
		minMMSE	0.1394	0.1452	0.1404	0.1502
40	$(\mathbf{1}_6, \mathbf{5}_2) \times 2$	FSSF-f	0.0888	0.0868	0.0888	0.0868
		FSSF-fr	0.0720	0.0767	0.0720	0.0767
		minMMSE	0.0817	0.0782	0.0819	0.0802
80	$(\mathbf{1}_6, \mathbf{1}_2) \times 4$	FSSF-f	0.0130	0.0128	0.0130	0.0128
		FSSF-fr	0.0102	0.0115	0.0102	0.0115
		minMMSE	0.0114	0.0116	0.0116	0.0118
80	$(\mathbf{1}_6, \mathbf{1}_2) \times 2$	FSSF-f	0.0862	0.0895	0.0862	0.0895
		FSSF-fr	0.0691	0.0795	0.0691	0.0795
		minMMSE	0.0806	0.0824	0.0818	0.0831
80	$(\mathbf{1}_6, \mathbf{5}_2) \times 2$	FSSF-f	0.0469	0.0428	0.0469	0.0428
		FSSF-fr	0.0342	0.0352	0.0342	0.0352
		minMMSE	0.0431	0.0421	0.0435	0.0423

Table 1.10. For $q = 8$, IMSE performance and robustness comparison of FSSF-f, FSSF-fr, and maxMSE designs. The reported values are the square root of the IMSE for various combinations of true and assumed $\boldsymbol{\theta}$. The shaded cells indicate when the assumed and true $\boldsymbol{\theta}$ coincide.

1.5.1.3. FSSF Designs with a Warm Start. As mentioned earlier, $\boldsymbol{\theta}$ is usually unknown, in which case it must be estimated via some initial design and corresponding response observations. Our FSSF R package includes a "warm-start" feature that allows an initial design to be specified and then generates the additional FSSF design points sequentially to add to the initial design. If $\boldsymbol{\theta}$ is estimated based on the initial design, it can also be included as an argument to the warm-start feature to generate the remainder of the design using a scaled-distance FSSF criterion. As an example, Figure 1.11 shows a design of size $n = 120$ for $q = 2$ using an initial non-scaled FSSF-fr design of size 20. The initial design, which is indicated by red squares, is evenly spaced inside $[0, 1]^2$. Suppose the estimate of $\boldsymbol{\theta}$ based on this initial design is $(1, 20) \times 0.5$. The round open circles in Figure 1.11 show the additional 100 design points selected via the scaled FSSF-fr algorithm using this estimated $\boldsymbol{\theta}$. These additional design points form horizontal bands in the rougher x_1 coordinate direction, similar to those in Figure 1.10.

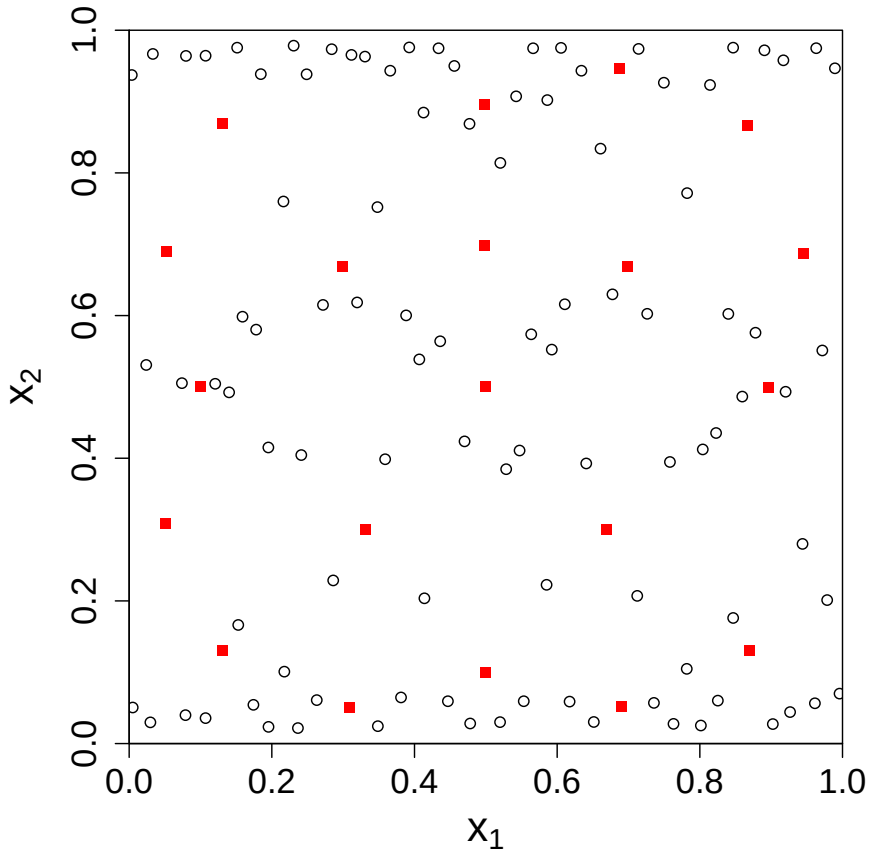


Figure 1.11. Scaled warm-start FSSF-fr design of total size $n_{\max} = 120$ using an initial non-scaled FSSF-fr design of initial size 20 (red squares) for $q = 2$. The additional 100 points from the warm-start scaled FSSF-fr algorithm (open black circles) are for $\boldsymbol{\theta}_{\text{assumed}} = (1, 20) \times 0.5$.

1.5.2. Extension to Non-distance-based Design Criteria

In this chapter we have focused on computer experiments (the presumption being that a GP model will be fitted), for which distance-based space-filling criteria (Equations (1.1) and (1.2)) are the most relevant. If one were interested in fitting a polynomial regression

model instead of a GP model, then other criteria such as the polynomial canonical correlation criteria (Jobson (2012) and Tang (1998)) or any of the alphabetic optimality criteria (Montgomery (2017)) may be relevant. Here we briefly discuss how one could extend the forward FSSF-f approach to the D-optimality criterion (assuming a linear model) and to the first order polynomial canonical correlation criterion. It would be more difficult to extend the backward FSSF-b approach to these other criteria, for reasons that will be discussed shortly.

Suppose one has already obtained a design of size n, $S_n = \{\boldsymbol{x}_{i_1}, \cdots, \boldsymbol{x}_{i_n}\}$, and the immediate goal is to select another design point in the candidate set $\mathcal{C} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$ to add to the current design to form S_{n+1} according to the D-optimality criterion. Here, we assume a main-effects linear model with Gaussian errors, so that the Fisher information matrix for S_n is (a scalar constant times) $\boldsymbol{I}_n = \sum_{k=1}^{n} \boldsymbol{x}_{i_k} \boldsymbol{x}_{i_k}^T$. The optimal candidate point to be added to S_n is the one that maximizes the determinant of $\boldsymbol{I}_{n+1}$, i.e.,

$$\boldsymbol{x_{n+1}} = \underset{\boldsymbol{z} \in \mathcal{C} \setminus S_n}{\arg\max} |\boldsymbol{I}_n + \boldsymbol{z}\boldsymbol{z}^T| = \underset{\boldsymbol{z} \in \mathcal{C} \setminus S_n}{\arg\max} |\boldsymbol{I}_n|(1 + \boldsymbol{z}^T \boldsymbol{I}_n^{-1} \boldsymbol{z}) = \underset{\boldsymbol{z} \in \mathcal{C} \setminus S_n}{\arg\max}\, \boldsymbol{z}^T \boldsymbol{I}_n^{-1} \boldsymbol{z}, \tag{1.11}$$

where $|\bullet|$ denotes the determinant of a matrix. The second equality in Equation (1.11) follows from the matrix determinant lemma for rank-one modifications (Harville (1997)). Note that $\boldsymbol{I}_n^{-1}$ can also be conveniently updated using the Sherman-Morrison formula (Sherman and Morrison (1950)) for the inverse of a rank-one modification of a matrix.

Now consider the total first order polynomial canonical correlation criterion, which is equivalent to the sum of absolute values of the sample correlations for all pairs of input variables (see Jobson (2012) and Tang (1998) for detailed definitions). If the next design

point is chosen to minimize this criterion, we have

$$
\begin{aligned}
&\boldsymbol{x}_{n+1} \\
&= \arg\min_{\boldsymbol{z}\in\mathcal{C}\setminus S_n} \sum_{1\le l<j\le q} \left| \frac{1}{n}\Big(\sum_{k=1}^{n} \Big(x_{i_k l} - \bar{\boldsymbol{x}}^{l}_{n+1}\Big)\Big(x_{i_k j} - \bar{\boldsymbol{x}}^{j}_{n+1}\Big) + (z_l - \bar{\boldsymbol{x}}^{l}_{n+1})(z_j - \bar{\boldsymbol{x}}^{j}_{n+1}\Big) \right| \\
&= \arg\min_{\boldsymbol{z}\in\mathcal{C}\setminus S_n} \sum_{1\le l<j\le q} \left| z_l \sum_{k=1}^{n} x_{i_k j} + z_j \sum_{k=1}^{n} x_{i_k l} - n z_l z_j \right| \\
&= \arg\min_{\boldsymbol{z}\in\mathcal{C}\setminus S_n} \sum_{1\le l<j\le q} \left| z_l \bar{\boldsymbol{x}}^{j}_{n} + z_j \bar{\boldsymbol{x}}^{l}_{n} - z_l z_j \right|
\end{aligned}
\tag{1.12}
$$

where $\bar{\boldsymbol{x}}^{j}_{n+1} = \bar{\boldsymbol{x}}^{j}_{n+1}(\boldsymbol{z}) = \frac{1}{n+1}(\sum_{t=1}^{n} x_{i_t j} + z_j)$, $\bar{\boldsymbol{x}}^{j}_{n} = \frac{1}{n}\sum_{t=1}^{n} x_{i_t j}$ and z_j is the jth coordinate of $\boldsymbol{z}$. To efficiently compute $\boldsymbol{x}_{n+1}$ in Equation (1.12), quantities like $\bar{\boldsymbol{x}}^{j}_{n}$ for each $j \in \{1, \cdots, q\}$ can be updated and stored at each iteration.

Equations (1.11) and (1.12) provide convenient formula to be used in a forward algorithm with D-optimality or polynomial canonical correlation as the criteria for selecting the next design point. Although the same formula could be used in a backward-elimination algorithm (analogous to Algorithm 3), we do not recommend extending the FSSF-b algorithm to these criteria. This is because the FSSF-b algorithm (Algorithm 3) for our distance-based criteria requires use of KNN for computational efficiency, which allows each $\boldsymbol{x}_{i_k}$ to be found in $O(\log N)$ time after eliminating $\boldsymbol{x}_{i_{k+1}}$. However, it is not clear how to use KNN to achieve the same computational efficiency for the D-optimality or first-order polynomial canonical correlation criteria.

1.6. Conclusions

In this chapter, we developed and investigated various versions of fully-sequential (one-at-a-time) space-filling design algorithms. The FSSF-fr and FSSF-b algorithms are

modifications of the FSSF-f algorithm, which is similar to the CADEX algorithm proposed by Kennard and Stone 1969. The FSSF-fr algorithm improves the performance of the FSSF-f algorithm in terms of $h_{\max}$ criterion (Equation (1.2)) by making use of the reflected design points. The FSSF-b algorithm improves the computational expense of the FSSF-f algorithm for large $n_{\max}$ using backward elimination and a fast K-NN algorithm. For fixed dimension q, the theoretical time complexity of FSSF-b is of $O(n_{\max} \log n_{\max})$ versus $O(n_{\max}^2)$ for FSSF-f/fr. In terms of actual runtime for the examples we considered, the FSSF-b algorithm was as much as 100 times faster than the FSSF-f algorithm (see Figure 1.9b).

We also proposed extensions of the FSSF algorithms to other distance-based and non-distance-based design criteria. As discussed in Section 1.5, the FSSF methods can be readily extended to other user-defined distances and can be modified to select fully-sequential design points according to certain non-distance-based design criteria.

For practitioners using the FSSF designs for computer experiments, we recommend the following guidelines. If computational expense is not an issue (e.g., for small $n_{\max}$, or for large $n_{\max}$ but with unlimited run time resources), then we recommend using either (i) FSSF-f, if $d_{\min}$ is the primary performance measure of interest or (ii) FSSF-fr, if $h_{\max}$ is the primary performance measure of interest. If computational expense is a concern and $n_{\max}$ is large, we recommend using FSSF-b. The computational expense may be approximated from Figure 1.9.

We note that the runtime of the FSSF-b algorithm could be further improved by utilizing parallel computing. As we noted earlier, in all experiments shown in this chapter, more than 90% of computation time of the FSSF-b algorithm was for the K-NN portion

of the algorithm. Consequently, improving the computational expense of the K-NN algorithm can reduce the runtime of the FSSF-b algorithm. For example, instead of using the kd-tree structure (the performance of which degrades as q increases), one could implement the K-NN on a GPU in parallel (Garcia, Debreuve, and Barlaud 2008). In contrast, none of the other sequential methods would be able to take much advantage of parallel processing, because most of their computational expense comes from the sequential part of the algorithm, and sequential algorithms by nature cannot be parallelized.

One potential disadvantage of the proposed FSSF algorithms is that they do not consider the space-filling properties of designs when projected into lower-dimensinal subspaces of the design space. As future work, we are investigating incorporating concepts in V Roshan Joseph, Gul, and Shan Ba 2015 into the FSSF algorithms to provide better space-filling properties in projected subspaces.

1.7. Supplementary Material

The code implementing all proposed algorithms in the chapter can be found in the R package FSSF (Shang and Apley 2020), which is available on the online Supplementary Materials section for Shang and Apley 2021.

CHAPTER 2

Diversity Subsampling: Custom Subsamples from Large Data Sets

2.1. Introduction

Diversity subsampling selects a subset of points from a data set with the goal of having the selected points spread out evenly over the data space. It has been found useful in various fields. For instance, diversity subsampling from massive scRNA-seq data sets helps preserve rare cell types (Song et al. 2022b); it serves as a data splitting tool in building predictive models (Puzyn et al. 2011); it is frequently used as a pre-processing step to select representative subsample points (Silveira and Barbeira 2022); and it has been applied to active learning algorithms (Yu and Kim 2010; Haussmann et al. 2020).

Suppose we have data set D consisting of an identically and independently distributed (i.i.d.) sample of size N of some random vector $\boldsymbol{X} \in \mathbb{R}^q$; i.e., $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\}$, where $\boldsymbol{x}_i \in \mathbb{R}^q$. Let $\mathcal{S}_n = \{\boldsymbol{x}_{j_1}, \cdots, \boldsymbol{x}_{j_n}\} \subset D$, for $\{j_1, \cdots, j_n\} \subset \{1, 2, \cdots, N\}$ denote a subsample of size n selected from D. In the literature, loosely speaking, $\mathcal{S}_n$ is called a diverse subsample from D if the points it constitutes are as different from each other as possible (sometimes called the repulsiveness property, such as in Wang et al. 2018; Bıyık et al. 2019) and cover as much of the effective support of the data as possible. The typical setting is that one will observe some response variable for the n cases in $\mathcal{S}_n$ (e.g., by conducting some follow-up experimental or observational study, surveying the cases, etc.) which will then serve as the training data for fitting some supervised learning model.

Existing popular methods for diversity subsampling from given data include Determinantal Point Processes (DPP, Bıyık et al. 2019), Computer Aided Design of EXperiments (CADEX, Kennard and Stone 1969), Poisson Disk Sampling (PDS, Cook 1986; Yuksel 2015) and k-means clustering (MacQueen et al. 1967; D. Wu 2018). DPP can be used for diversity subsampling by first specifying an appropriately constructed kernel matrix

so that the most diverse set of points will have the largest probability of being selected under this DPP; this set is thus the mode of the DPP. It has been shown that finding this mode is an NP-hard problem (Ko, Lee, and Queyranne 1995); various algorithms have been proposed to approximate the mode of a DPP, such as (Bıyık et al. 2019; Han and Gillenwater 2020). CADEX selects points sequentially from data such that the distance between each newly selected point and its closest point in the previously selected point set is as large as possible. Song et al. 2022b recently proposed an efficient heuristic for the CADEX algorithm, which is called the scSampler algorithm. PDS is a conceptually simple algorithm that selects subsequent points outside of the union of hyper-spherical neighborhoods of all selected points. The radius of these hyper-spherical neighborhoods is usually user-specified and taken as an input of the algorithm, which ensures that the pairwise distances between all selected points are no smaller than twice the specified radius (Cook 1986). Existing PDS algorithms that do not require the users to pre-specify a fixed radius include McCool and Fiume 1992 and Yuksel 2015. D. Wu 2018 proposed the RD ALR method (Representativeness and Diversity, Active Learning for Regression) that uses k-means clustering (MacQueen et al. 1967) to select a diverse subsample from D; it selects each point in $\mathcal{S}_n$ sequentially. At iteration j, RD ALR clusters the entire data into $j+1$ clusters using k-means, and the newly selected point are chosen from clusters that do not contain any previously selected points.

One common property of the above-mentioned methods is that the subsamples they select are all repulsive, in the sense that they attempt to ensure that the selected points are as far away from each other as possible. Although repulsiveness is an appealing property of a uniform or space-filling subsample, it might not be achievable when there are many

replicated observations in D. In contrast to the existing subsampling algorithms, we define our diversity subsampling goal as to select a subsample that follows, as closely as possible, a mutually independent uniform distribution over the effective support of D, and we refer to our proposed algorithm as the diversity subsampling (DS) algorithm. Our DS algorithm has substantial benefits compared with existing ones. First, aiming to select an i.i.d uniform subsample over the effective support of D allows replications to occur in the subsample. This has advantages in machine learning applications in which the response variable Y is observed with error and/or when there are additional latent variables, since in these situations it may be desirable to observe Y at multiple very similar X values. Second, our DS algorithm handles data with large numbers of replicated values along all or certain dimensions much better than existing algorithms: In this challenging situation, the subsample selected by the DS algorithm is still as evenly spread out over the data space as possible, but the subsamples selected by existing algorithms degraded to random sampling (we provide an example to illustrate this shortly). Third, our DS algorithm turns out to be much more computationally efficient, especially with larger N and n (which is increasingly common in the big data era) than existing methods (see Section 2.5.3 for details). Lastly, it is easy to generalize our DS algorithm to select subsamples following any desired distribution (not just uniform).

To illustrate the better uniformity properties of the DS subsample than existing methods when there are many replicated observations in D, consider the following simple example in which $\boldsymbol{X}$ follows a bivariate standard normal distribution with independent components. We set $q = 2$, $N = 10,000$ and $n = 2,000$ in this example. The $10,000$ data points were generated using $2,000$ independently drawn points from the bivariate

standard normal distribution, each of which was replicated five times. Figure 2.1 compares heat maps of the estimated density of typical subsamples selected by three different methods: random sampling from a data set, scSampler-sp1 (Song et al. 2022b), and our DS algorithm (see Section 2.2 for details of the DS algorithm). The densities were estimated using a Gaussian kernel with a bandwidth of 0.1. As Figure 2.1 shows, the points selected by random sampling (the left plot of Figure 2.1) follow the original bivariate standard normal distribution, i.e., they concentrate in the middle region of the plotted area, where the data points (not shown in Figure 2.1) are denser. The subsample selected by the scSampler-sp1 algorithm (the middle plot of Figure 2.1) also exhibits the same phenomenon. In contrast, the subsample selected by DS (the right plot of Figure 2.1) is much closer to being uniformly distributed over the effective support of the data.

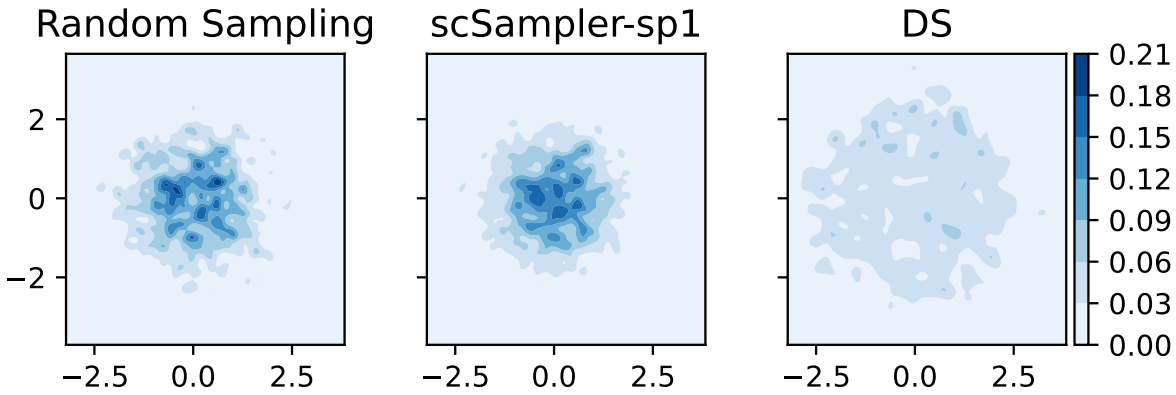


Figure 2.1. Heat maps comparing the estimated densities of the selected subsamples using three different methods for $q = 2$. The left, middle, and right plots are for the subsamples selected by random sampling, scSampler-sp1, and DS, respectively. The DS subsample is far more uniformly distributed than the other two.

The structure of this chapter is as follows. Section 2.2 presents our DS algorithm. Section 2.3 discusses generalizations of the proposed DS algorithm to select subsamples

following any desired distribution and proves that under mild conditions, the subsample (of fixed size n) selected by the generalized DS algorithm converges in distribution to an i.i.d sample following the desired distribution, as N approaches infinity. Section 2.5 numerically demonstrates the effectiveness of the DS algorithm and its advantages compared to known methods using data exhibiting a variety of distributions. Section 2.6 concludes the chapter.

2.2. The DS Algorithm for Diversity Subsampling

The goal of our DS algorithm is to select a subsample $\mathcal{S}_n$ of size n (typically, n is much less than the data size N) points in $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\}$ that is distributed as closely as possible to an i.i.d. sample from the uniform distribution over S, the effective support of the data distribution. First, we discuss the case when the distribution of $\boldsymbol{X}$ is continuous, then we extend our DS method for general data sets with discrete or mixed continuous and discrete data distributions. Let $f(\boldsymbol{x})$ and S denote the probability density function (p.d.f) of $\boldsymbol{X}$ and its support, both of which are assumed unknown. We develop two versions of the DS algorithm, for sampling from D with or without replacement, the former being relevant in a smaller class of applications in which it is desirable to observe the response Y multiple times for the same observation $\boldsymbol{X}$. For brevity, we focus on the sampling-without-replacement version of the DS algorithm (denoted as the DS algorithm) in this section and discuss the sampling-with-replacement version (denoted as the DS-WR algorithm) briefly. In this section, we also discuss the relation of the proposed DS algorithm to the well-known acceptance-rejection sampling approach (Casella, Robert, and Wells 2004), as the DS algorithm has certain conceptual similarities but is developed

for the more challenging situation of sampling from a given finite data set D, as compared to generating random samples from some specified proposal distribution. We prove certain asymptotic performance results for the DS algorithm in a more general way in Section 2.4, where we demonstrate that under certain assumptions the uniformity and independence of points in $\mathcal{S}_n$ can be guaranteed asymptotically.

Procedure 1 summarizes the DS algorithm for sampling without replacement, which we describe as follows. We first estimate the density $f(\boldsymbol{x})$ at each data point in D using some suitable density estimator [1] (see Appendix B.1 for details). Denote the estimated density at $\boldsymbol{x}_i$ as $\hat{f}(\boldsymbol{x}_i)$, for $i = 1, \cdots, N$. Then the DS algorithm selects the n points in $\mathcal{S}_n$ from D sequentially at each iteration k as follows (for $k = 1, 2, ..., n$).

At iteration $k = 1$, the DS algorithm selects the index $j_1 \in \{1, \cdots, N\}$ of the first point in $\mathcal{S}_n$, denoted by $\boldsymbol{x}_{j_1}$, with probability

$$P(j_1) = \frac{\frac{1}{\hat{f}(\boldsymbol{x}_{j_1})}}{\sum_{i=1}^{N} \frac{1}{\hat{f}(\boldsymbol{x}_i)}}. \tag{2.1}$$

At iteration $k \geq 2$, the DS algorithm selects the index $j_k \in \{1, \cdots, N\} \setminus \{j_1, \cdots, j_{k-1}\}$ of the next sampled point $\boldsymbol{x}_{j_k}$ with probability

$$P(j_k | j_1, \cdots, j_{k-1}) = \frac{\frac{1}{\hat{f}(\boldsymbol{x}_{j_k})}}{\sum_{i=1}^{N} \frac{1}{\hat{f}(\boldsymbol{x}_i)} - \sum_{i=1}^{k-1} \frac{1}{\hat{f}(\boldsymbol{x}_{j_i})}}. \tag{2.2}$$

Two practical issues must be addressed when applying the above idea to real data sets. One issue is that n might not be negligibly small compared to N. Since we are sampling without replacement, the size of the remaining data set $D \setminus \mathcal{S}_{k-1}$ at iteration

[1] One wants to make sure that there are no zero or negative values in the estimates e.g., due to numerical errors.

k decreases with k; thus the distribution of the remaining data can change dramatically from the distribution of the original data D. We remedy this by updating the estimated density regularly so that it reflects the distribution of the remaining data and not the distribution of D. In particular, we update the density of remaining points in D after every $M = \max\{100, \lfloor \frac{n}{U} \rfloor\}$ points have been selected, where U is a user-chosen integer (we use $U = 10$ in all examples) . Here the symbol $\lfloor \frac{n}{U} \rfloor$ denotes the largest integer not larger than $\frac{n}{U}$. We provide a method to update density efficiently in Appendix B.1.

The other issue is that the density estimation method proposed in Appendix B.1 only applies when the data distribution is continuous. In the case when the data distribution is discrete or mixed continuous and discrete, we approximate the true distribution with a continuous one by adding a small Gaussian noise (lines 3–10 of Procedure 1). For simplicity and since data are often rounded, we apply this Gaussian perturbation for all data sets regardless of the true data distribution being discrete or not. To be specific, for the density estimation step, $\forall i \in \{1, \cdots, N\}$, we replace $\boldsymbol{x}_i$ with $\boldsymbol{x}_i + \sigma_p \boldsymbol{Z}_i$, where $\{\boldsymbol{Z}_i\}_{i=1}^N$ follow i.i.d multivariate normal distribution $\mathcal{N}(\mathbf{0}_q, \mathbf{1}_{q\times q})$, where $\mathbf{0}_q$ is the zero vector in $\mathcal{R}^q$ and $\mathbf{1}_{q\times q}$ is the identity matrix in $\mathcal{R}^{q\times q}$. We choose $\sigma_p > 0$ to be a small number compared to the pairwise distances of D. Note that adding Gaussian noise is actually consistent with the notion of Gaussian kernel density estimation (KDE), which can be viewed as convolving the empirical distribution of D with a Gaussian "noise" density with standard deviation related to the kernel bandwidth.

Procedure 1 summarizes the DS algorithm.

Procedure 1 DS Algorithm: Diversity Subsampling without Replacement

Input: n, $D = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\} \subset [0,1]^q$

1: $N \leftarrow |D|$

2: Standardize D into $[0,1]^q$, and store this new data set as $\tilde{D} = \{\tilde{\boldsymbol{x}}_i\}_{i=1}^N$

3: $n_s \leftarrow \min\{2000, \lfloor \frac{N}{4} \rfloor\}$

4: Randomly select $\{r_1, \cdots, r_{n_s}\} \subset \{1, \cdots, N\}$ with equal probabilities

5: $D_{\text{sub}} \leftarrow$ all unique points in $\{\tilde{\boldsymbol{x}}_{r_j}\}_{j=1}^{n_s}$

6: Repeat line 4 and 5 until $|D_{\text{sub}}| > 1$

7: $\sigma_p \leftarrow \frac{1}{8} \min_{\{\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}\} \subset D_{\text{sub}}} \|\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{y}}\|_2$

8: **for** $i = 1, \cdots, N$ **do**

9: Independently generate $\boldsymbol{Z}_i \sim \mathcal{N}(\boldsymbol{0}_q, \boldsymbol{1}_{q \times q})$

10: $\tilde{\boldsymbol{x}}_i \leftarrow \tilde{\boldsymbol{x}}_i + \sigma_p \boldsymbol{Z}_i$

11: **end for**

12: Estimate density $\hat{f}(\tilde{\boldsymbol{x}}_i)$, for each $i = 1, 2, \cdots, N$ (see Appendix B.1)

13: $\hat{f}(\boldsymbol{x}_i) \leftarrow \hat{f}(\tilde{\boldsymbol{x}}_i)$, for each $i = 1, 2, \cdots, N$

14: Set index array $I_1 \leftarrow \{1, \cdots, N\}$

15: Select j_1 randomly from I_1 with probabilities given by Equation (2.1)

16: $\mathcal{S}_1 \leftarrow \{\boldsymbol{x}_{j_1}\}$

17: Set an estimated number of density estimation updates U (e.g. $U \leftarrow 10$)

18: Set the number of points to select before conducting a density update. $M \leftarrow \max\{100, \lfloor \frac{n}{U} \rfloor\}$

19: Set $UpdateState \leftarrow \{M, 2M, \cdots, \lfloor \frac{n}{M} \rfloor M\}$

20: **for** $k = 2, \cdots, n$ **do**

21: $I_k \leftarrow I_{k-1} \setminus j_{k-1}$

22: **if** $k - 1 \in UpdateState$ **then**

23: Re-Estimate $\{\hat{f}(\tilde{\boldsymbol{x}}_i)\}_{i \in I_k}$ by efficient updating (See Appendix B.1 for details)

24: Reset $\hat{f}(\boldsymbol{x}_i) \leftarrow \hat{f}(\tilde{\boldsymbol{x}}_i)$, for each $i \in I_k$

Sampling-With-Replacement Version of the DS Algorithm In certain situations, subsampling from D with replacement may be relevant. In our primary setting where D is unlabeled data and the intent is to observe a response Y for each point in $\mathcal{S}_n$, sampling with replacement is only relevant if it is possible to experiment on (i.e., observe a response y_i for) the same subject $\boldsymbol{x}_i$ multiple times, each time observing a potentially different stochastic value for y_i. As an example, suppose $\boldsymbol{x}_i$ is a set of measured characteristics of sample i of a chemical compound from a large set D of samples over which $\boldsymbol{X}$ varies randomly, y_i is some output property of a subsequent chemical reaction with reactants taken from sample i, and y_i can vary randomly if the reaction is repeated using reactants from sample i again. In this case, one may want to use sampling from D with replacement, so the same sample can be experimented on multiple times if needed. On the other hand, suppose $\boldsymbol{x}_i$ is a set of clinical, phenotypical, and demographic characteristics of a patient i having a particular condition, and y_i is the efficacy of a treatment for that condition administered to patient i. In this case, it is not meaningful to apply an experimental treatment to the same patient twice, so sampling with replacement becomes irrelevant.

For the sampling-with-replacement version of the DS algorithm, denoted by DS-WR, at each iteration $k \in \{1, 2, \cdots, n\}$, we select the next point $\boldsymbol{x}_{j_k}$ from D as follows. The index $j_k \in \{1, 2, ..., N\}$ is selected with probability

$$P(j_k | j_1, \cdots, j_{k-1}) = P(j_k) = \frac{\frac{1}{\hat{f}(\boldsymbol{x}_{j_k})}}{\sum_{j=1}^{N} \frac{1}{\hat{f}(\boldsymbol{x}_j)}}, \tag{2.3}$$

independently of which points have been selected in earlier iterations. The DS-WR algorithm is identical to Procedure 1, except that for DS-WR, one repeats line 15 n times to

produce the indices $\{j_1, j_2, ..., j_n\}$ of the desired subsample of size n, and all lines after line 15 are omitted. The asymptotic performance of DS-WR can be proven similarly to Theorem 1, which we omit for brevity.

If the DS-WR version is relevant for a particular problem, there is a potential drawback that should be considered if there are severe outliers in the data. Most existing density estimation techniques tend to underestimate the density in extremely sparse tail regions of S, where there are only a few outlier points in D. This causes the estimated density at the outlier $\boldsymbol{x}$ points to have extremely low values and therefore very high probabilities of getting selected in an iteration of the DS algorithm. To provide the most diversity, it is generally desirable to select these points for the subsample, which the DS algorithm does. However, in the DS-WR algorithm, the same outlier points may get selected repeatedly for the subsample, which ends up being counterproductive for the diversity goal.

Figure 2.2 shows an example to illustrate this. The subsample of size $n = 100$ was selected by the DS-WR algorithm from a data set D with $N = 3000$ and $q = 2$. The open red circles are the selected points in $\mathcal{S}_n$, and the numbers beside them indicate how many times each point was selected. Notice that there are only 11 unique points in the subsample of size 100, and over half of the 100 points are repeated values of only three $\boldsymbol{x}_i$ points that were repeated 30, 15 and 14 times, respectively. The DS algorithm (without replacement, see Procedure 1) would likely have selected each of the outlier points, but only once each, which seems more desirable in this case.

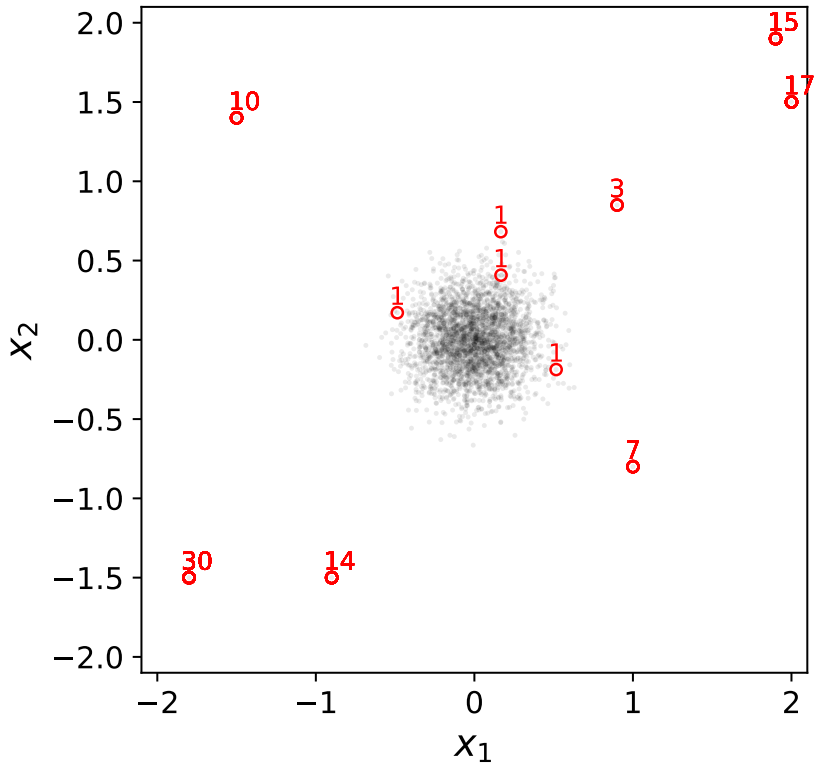


Figure 2.2. Illustration of the oversampling of outliers that can occur when sampling with replacement is used. The open red circles are the subsample $\mathcal{S}_n$ with $n = 100$, and the gray points are the set D of size $N = 3000$. The numbers next to the outlier points indicate how many times that point was selected in $\mathcal{S}_n$.

Connection to acceptance sampling. The fact that in the DS algorithm the probability a point $\boldsymbol{x}_i$ is selected is inversely proportional to $\hat{f}(\boldsymbol{x}_i)$ is reminiscent of the popular acceptance-rejection sampling approach (Casella, Robert, and Wells 2004). To relate our DS algorithm to acceptance-rejection sampling, suppose $f(\boldsymbol{x})$ denotes some specified proposal distribution in acceptance sampling, and the target distribution is uniform over S. Then acceptance-rejection sampling generates points $\{\boldsymbol{x}_i : i = 1, 2, \cdots\}$ randomly from $f(\boldsymbol{x})$, and each $\boldsymbol{x}_i$ is accepted with probability $\frac{\mathbf{1}_S(\boldsymbol{x}_i)}{c|S|f(\boldsymbol{x}_i)}$. Here $c \geq \sup_{\boldsymbol{x}\in S} \frac{1}{|S|f(\boldsymbol{x})} = \frac{1}{|S|\underline{f}}$,

where $\underline{\mathrm{f}} \stackrel{\mathsf{def}}{=} \inf_S f(\boldsymbol{x})$, $|S|$ denotes the volume of S and $\mathbf{1}_S(\boldsymbol{x})$ is an indicator function that equals 1 if $\boldsymbol{x} \in S$ and 0 otherwise. The optimal choice of c is $\frac{1}{|S|\underline{\mathrm{f}}}$ (Casella, Robert, and Wells 2004). Although acceptance-rejection sampling draws samples from a specified distribution $f(\boldsymbol{x})$ and not from D, one could hypothetically consider modifying it so that it does the latter. In particular, we could consider selecting points $\{\boldsymbol{x}_i : i = 1, 2, \cdots\}$ randomly from D with equal probability, and then accepting each $\boldsymbol{x}_i$ with probability $\frac{\mathbf{1}_S(\boldsymbol{x}_i)\underline{\mathrm{f}}}{f(\boldsymbol{x}_i)}$, where we take $c = \frac{1}{|S|\underline{\mathrm{f}}}$.

There are a number of reasons why this modified acceptance-rejection sampling is not well-suited to our objective of sampling from D. First, to produce the target uniform distribution, the acceptance probability $\frac{\mathbf{1}_S(\boldsymbol{x}_i)\underline{\mathrm{f}}}{f(\boldsymbol{x}_i)}$ requires that $\underline{\mathrm{f}}$ is known, which in turn requires a region S to be specified (our DS algorithms do not require that S is known). Second, even with optimal choice of c, the probability of accepting a randomly generated subsample from D is $\frac{1}{c} = |S|\underline{\mathrm{f}}$ (see Remark 2.5 of Asmussen and Glynn 2007), which will typically be small if we want to include regions in which $f(\boldsymbol{x})$ is very small. This problem is compounded by the fact that we must use a density estimator $\hat{f}(\boldsymbol{x})$ instead of $f(\boldsymbol{x})$, because density estimators often underestimate the tail densities (for example, KDE (Rosenblatt 1956 or K-nearest neighbor density estimation (Mack and Rosenblatt 1979)). With such a small average acceptance probability, there may not even be enough points in D to allow acceptance of n points for the subsample $\mathcal{S}_n$.

2.3. Generalization to Customized Non-Uniform Subsamples

In this section, we discuss how to generalize the DS method to select subsamples following desired distributions other than uniform and provide an example using the

generalized DS method to select a subsample with desired properties without using a well-defined target distribution. The theoretical performance of the generalized DS algorithm is shown in Section 2.4.

Analogous to acceptance-rejection sampling (Casella, Robert, and Wells 2004), the DS algorithm can also be generalized to select a subsample from D that follows a general, and not just uniform, target distribution. Consider the same setting with as in Section 2.2, and suppose we want to select an i.i.d. subsample from D following some desired distribution with specified density $g(\boldsymbol{x}) = c\tilde{g}(\boldsymbol{x})$ having support $S_g \subset S$. Here $\tilde{g}(\boldsymbol{x})$ is known but c (the constant necessary for $g(\boldsymbol{x})$ to be a proper density) can be unknown. The generalized DS algorithm differs from the DS algorithm only in that each $\boldsymbol{x}_i$ is selected with probability $\frac{\frac{\tilde{g}(\boldsymbol{x}_i)}{\hat{f}(\boldsymbol{x}_i)}}{\sum_{k=1}^{N}\frac{\tilde{g}(\boldsymbol{x}_k)}{\hat{f}(\boldsymbol{x}_k)}}$ for the generalized DS. Specifically, at iteration $k = 1$, the generalized DS algorithm selects the index $j_1 \in \{1, \cdots, N\}$ of the first select point $\boldsymbol{x}_{j_1}$, with probability

$$P(j_1) = \frac{\frac{\tilde{g}(\boldsymbol{x}_{j_1})}{\hat{f}(\boldsymbol{x}_{j_1})}}{\sum_{i=1}^{N}\frac{\tilde{g}(\boldsymbol{x}_i)}{\hat{f}(\boldsymbol{x}_i)}}. \tag{2.4}$$

At iteration $k \geq 2$, the generalized DS algorithm selects the index $j_k \in \{1, \cdots, N\} \setminus \{j_1, \cdots, j_{k-1}\}$ of the next sampled point $\boldsymbol{x}_{j_k}$ with probability

$$P(j_k|j_1, \cdots, j_{k-1}) = \frac{\frac{\tilde{g}(\boldsymbol{x}_{j_k})}{\hat{f}(\boldsymbol{x}_{j_k})}}{\sum_{i=1}^{N}\frac{\tilde{g}(\boldsymbol{x}_i)}{\hat{f}(\boldsymbol{x}_i)} - \sum_{i=1}^{k-1}\frac{\tilde{g}(\boldsymbol{x}_{j_i})}{\hat{f}(\boldsymbol{x}_{j_i})}}. \tag{2.5}$$

Consequently, Procedure 1 applies to the generalized DS algorithm with only line 15 and line 26 modified according to Equation (2.4) and Equation (2.5) respectively. We note that since N is finite in practice, the performance of the generalized DS algorithm will degrade if the tail of $f(\boldsymbol{x})$ happens to be the mode of $g(\boldsymbol{x})$, since in D there may not

be enough points to choose to form a subsample with density $g(\boldsymbol{x})$. The same issue is of course present in the DS algorithm, since the uniform target density also will be much larger than $f(\boldsymbol{x})$ in its tail regions. Thus, the points in D falling in the tail regions of $f(\boldsymbol{x})$ will be depleted to quickly to allow for a uniform subsample over the tails. We consider this phenomenon in our examples in Section 2.5, where we also observe that this is a unavoidable problem for any existing diversity subsampling algorithm.

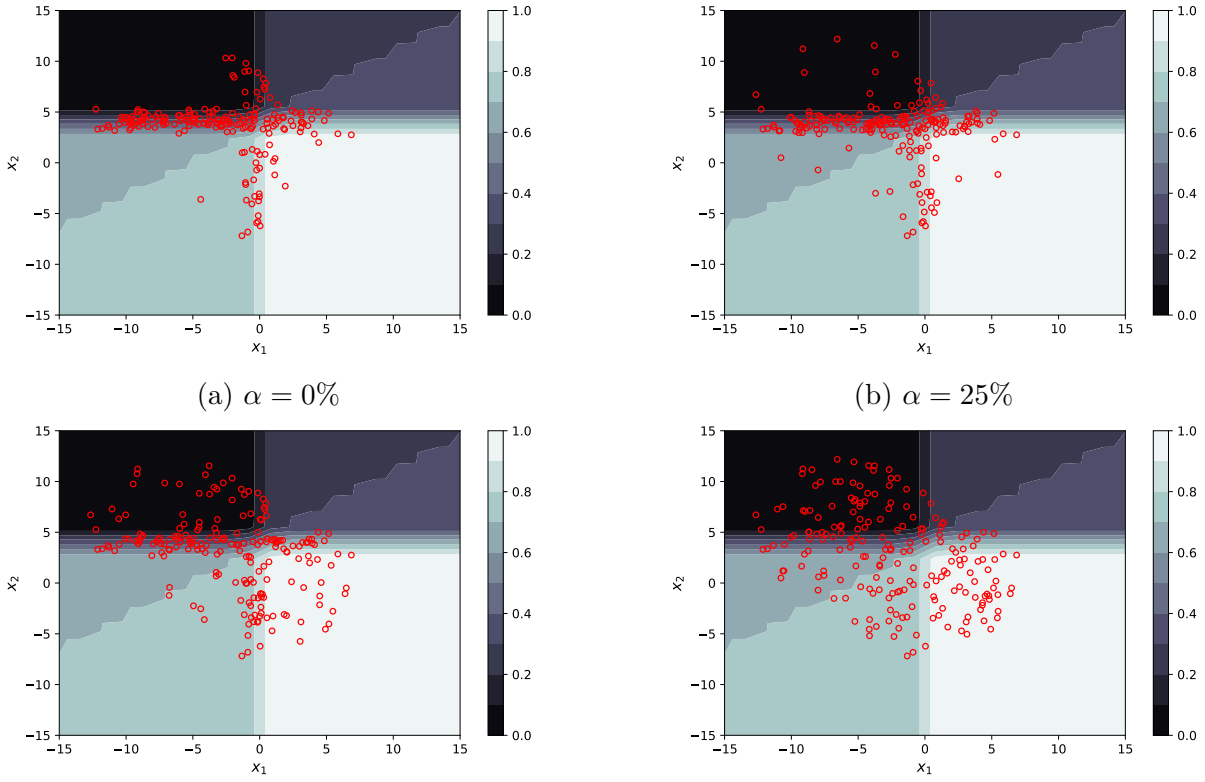
We now provide an example using the generalized DS algorithm in the case when $g(\boldsymbol{x})$ is not a well-defined p.d.f. For classification problems in active learning, uncertainty and diversity are two desirable properties of the selected subsample. And many works have found that a subsample incorporating both properties can be more beneficial (Ren et al. 2021). Here we provide a simple example illustrating how to use the generalized DS for such purposes.

Consider the following binary classification problem with $q = 2$. The data set D of predictor vectors is the same as in the MGM example described in Section 2.5.1, and we generate the binary response variable Y (= 0 or 1) via

$$P(Y = 1|\boldsymbol{X}) = P(Y = 1|X_1, X_2) = 0.3\frac{1}{1 + e^{-2X_1}} + 0.7\frac{1}{1 + e^{3(X_2-4)}}. \tag{2.6}$$

We use the Euclidean norm of the gradient of Equation (2.6) as a simple surrogate measure of classification uncertainty (a large gradient means the predicted probability $P(Y = 1|\boldsymbol{X})$ is changing rapidly in that region, which usually translates to higher classification uncertainty) and denote it as $u(\boldsymbol{x}) \stackrel{\text{def}}{=} \|\nabla_{\boldsymbol{X}} P(Y = 1|\boldsymbol{X})\|_2$. To incorporate both uncertainty and diversity into the subsample, we let $g(\boldsymbol{x}) = u(\boldsymbol{x}) + u_\alpha$, where u_α is a constant (and thus represents a uniform distribution to promote the diversity, whereas $u(\boldsymbol{x})$ considers

the classification uncertainty) that we take to be the lower α quantile of set $\{u(\boldsymbol{x}_i)\}_{i=1}^N$ for some specified α. Here $g(\boldsymbol{x})$ typically will not be a well-defined p.d.f. For the generalized DS algorithm, the sampling probability of each data point will be proportional to $\frac{g(\boldsymbol{x}_i)}{f(\boldsymbol{x}_i)}$. Selecting a larger α will promote more diversity, and selecting a smaller α will result in more subsample points chosen in the areas with higher uncertainty. Figure 2.3 shows the selected subsamples for various α. The colorbar shows the values of Equation (2.6). We observe that when $\alpha = 0$, most of the selected points locate close to the decision boundary $\boldsymbol{x} : P(Y = 1|\boldsymbol{x}) = 0.5$ (see Figure 2.3a), where the highest uncertainty occurs in this model. And using a larger α, such as $\alpha = 50\%$ allows more subsample points in other regions in the predictor space (see Figure 2.3c). And when $\alpha = 100\%$, the subsample appears to be quite diverse.



(a) $\alpha = 0\%$

(b) $\alpha = 25\%$

(c) $\alpha = 50\%$

(d) $\alpha = 100\%$

Figure 2.3. Subsample selected by generalized DS method with varying α. The red open circles indicate a selected subsample point. The color bar on the right side of each plot shows the value of the event probability specified in Equation (2.6).

2.4. Theoretical Performance Analysis

We state and prove the asymptotic performances of the generalized DS algorithm (see Section 2.3) in Lemma 1 and Theorem 1.

Lemma 1. *Let* $\boldsymbol{X}^N = (\boldsymbol{X}_i : i = 1, 2, \cdots, N)$, *where* $\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_N \in \mathbb{R}^q$ *are independently and identically distributed (i.i.d.) copies of random vector* $\boldsymbol{X}$. *We assume that these random variables are all defined on the same complete probability space. Let*

$\boldsymbol{X}$ have absolutely continuous cumulative distribution function (c.d.f) $F(\boldsymbol{x})$ and density function $f(\boldsymbol{x})$ with support $S \subset \mathbb{R}^q$. Let $g(\boldsymbol{x}) = c\tilde{g}(\boldsymbol{x})$ be the desired density function for the selected subsample that is known up to a constant $c \in \mathbb{R}^+$. Let $G(\boldsymbol{x})$ be the c.d.f corresponding to $g(\boldsymbol{x})$. Assume that the support of $g(\boldsymbol{x})$, $S_g \subset S$. Conditional on $\boldsymbol{X}^N$, let $\boldsymbol{Z}_N|\boldsymbol{X}^N$ be a discrete random variable drawn from $\boldsymbol{X}^N$ with probability mass function

$$P(\boldsymbol{Z}_N = \boldsymbol{X}_i|\boldsymbol{X}^N) = \frac{\frac{\tilde{g}(\boldsymbol{x})}{f(\boldsymbol{X}_i)}}{\sum_{j=1}^{N} \frac{\tilde{g}(\boldsymbol{x})}{f(\boldsymbol{X_j})}}, \tag{2.7}$$

for $i = 1, 2, \cdots, N$. Then as $N \to \infty$, $\boldsymbol{Z}_N$ converges in distribution to a random vector following distribution G, which we denote by $\boldsymbol{Z}_\infty \sim G$.

PROOF. Consider any $\boldsymbol{x} = (x_1, x_2, \cdots, x_q)^T \in \mathbb{R}^q$, and let $R(\boldsymbol{x}) = (-\infty, x_1] \times (-\infty, x_2] \times \cdots \times (-\infty, x_q] \subset \mathbb{R}^q$ denote the rectangle southwest of $\boldsymbol{x}$. The distribution function of $\boldsymbol{Z}_N$ is

$$F_{\boldsymbol{Z}_N}(\boldsymbol{x}) := P(\boldsymbol{Z}_N \in R(\boldsymbol{x})) = E[I_{R(\boldsymbol{x})}(\boldsymbol{Z}_N)] = E\left[E\left[I_{R(\boldsymbol{x})}(\boldsymbol{Z}_N)\middle|\boldsymbol{X}^N\right]\right], \tag{2.8}$$

where $I_{R(\boldsymbol{x})}(\boldsymbol{Z}_N)$ denotes the indicator function of the event $\boldsymbol{Z}_N \in R(\boldsymbol{x})$. In other words, $I_{R(\boldsymbol{x})}(\boldsymbol{Z}_N) = 1$ if $\boldsymbol{Z}_N \in R(\boldsymbol{x})$; and $I_{R(\boldsymbol{x})}(\boldsymbol{Z}_N) = 0$ otherwise. The last equality in Equation (2.8) holds by the law of total expectation. The inner expectation in Equation (2.8) is

$$
\begin{aligned}
E[I_{R(\boldsymbol{x})}(\boldsymbol{Z}_N)|\boldsymbol{X}^N] =& P(\boldsymbol{Z}_N \in R(\boldsymbol{x})|\boldsymbol{X}^N) = \sum_{i=1}^{N} P(\boldsymbol{Z}_N = \boldsymbol{X}_i|\boldsymbol{X}^N) I_{R(\boldsymbol{x})}(\boldsymbol{X}_i), \\
=& \frac{\sum_{i=1}^{N} \frac{\tilde{g}(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)} I_{R(\boldsymbol{x})}(\boldsymbol{X}_i)}{\sum_{i=1}^{N} \frac{\tilde{g}(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}} = \frac{\sum_{i=1}^{N} \frac{c\tilde{g}(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)} I_{R(\boldsymbol{x})}(\boldsymbol{X}_i)}{\sum_{i=1}^{N} \frac{c\tilde{g}(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}}, \\
=& \frac{\sum_{i=1}^{N} \frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)} I_{R(\boldsymbol{x})}(\boldsymbol{X}_i)}{\sum_{i=1}^{N} \frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}}.
\end{aligned} \tag{2.9}
$$

Combining Equations (2.8) and (2.9) and taking the limit as $N \to \infty$ gives

$$
\lim_{N\to\infty} F_{\boldsymbol{Z}_N}(\boldsymbol{x}) = \lim_{N\to\infty} E\left[\frac{\sum_{i=1}^{N} \frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)} I_{R(\boldsymbol{x})}(\boldsymbol{X}_i)}{\sum_{i=1}^{N} \frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}}\right]. \tag{2.10}
$$

Now consider the random variables $\frac{1}{N}\sum_{i=1}^{N} \frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}$ and $\frac{1}{N}\sum_{i=1}^{N} \frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)} I_{R(\boldsymbol{x})}(\boldsymbol{X}_i)$. Since $\boldsymbol{X}, \boldsymbol{X}_1, \cdots, \boldsymbol{X}_N$ are i.i.d. and $\forall \boldsymbol{z} \in S$, $f(\boldsymbol{z}) > 0$ and $g(\boldsymbol{z}) > 0$, we have $\forall i \in \{1, 2, \cdots, N\}$,

$$
E\left[\left|\frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}\right|\right] = E\left[\frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}\right] = \int_S \frac{g(\boldsymbol{z})}{f(\boldsymbol{z})} f(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z} = \int_S g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z} = \int_{S_g} g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z} = 1 < \infty, \tag{2.11}
$$

and

$$
\begin{aligned}
E\left[\left|\frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)} I_{R(\boldsymbol{x})}(\boldsymbol{X}_i)\right|\right] =& E\left[\frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)} I_{R(\boldsymbol{x})}(\boldsymbol{X}_i)\right] = \int_S \frac{g(\boldsymbol{z})}{f(\boldsymbol{z})} I_{R(\boldsymbol{x})}(\boldsymbol{z}) f(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}, \\
=& \int_{S_g} g(\boldsymbol{z}) I_{R(\boldsymbol{x})}(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z} = \int_{R(\boldsymbol{x})\cap S_g} g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z} = \int_{R(\boldsymbol{x})} g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z} < 1 < \infty.
\end{aligned} \tag{2.12}
$$

Therefore, by the strong law of large numbers,

$$\frac{1}{N}\sum_{i=1}^{N}\frac{g(\boldsymbol{X_i})}{f(\boldsymbol{X_i})} \xrightarrow{a.s.} 1, \text{ and} \tag{2.13}$$

$$\frac{1}{N}\sum_{i=1}^{N}\frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}I_{R(\boldsymbol{x})}(\boldsymbol{X}_i) \xrightarrow{a.s.} \int_{R(\boldsymbol{x})} g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}, \tag{2.14}$$

where the $\xrightarrow{a.s.}$ symbol denotes almost sure (a.s.) convergence, and so their ratio also converges a.s., i.e.,

$$\frac{\frac{1}{N}\sum_{i=1}^{N}\frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}I_{R(\boldsymbol{x})}(\boldsymbol{X}_i)}{\frac{1}{N}\sum_{i=1}^{N}\frac{g(\boldsymbol{X_i})}{f(\boldsymbol{X_i})}} \xrightarrow{a.s.} \int_{R(\boldsymbol{x})} g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}. \tag{2.15}$$

Noticing that $\left|\frac{\frac{1}{N}\sum_{i=1}^{N}\frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}I_{R(\boldsymbol{x})}(\boldsymbol{X}_i)}{\frac{1}{N}\sum_{i=1}^{N}\frac{g(\boldsymbol{X_i})}{f(\boldsymbol{X_i})}}\right| \leq 1$ a.s., we can apply bounded convergence theorem to obtain

$$\lim_{N\to\infty} E\left[\frac{\frac{1}{N}\sum_{i=1}^{N}\frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}I_{R(\boldsymbol{x})}(\boldsymbol{X}_i)}{\frac{1}{N}\sum_{i=1}^{N}\frac{g(\boldsymbol{X_i})}{f(\boldsymbol{X_i})}}\right] = E\left[\int_{R(\boldsymbol{x})} g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}\right] = \int_{R(\boldsymbol{x})} g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}. \tag{2.16}$$

Then by using Equations (2.10) and (2.16) we have

$$\begin{aligned}\lim_{N\to\infty} F_{\boldsymbol{Z}_N}(\boldsymbol{x}) &= \lim_{N\to\infty} E\left[\frac{\sum_{i=1}^{N}\frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}I_{R(\boldsymbol{x})}(\boldsymbol{X}_i)}{\sum_{i=1}^{N}\frac{g(\boldsymbol{X_i})}{f(\boldsymbol{X_i})}}\right] = \lim_{N\to\infty} E\left[\frac{\frac{1}{N}\sum_{i=1}^{N}\frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}I_{R(\boldsymbol{x})}(\boldsymbol{X}_i)}{\frac{1}{N}\sum_{i=1}^{N}\frac{g(\boldsymbol{X_i})}{f(\boldsymbol{X_i})}}\right] \\ &= \int_{R(\boldsymbol{x})} g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z},\end{aligned} \tag{2.17}$$

which implies $\boldsymbol{Z}_N$ converges in distribution to a random vector following distribution G. □

Theorem 1. *Let* $\boldsymbol{X}^N, \boldsymbol{X}, F(\boldsymbol{x}), f(\boldsymbol{x}), G(\boldsymbol{x}), g(\boldsymbol{x}), c, \tilde{g}(\boldsymbol{x}), S, S_g$ *and* $\boldsymbol{Z}_N$ *be as in the statement of Lemma 1, and define* $\boldsymbol{Z}_N^1 \stackrel{\mathsf{def}}{=} \boldsymbol{Z}_N$. *Suppose* $n \geq 2$ *is fixed. We further assume that* $\sup_{\boldsymbol{x}\in S} \frac{g(\boldsymbol{x})}{f(\boldsymbol{x})} < +\infty$. *For each* $N = 1, 2, \cdots$, *let* $(\boldsymbol{Z}_N^k : k = 2, 3, \cdots, n)$ *be drawn sequentially from* $\boldsymbol{X}^N$, *without replacement, according to the conditional probability mass function*

$$P(\boldsymbol{Z}_N^k = \boldsymbol{X}_i | \boldsymbol{X}^N, \boldsymbol{Z}_N^1, \cdots, \boldsymbol{Z}_N^{k-1}) = \frac{\frac{\tilde{g}(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}}{\sum_{j=1}^N \frac{\tilde{g}(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)} - \sum_{j=1}^{k-1} \frac{\tilde{g}(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}}, \tag{2.18}$$

for $i = 1, 2, \cdots, N$ *and* $i \notin \{j_1^N, j_2^N, \cdots, j_{k-1}^N\}$. *Here* $j_1^N, j_2^N, \cdots, j_n^N$ *are the indices of the sampled observations in* $\boldsymbol{X}^N$, *i.e.,* $\boldsymbol{Z}_N^k = \boldsymbol{X}_{j_k^N}$, *for* $k = 1, 2, \cdots, n$. *Then as* $N \to \infty$, *the joint distribution of* $(\boldsymbol{Z}_N^k : k = 1, 2, \cdots, n)$ *converges in distribution to* n *i.i.d. random vectors following distribution* G, *which we denote by* $(\boldsymbol{Z}_\infty^k : k = 1, 2, \cdots, n) \sim G^n$.

Proof. As in the proof of Lemma 1, we consider point $\boldsymbol{x} = (x_1, \cdots, x_q)^T \in \mathbb{R}^q$ and let $R(\boldsymbol{x}) = (-\infty, x_1] \times (-\infty, x_2] \times \cdots, \times (-\infty, x_q] \subset \mathbb{R}^q$ denote the the rectangle southwest of point $\boldsymbol{x}$. We prove Theorem 1 by induction. Note that when $n = 1$, Theorem 1 holds by Lemma 1. Now if we assume for some $n > 1$, $(\boldsymbol{Z}_\infty^k : k = 1, \cdots, n-1) \sim G^{n-1}$, the goal is to show $(\boldsymbol{Z}_\infty^k : k = 1, \cdots, n) \sim G^n$.

For any $n > 1$, the joint cumulative distribution function (c.d.f) of $(\boldsymbol{Z}_N^1, \boldsymbol{Z}_N^2, \cdots, \boldsymbol{Z}_N^n)$ is

$$
\begin{aligned}
& F_{(\boldsymbol{Z}_N^1, \cdots, \boldsymbol{Z}_N^n)}(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n) \\
=& P(\boldsymbol{Z}_N^1 \in R(\boldsymbol{x}_1), \cdots, \boldsymbol{Z}_N^n \in R(\boldsymbol{x}_n)) \\
=& E\left[\prod_{k=1}^{n} I_{R(\boldsymbol{x}_k)}(\boldsymbol{Z}_N^k)\right] \\
=& E\left[\left(\prod_{k=1}^{n-1} I_{R(\boldsymbol{x}_k)}(\boldsymbol{Z}_N^k)\right) E\left[I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^n) | \boldsymbol{X}^N, \boldsymbol{Z}_N^1, \cdots, \boldsymbol{Z}_N^{n-1}\right]\right],
\end{aligned} \tag{2.19}
$$

where $\boldsymbol{x}_i \in \mathbb{R}^q$, for $i = 1, 2, \cdots, n$. As before, the symbol $I_{R(\boldsymbol{x}_k)}(\boldsymbol{Z}_N^k)$ denotes the indicator function of $\boldsymbol{Z}_N^k \in R(\boldsymbol{x}_k)$. The third equality in Equation (2.19) holds due to the law of total expectation.

By design of sampling mechanism specified in Equation (2.18), the inner expectation in Equation (2.19) is

$$
\begin{aligned}
& E\left[I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^n) | \boldsymbol{X}^N, \boldsymbol{Z}_N^1, \cdots, \boldsymbol{Z}_N^{n-1}\right] \\
=& \sum_{i=1}^{N} \frac{\frac{\tilde{g}(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}}{\sum_{j=1}^{N} \frac{\tilde{g}(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)} - \sum_{j=1}^{n-1} \frac{\tilde{g}(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}} I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_i)(1 - I_{\{\boldsymbol{Z}_N^1, \cdots, \boldsymbol{Z}_N^{n-1}\}}(\boldsymbol{X}_i)) \\
=& \sum_{i=1}^{N} \frac{\frac{c\tilde{g}(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}}{\sum_{j=1}^{N} \frac{c\tilde{g}(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)} - \sum_{j=1}^{n-1} \frac{c\tilde{g}(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}} I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_i)(1 - I_{\{\boldsymbol{Z}_N^1, \cdots, \boldsymbol{Z}_N^{n-1}\}}(\boldsymbol{X}_i)) \\
=& \sum_{i=1}^{N} \frac{\frac{g(\boldsymbol{X}_i)}{f(\boldsymbol{X}_i)}}{\sum_{j=1}^{N} \frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)} - \sum_{j=1}^{n-1} \frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}} I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_i)(1 - I_{\{\boldsymbol{Z}_N^1, \cdots, \boldsymbol{Z}_N^{n-1}\}}(\boldsymbol{X}_i)) \\
=& \frac{\sum_{j=1}^{N} \frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)} I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_j) - \sum_{j=1}^{n-1} \frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)} I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^j)}{\sum_{j=1}^{N} \frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)} - \sum_{j=1}^{n-1} \frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}}
\end{aligned} \tag{2.20}
$$

Equations (2.19) and (2.20) together yield

$$
\begin{aligned}
&F_{(\boldsymbol{Z}_N^1,\cdots,\boldsymbol{Z}_N^n)}(\boldsymbol{x}_1,\cdots,\boldsymbol{x}_n) \\
=&E\left[\left(\prod_{k=1}^{n-1} I_{R(\boldsymbol{x}_k)}(\boldsymbol{Z}_N^k)\right)\frac{\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_j)-\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^j)}{\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}-\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}}\right],
\end{aligned}
\tag{2.21}
$$

the limit of which we seek as $N\to\infty$. Towards this end, consider the convergence of random sequence $\left(\prod_{k=1}^{n-1} I_{R(\boldsymbol{x}_k)}(\boldsymbol{Z}_N^k)\right)\frac{\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_j)-\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^j)}{\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}-\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}}$ as $N\to\infty$.

First consider the convergence of $\prod_{k=1}^{n-1} I_{R(\boldsymbol{x}_k)}(\boldsymbol{Z}_N^k)$ as $N\to\infty$. By the induction assumption, we have

$$
(\boldsymbol{Z}_N^1,\cdots,\boldsymbol{Z}_N^{n-1})\xrightarrow{D} G^{n-1},
\tag{2.22}
$$

as $N\to\infty$, where symbol $\xrightarrow{D}$ denotes convergence in distribution. We use the notation $G^{n-1}=(G_1,\cdots,G_{n-1})$ with $G_1,\cdots,G_{n-1}$ being i.i.d. random vectors following distribution G. By the continuous mapping theorem (Theorem 2.3 in Van der Vaart 2000), we obtain

$$
\prod_{k=1}^{n-1} I_{R(\boldsymbol{x}_k)}(\boldsymbol{Z}_N^k)\xrightarrow{D}\prod_{k=1}^{n-1} I_{R(\boldsymbol{x}_k)}(G_k).
\tag{2.23}
$$

Next we discuss the convergence of $\frac{\frac{1}{N}\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_j)-\frac{1}{N}\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^j)}{\frac{1}{N}\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}-\frac{1}{N}\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}}$ as $N\to\infty$. We do so by considering the convergence of random sequences $\frac{1}{N}\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_j)$, $\frac{1}{N}\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}$, $\frac{1}{N}\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^j)$, $\frac{1}{N}\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}$. In the proof of Lemma 1, we have

argued that (see Equations (2.13) and (2.14)), as $N \to \infty$,

$$\frac{1}{N}\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_j) \xrightarrow{a.s.} \int_{R(\boldsymbol{x}_n)} g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z} \text{ and,} \tag{2.24}$$

$$\frac{1}{N}\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)} \xrightarrow{a.s.} 1, \tag{2.25}$$

where symbol $\xrightarrow{a.s.}$ denotes almost sure convergence. Since $\sup_{\boldsymbol{x}\in S}\frac{g(\boldsymbol{x})}{f(\boldsymbol{x})} < +\infty$, by using the first Borel-Cantelli Lemma (see Theorem 18.1 in Gut 2013), it is easy to show that as $N \to \infty$, $\forall \epsilon > 0$,

$$P\left(\left|\frac{1}{N}\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}\right| \geq \epsilon \text{ i.o.}\right) = 0 \text{ and,} \tag{2.26}$$

$$P\left(\left|\frac{1}{N}\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^j)\right| \geq \epsilon \text{ i.o.}\right) = 0, \tag{2.27}$$

where i.o. stands for infinitely often. Therefore,

$$-\frac{1}{N}\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)} \xrightarrow{a.s.} 0 \text{ and,} \tag{2.28}$$

$$-\frac{1}{N}\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^j) \xrightarrow{a.s.} 0. \tag{2.29}$$

Since by assumption the probability space is complete, Equations (2.24), (2.25), (2.28) and (2.29) together yield

$$
\begin{aligned}
&\frac{\sum_{j=1}^{N} \frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)} I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_j) - \sum_{j=1}^{n-1} \frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)} I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^j)}{\sum_{j=1}^{N} \frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)} - \sum_{j=1}^{n-1} \frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}} \\
=&\frac{\frac{1}{N}\sum_{j=1}^{N} \frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)} I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_j) - \frac{1}{N}\sum_{j=1}^{n-1} \frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)} I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^j)}{\frac{1}{N}\sum_{j=1}^{N} \frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)} - \frac{1}{N}\sum_{j=1}^{n-1} \frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}} \\
\xrightarrow{a.s.}& \int_{R(\boldsymbol{x}_n)} g(\boldsymbol{z})\, \mathrm{d}\boldsymbol{z},
\end{aligned} \tag{2.30}
$$

where we used the properties of almost sure convergence and continuous mapping theorem (Theorem 2.3 in Van der Vaart 2000).

Noticing that the limit in Equation (2.30) is a constant, we apply Slutsky's Theorem (Slutsky 1925) with Equations (2.23) and (2.30) and obtain

$$
\begin{aligned}
&\left(\prod_{k=1}^{n-1} I_{R(\boldsymbol{x}_k)}(\boldsymbol{Z}_N^k)\right) \frac{\sum_{j=1}^{N} \frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)} I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_j) - \sum_{j=1}^{n-1} \frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)} I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^j)}{\sum_{j=1}^{N} \frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)} - \sum_{j=1}^{n-1} \frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}} \\
\xrightarrow{D}& \int_{R(\boldsymbol{x}_n)} g(\boldsymbol{z})\, \mathrm{d}\boldsymbol{z} \prod_{k=1}^{n-1} I_{R(\boldsymbol{x}_k)}(G_k),
\end{aligned} \tag{2.31}
$$

as $N \to \infty$.

Now consider the random variable on the left-hand-side of Equation (2.31). We have

$$
\begin{aligned}
& P\left(\left|\left(\prod_{k=1}^{n-1} I_{R(\boldsymbol{x}_k)}(\boldsymbol{Z}_N^k)\right)\frac{\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_j)-\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^j)}{\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}-\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}}\right|\leq 1\right) \\
\geq & P\left(\left|\frac{\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_j)-\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^j)}{\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}-\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}}\right|\leq 1\right) \\
\geq & P\left(\left|\frac{\sum_{j\in\{1,\cdots,N\}\setminus\{j_1^N,\cdots,j_{n-1}^N\}}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_j)}{\sum_{j\in\{1,\cdots,N\}\setminus\{j_1^N,\cdots,j_{n-1}^N\}}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}}\right|\leq 1\right) \\
= & 1
\end{aligned}
\tag{2.32}
$$

Thus the random variable $\left(\prod_{k=1}^{n-1} I_{R(\boldsymbol{x}_k)}(\boldsymbol{Z}_N^k)\right)\frac{\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_j)-\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^j)}{\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}-\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}}$ is bounded from above by a constant with probability 1, which indicates that it is uniformly integrable (Theorem 4.4 of Gut 2013). Further considering Equation (2.31), we conclude (see Thm 25.12 in Billingsley 1995 on page 338) that as $N\to\infty$,

$$
\begin{aligned}
& E\left[\left(\prod_{k=1}^{n-1} I_{R(\boldsymbol{x}_k)}(\boldsymbol{Z}_N^k)\right)\frac{\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{X}_j)-\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}I_{R(\boldsymbol{x}_n)}(\boldsymbol{Z}_N^j)}{\sum_{j=1}^{N}\frac{g(\boldsymbol{X}_j)}{f(\boldsymbol{X}_j)}-\sum_{j=1}^{n-1}\frac{g(\boldsymbol{Z}_N^j)}{f(\boldsymbol{Z}_N^j)}}\right] \\
\to & E\left[\int_{R(\boldsymbol{x}_n)} g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}\prod_{k=1}^{n-1} I_{R(\boldsymbol{x}_k)}(G_k)\right] \\
= & \int_{R(\boldsymbol{x}_n)} g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}\int_{S_g}\cdots\int_{S_g}\left(\prod_{k=1}^{n-1} I_{R(\boldsymbol{x}_k)}(\boldsymbol{y}_k)g(\boldsymbol{y}_k)\right)\mathrm{d}\boldsymbol{y}_1\cdots\mathrm{d}\boldsymbol{y}_{n-1} \\
= & \int_{R(\boldsymbol{x}_n)} g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}\prod_{k=1}^{n-1}\int_{R(\boldsymbol{x}_k)} g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z} \\
= & \prod_{k=1}^{n}\int_{R(\boldsymbol{x}_k)} g(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}.
\end{aligned}
\tag{2.33}
$$

The result follows immediately from Equations (2.21) and (2.33). □

Remark 1. *Theorem 1 applies to the DS algorithm (Procedure 1) for producing a uniform sample by setting* $\tilde{g}(\boldsymbol{x}) = 1$, $\forall \boldsymbol{x} \in S_g$ *in Lemma 1 and Theorem 1.*

The significance of Theorem 1 is that it shows that, as $N \to \infty$, the generalized DS algorithm does indeed produce an i.i.d. sample following the desired target distribution G, providing that the support of f contains the support of g and that $\frac{g}{f}$ is bounded from above by a finite number.

2.5. Numerical Performance Analysis of the DS Algorithm

In this section, we test the performance of the DS algorithm using data sets following various distributions and compare it with competing subsampling methods. Section 2.5.1 discusses performance evaluation criteria and the experimental settings for the comparative examples. Section 2.5.2 provides numerical results comparing uniformity performances of DS with competing algorithms, and Section 2.5.3 compares runtime of the algorithms.

2.5.1. Experimental Settings

As a performance evaluation criterion, we use the sample version of energy distance (G. Székely 2003; Gábor J Székely and Rizzo 2004) to measure the extent to which a subsample is drawn from an i.i.d uniform distribution over some specified $\Omega \subset S$, where Ω is compact. We provide further details of the choice of Ω below. Let $\boldsymbol{X}, \boldsymbol{U}$ be mutually independent random variables in $\mathbb{R}^q$. The sample version of energy distance is as follows. Let $\mathcal{A}_n = \{\boldsymbol{X}_1, \cdots, \boldsymbol{X}_n\}$ and $\mathcal{U} = \{\boldsymbol{U}_1, \cdots, \boldsymbol{U}_{N_1}\}$ be i.i.d. samples of size n and N_1, respectively,

from the same distributions followed by $\boldsymbol{X}$ and $\boldsymbol{U}$, respectively. The energy distance between samples $\mathcal{A}_n$ and $\mathcal{U}$ is (G. Székely 2003)

$$(2.34)\quad e(\mathcal{A}_n,\mathcal{U}) = \frac{2}{nN_1}\sum_{i=1}^{n}\sum_{j=1}^{N_1}\|\boldsymbol{X}_i-\boldsymbol{U}_j\| - \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\|\boldsymbol{X}_i-\boldsymbol{X}_j\| - \frac{1}{N_1^2}\sum_{i=1}^{N_1}\sum_{j=1}^{N_1}\|\boldsymbol{U}_i-\boldsymbol{U}_j\|.$$

The Euclidean norm will be used throughout this chapter unless otherwise specified. By Proposition 1 in G. Székely 2003, as $n \to \infty$ and $N_1 \to \infty$, one can show that $E[e(\mathcal{A}_n,\mathcal{U})] \to 0+$ if and only if $\boldsymbol{X} \overset{\mathsf{D}}{=} \boldsymbol{U}$. Here $\overset{\mathsf{D}}{=}$ denotes equality in distribution.

We use Equation (2.34) in the following way. We choose $\mathcal{U}$ to be a large uniform sample over Ω of size $N_1 >> n$, and we take $\mathcal{A}_n$ to be the subsample of size n selected from D using some subsampling algorithm. If points in subsample $\mathcal{A}_n$ (subsample points located outside of Ω are excluded under this criterion) follow an i.i.d uniform distribution over Ω, $E[e(\mathcal{A}_n,\mathcal{U})]$ should be close to zero if n is sufficiently large (we take a fixed large N_1 in this chapter), and the more $\mathcal{A}_n$ differs from an i.i.d uniform distribution over Ω, the larger we expect $E[e(\mathcal{A}_n,\mathcal{U})]$ to be.

Intuitively, in order to produce a diverse and space-filling subsample, it is typically desirable to select all or most of the points in the regions for which $f(\boldsymbol{x})$ has low-density, which we denote by Ω^c, where $S = \Omega \cup \Omega^c$. Since we cannot expect uniformity over both Ω and Ω^c for large n (unless N is extremely large) as the points in Ω^c will be quickly depleted, we use a separate measure for performances within Ω^c. Since data are sparse in Ω^c, it is generally desirable to select as high a percentage of points in Ω^c as possible. Consequently, we define the following low-density region sampling ratio to measure this property. Suppose Ω is chosen such that $\forall \boldsymbol{x} \in \Omega$, $f(\boldsymbol{x}) \geq \delta$ and $\forall \boldsymbol{x} \in \Omega^c$, $f(\boldsymbol{x}) < \delta$, for some specified value δ. Then for a subsample $\mathcal{S}_n$ of size n selected from D, the low-density

region sampling ratio of $\mathcal{S}_n$ is defined as

$$r(\mathcal{S}_n) = \frac{\sum_{k=1}^{n} I_{\Omega^c}(\boldsymbol{x}_{j_k})}{\sum_{i=1}^{N} I_{\Omega^c}(\boldsymbol{x}_i)}. \tag{2.35}$$

In situations where a uniform subsample over S of size n will exhaust points in Ω^c, a larger $r(\mathcal{S}_n)$ is desirable, with $r(\mathcal{S}_n) = 1$ typically being the ideal goal (in conjunction with a uniform distribution over Ω).

For the examples, we consider data sets in which each element of $\boldsymbol{X}$ follows independent Gaussian, gamma, exponential, geometric, and multivariate Gaussian mixture (MGM) distributions. We refer to these five different data distributions as the normal, gamma, exponential, geometric, and MGM. The normal example illustrates the performances of the subsampling algorithms when the data distribution is symmetric; the mode of the density is achieved in the interior region of S; S is unbounded. The gamma example illustrates the performances when the data distribution is skewed and S had boundaries but is unbounded. The exponential distribution, which is an extreme case of the gamma distribution, illustrates the performances when the data distribution is extremely skewed and the mode is located at the boundary. The MGM examples are to compare the subsampling algorithms when the data distribution is multimodal with correlated predictors. We also include geometric examples to illustrate the performances of different algorithms when the data distribution is discrete. For all examples in this section, the size of D is $N = 10^4$ for $q = 2$ and $N = 10^5$ for $q = 10$.

The data distributions of each simulation example are as follows. For the normal, gamma, and exponential examples, the elements of $\boldsymbol{X}$ are independent and the data distribution has density $f(\boldsymbol{x}) = \prod_{j=1}^{q} f(x_j)$, where $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$, $f(x) = \frac{1}{4\Gamma(2)} x e^{-\frac{x}{2}}$ and

$f(x) = e^{-x}$, respectively, where $\Gamma()$ denotes the gamma function. For the MGM example, D was generated as a mixture of two Gaussian clusters in $\mathbb{R}^q$. The true density function is as follows.

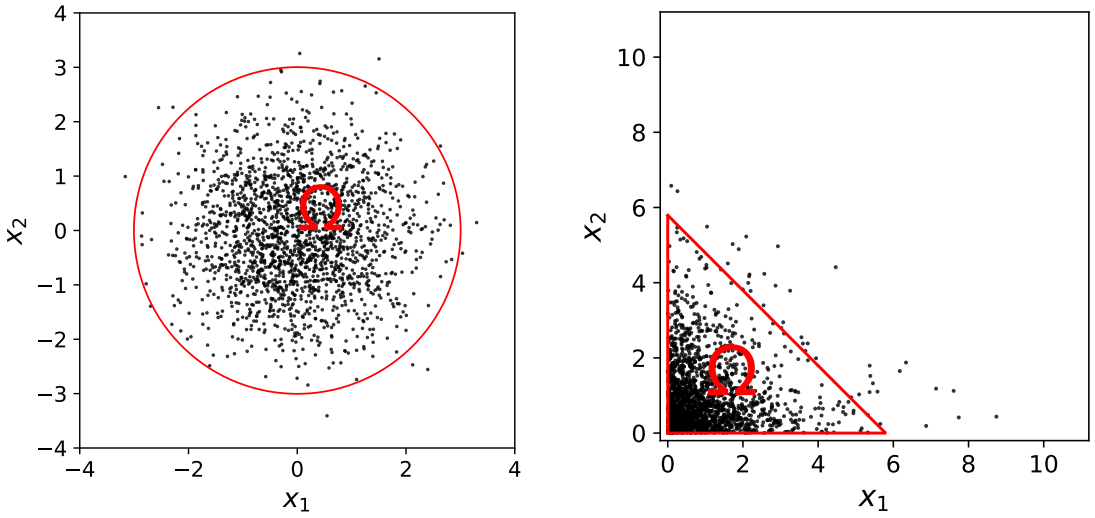
$$f(\boldsymbol{x}) = 0.5 f_1(\boldsymbol{x}; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}) + 0.5 f_2(\boldsymbol{x}; \boldsymbol{\mu}_2, \boldsymbol{\Sigma}), \tag{2.36}$$

where

$$f_i(\boldsymbol{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^q |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_i)\right), \tag{2.37}$$

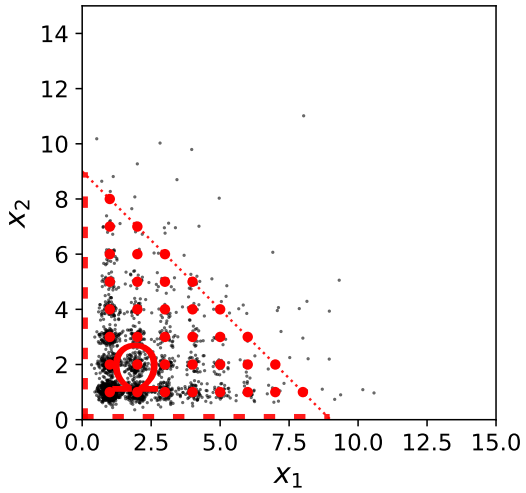
for $i = 1, 2$. That is, the two Gaussian clusters have a common covariance matrix $\boldsymbol{\Sigma}$ but different means $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2$. We let $\boldsymbol{\mu}_1 = \mathbf{0}_q$ and $\boldsymbol{\mu}_2 = 5.0((-1)^i)_{i=1}^q$. For the common covariance matrix $\boldsymbol{\Sigma}$, we set $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}_{q \times q} + \alpha \boldsymbol{a}\boldsymbol{a}^T$. Here $\sigma^2 = 4.0$, $\alpha = 1.0$, $\boldsymbol{I}_{q \times q} \in \mathbb{R}^{q \times q}$ is the identity matrix, and $\boldsymbol{a} = (a_i)_{i=1}^q$, for $a_i = 0.2(i-2)(-1)^i$. The data distribution of the geometric example also has independent covariates; thus the joint probability mass function (p.m.f) is $f(\boldsymbol{x}) = \prod_{j=1}^q f(x_j)$, where $f(x) = (1-p)^{x-1}p$. We set $p = 0.5$ for $q = 2$ and $p = 0.9$ for $q = 10$. The supports of the p.d.f functions of the normal, gamma, exponential, MGM and geometric examples are respectively $S = \mathbb{R}^q, \mathbb{R}_+^q, \mathbb{R}_+^q, \mathbb{R}^q, \mathbb{Z}_+^q$.

We choose $\Omega = \{\boldsymbol{x} \in S | f(\boldsymbol{x}) \geq \delta\}$, where δ is chosen such that 99% of the data points in D fall inside of Ω when $q = 2$ and 99.9% of the data points in D do so when $q = 10$. Figure 2.4 depicts Ω for selected examples with $q = 2$. To be clear, for discrete distributions, the target uniform distribution is a discrete uniform distribution over S. This distinction between continuous and discrete distributions is not used in any way in the DS algorithm, although it is used in our performance measures.



(a) Normal example

(b) Exponential example



(c) Geometric example

Figure 2.4. Depictions of the distribution of D (black dots) for $N = 2000$ and the chosen regions Ω (the boundaries of which are the red solid curves) for the normal, exponential, and geometric examples used in this section for $q = 2$. For normal and exponential examples, the red curve indicates the boundary of Ω. For the geometric example, the red dotted curves show region $\{\boldsymbol{x} \in \mathbb{R}^q | f(\boldsymbol{x}) \geq \delta\}$, and the union of solid red dots corresponds to the region $\Omega = S \cap \{\boldsymbol{x} \in \mathbb{R}^q | f(\boldsymbol{x}) \geq \delta\}$, which consists of a finite number of discrete points. 2000 randomly chosen data points are plotted in this graph for each example and are shown by black dots. The geometric data is perturbed for easy visualization.

2.5.2. Numerical Results for Performance Comparison

The competing algorithms that we compare include scSampler-sp1, scSampler-sp16, DPP, WSE (Weighted Sample Elimination algorithm, an heuristic for PDS proposed by Yuksel 2015), and simple random sampling. Here scSampler-sp1 (Song et al. 2022b) serves as an efficient heuristic for the CADEX algorithm. scSampler-sp16 (Song et al. 2022b) is included as a modification of scSampler-sp1 for better computational efficiency. We used published codes for scSampler-sp1 and scSampler-sp16 (Song et al. 2022a) , DPP (Bıyık 2019) and WSE (Yuksel 2016). We omit the results for RD ALR in this chapter, as we found that it generally did not perform as well as other methods.

The implementation details for competing methods are as follows. For the two versions of the scSampler algorithm, no additional parameters need to be set. For DPP, we set hyperparameters $\alpha = 4.0$, $\gamma = 0.0$ and $steps = 0$ (Bıyık et al. 2019 recommended setting $\alpha >= 2.0$ and $\gamma = 0.0$ in their chapter to get the most diverse subsample; we chose $steps = 0$ for computational efficiency at the cost of the resulted subsample being deterministic). For WSE, we used the default weighting function in Yuksel 2016 and set $Progressive = True$ to make the selected subsample sequential.

2.5.2.1. Visual Illustration of Typical Subsamples for Various Methods. This section shows typical subsamples of the random sampling, DS, scSampler-sp1, scSampler-sp16, DPP and WSE at $n = 200$, $q = 2$ using the normal example. From Figures 2.5b to 2.5f we see that the subsamples selected by DS, scSampler-sp1, scSampler-sp16, DPP and WSE all appear to be spread-out over S. The subsample selected by Random Sampling (see Figure 2.5a) appears concentrated in the region where the data points are

dense. Different from the subsamples selected by scSampler-sp1, and DPP, the subsample selected by DS (see Figure 2.5b) is non-repulsive, i.e. allows replicates to occur, and looks like a uniform sample over S, while the subsamples selected by scSampler-sp1 and DPP (Figures 2.5c and 2.5e) are repulsive, i.e. do not allow replicates to occur, and looks like Poisson disk distribution (see McCool and Fiume 1992 for definition) samples over S. Although aiming at selecting a subsample following the Poisson disk distribution, the WSE subsample (Figure 2.5f) does have subsample points located very close to each other. Different from the DS and scSampler-16 (Figures 2.5b and 2.5d) subsamples, the WSE subsample does not appear to be similar to either a uniform sample over S or a Poisson disk sample over S. The DS subsample and scSampler-sp16 subsample appear visually similar at $n = 200$ for the normal example at $q = 2$. Section 2.5.2.2 shows that the DS subsamples actually have superior uniform property than the subsamples selected by scSampler-sp16.

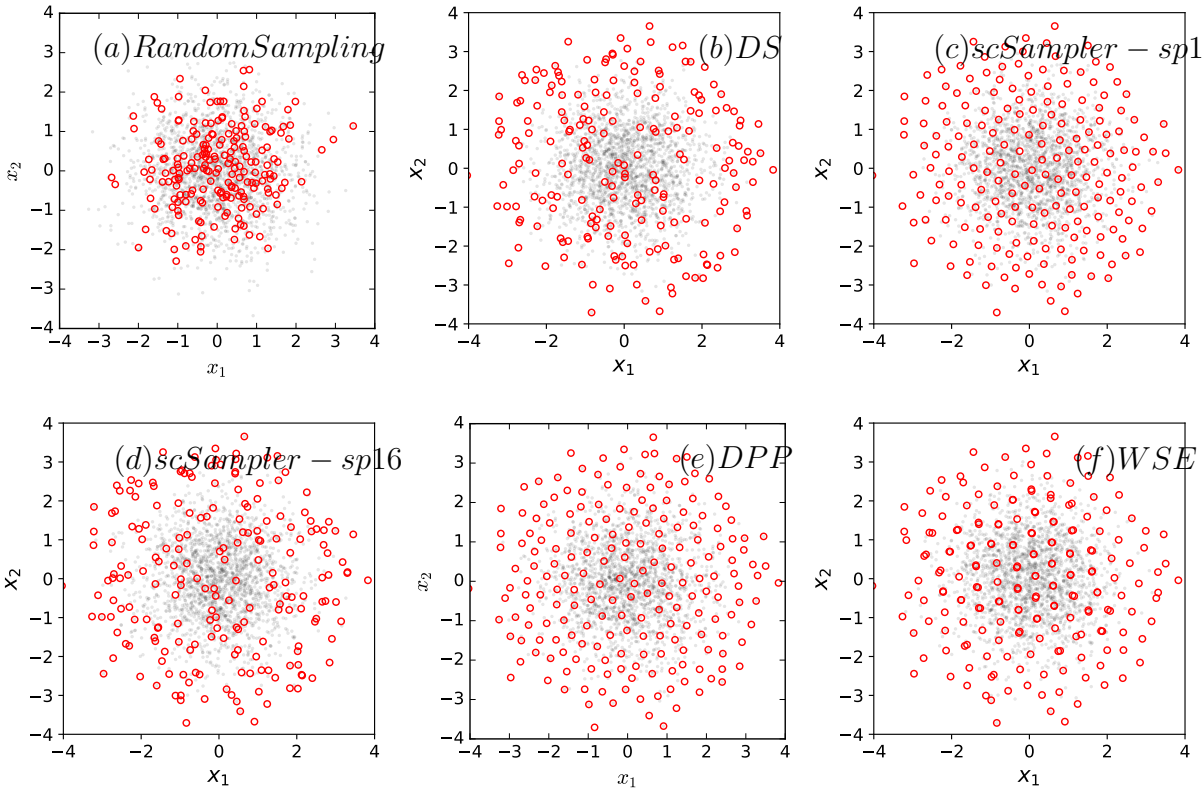
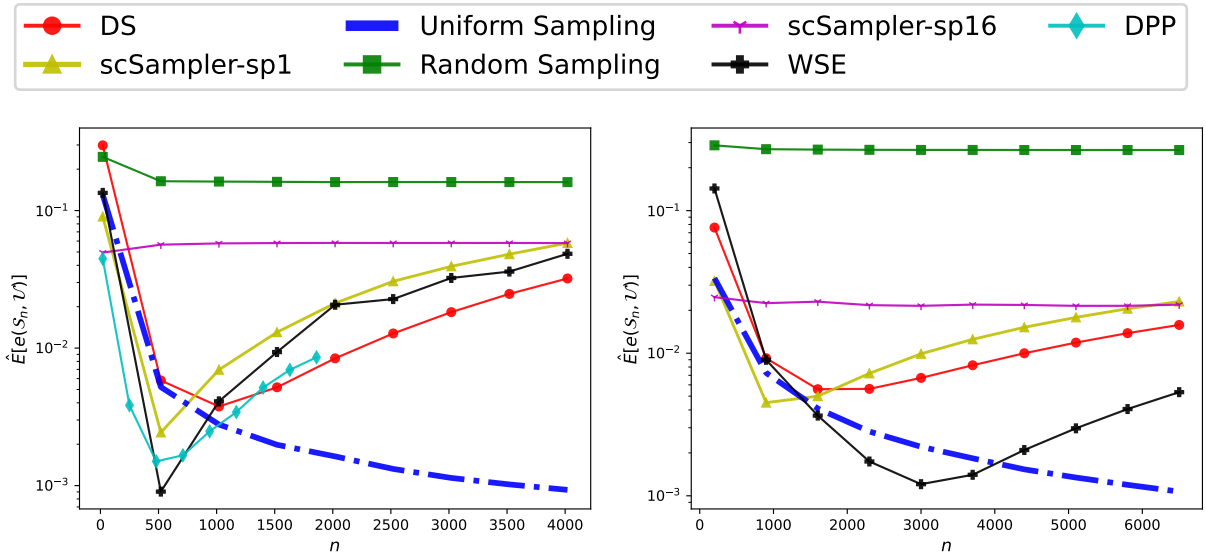


Figure 2.5. Typical subsamples selected by random sampling, DS, scSampler-sp1, scSampler-sp16, DPP and WSE at $n = 200$ and $q = 2$ using the normal example. Rep open circles represent selected points. Gray dots represent 2000 randomly chosen data points from D with equal probabilities.

2.5.2.2. Numerical Results using Quantified Measurements. Figure 2.6 shows that average value of $e(\mathcal{S}_n, \mathcal{U})$ for Uniform Sampling converges to 0 as n increases, whereas for most subsampling algorithms it first decreases and then increases instead of monotonically decreasing with n. This is because Uniform Sampling generates sample points from Ω directly, while the subsampling algorithms select points from a finite D without replacement. After points in D in the lower-density regions in Ω are depleted, the subsampling

algorithms can no longer produce uniform samples. We discuss this phenomenon in more detail in Appendix B.2.

We also observe from Figure 2.6 that some algorithms have an average $e(\mathcal{S}_n, \mathcal{U})$ measure that is actually less than Uniform Sampling, such as DPP and WSE for the normal example at $q = 2$ and $n \leq 1000$ (see Figure 2.6a). This is a well-known phenomenon, by which points on a uniform grid can have a lower $e(\mathcal{S}_n, \mathcal{U})$ measure than a truly uniform sample (Mak and V Roshan Joseph 2018). For instance, in separate experiments (not shown here, for brevity), we found that, a grid of size $n = 100$ in $[0, 1]^2$ often has a smaller sample energy distance to $\mathcal{U}$ than a true uniformly distributed sample. Consequently, we can view a subsampling algorithm whose performance is as close to the Uniform Sampling as possible as the most effective method in terms of selecting i.i.d uniformly distributed subsamples over Ω. From Figure 2.6, we see that overall, the average sample energy distances of DS deviates least from the Uniform Sampling results for most cases, compared with other subsampling algorithms.

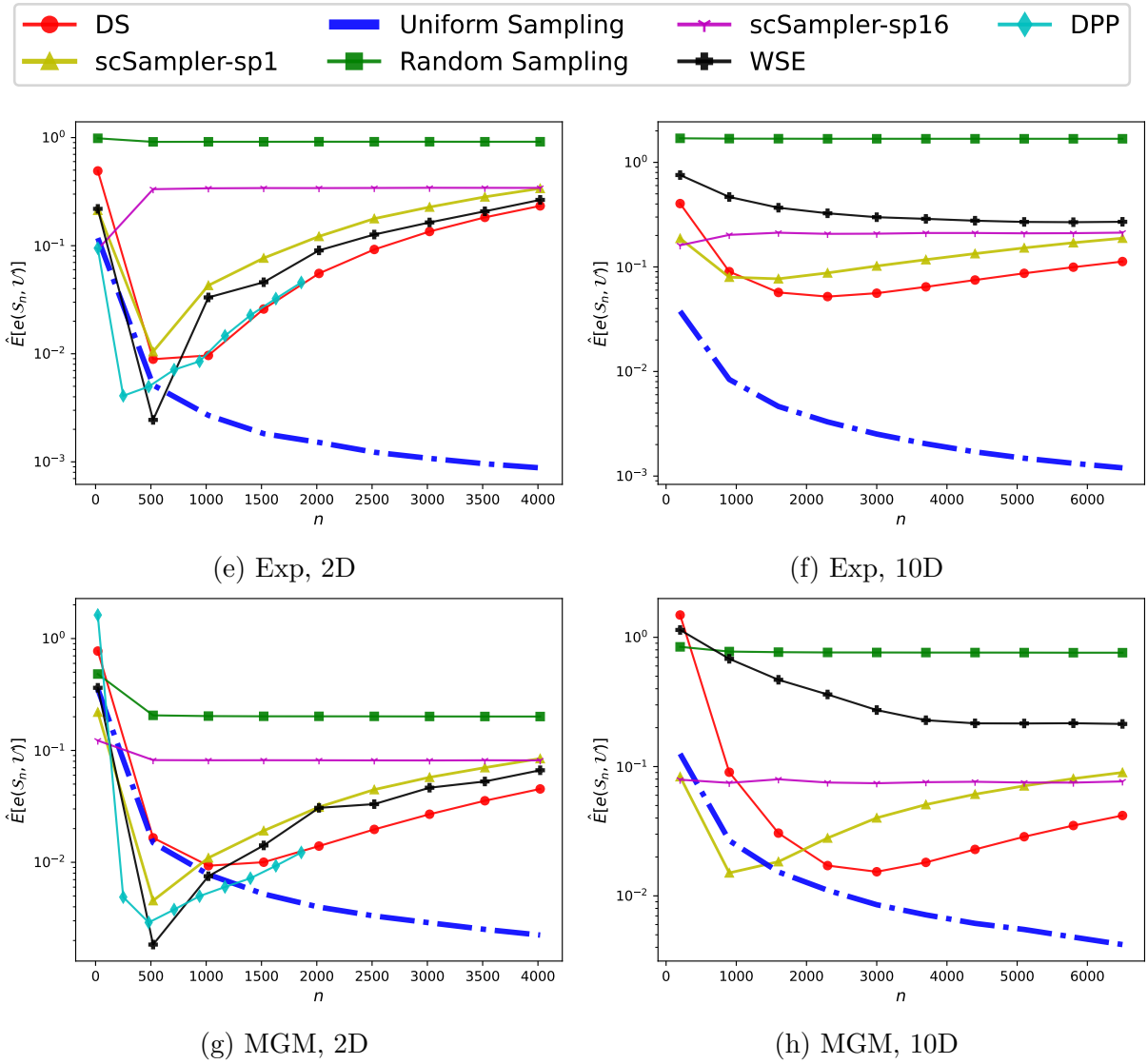


(a) Normal, 2D

(b) Normal, 10D

(c) Gamma, 2D

(d) Gamma, 10D

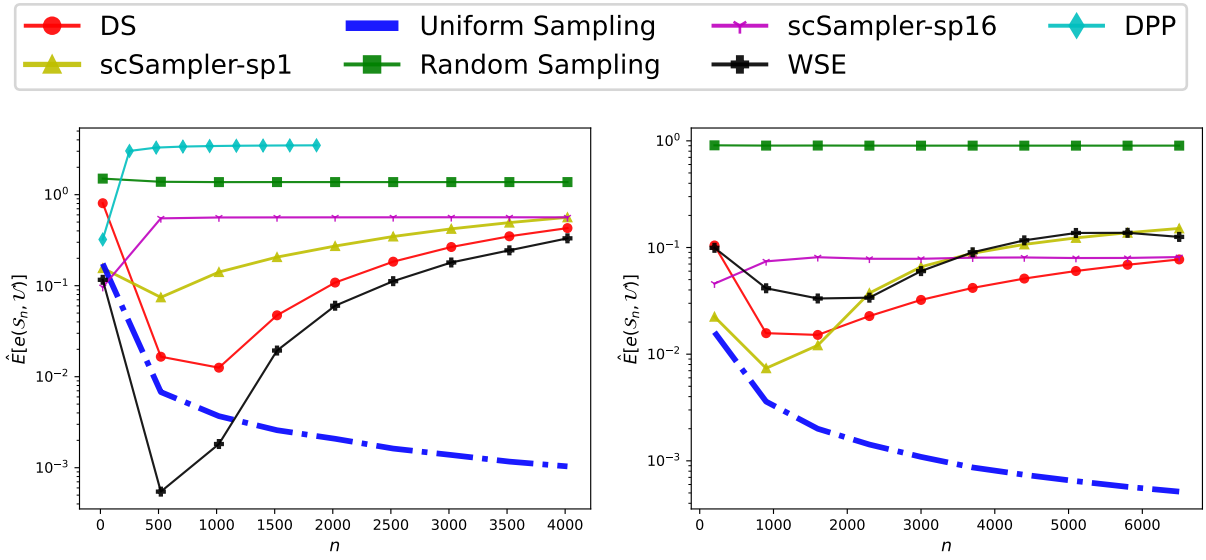


(e) Exp, 2D

(f) Exp, 10D

(g) MGM, 2D

(h) MGM, 10D

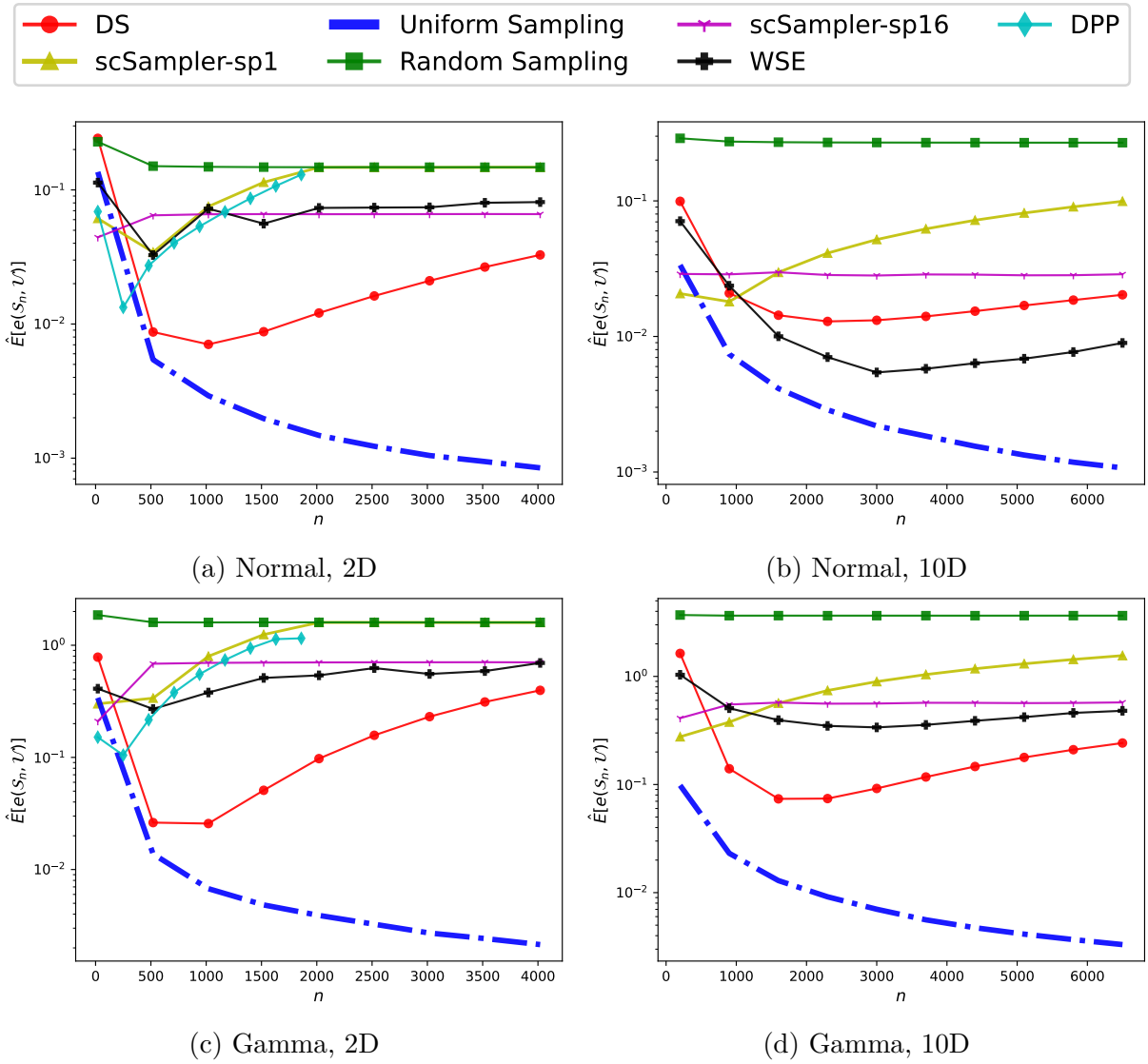


(i) Geometric, 2D

(j) Geometric, 10D

Figure 2.6. Averaged $e(\mathcal{S}_n, \mathcal{U})$, as a function of n, for subsamples selected by DS, random sampling from D ('Random Sampling'), scSampler-sp1, scSampler-sp16, WSE, and DPP. Uniform Sampling serves as a reference (the closer an algorithm is to the Uniform Sampling curve, the better).

We also repeated the experiments in Figure 2.6 using data sets in which observations were replicated, for the continuous distributions. Specifically, we generated a data set of size $\frac{N}{5}$ and then replicated this data set five times to generate the set D of size N, which we refer to as replicated data sets. The results using the replicated data sets are shown by Figure 2.7. Compared to Figure 2.6, the DS performance has barely changed, while the scSampler-sp1 performance has degraded significantly. We conclude that the performance of DS relative to the existing methods further improves, often substantially, for replicated data.

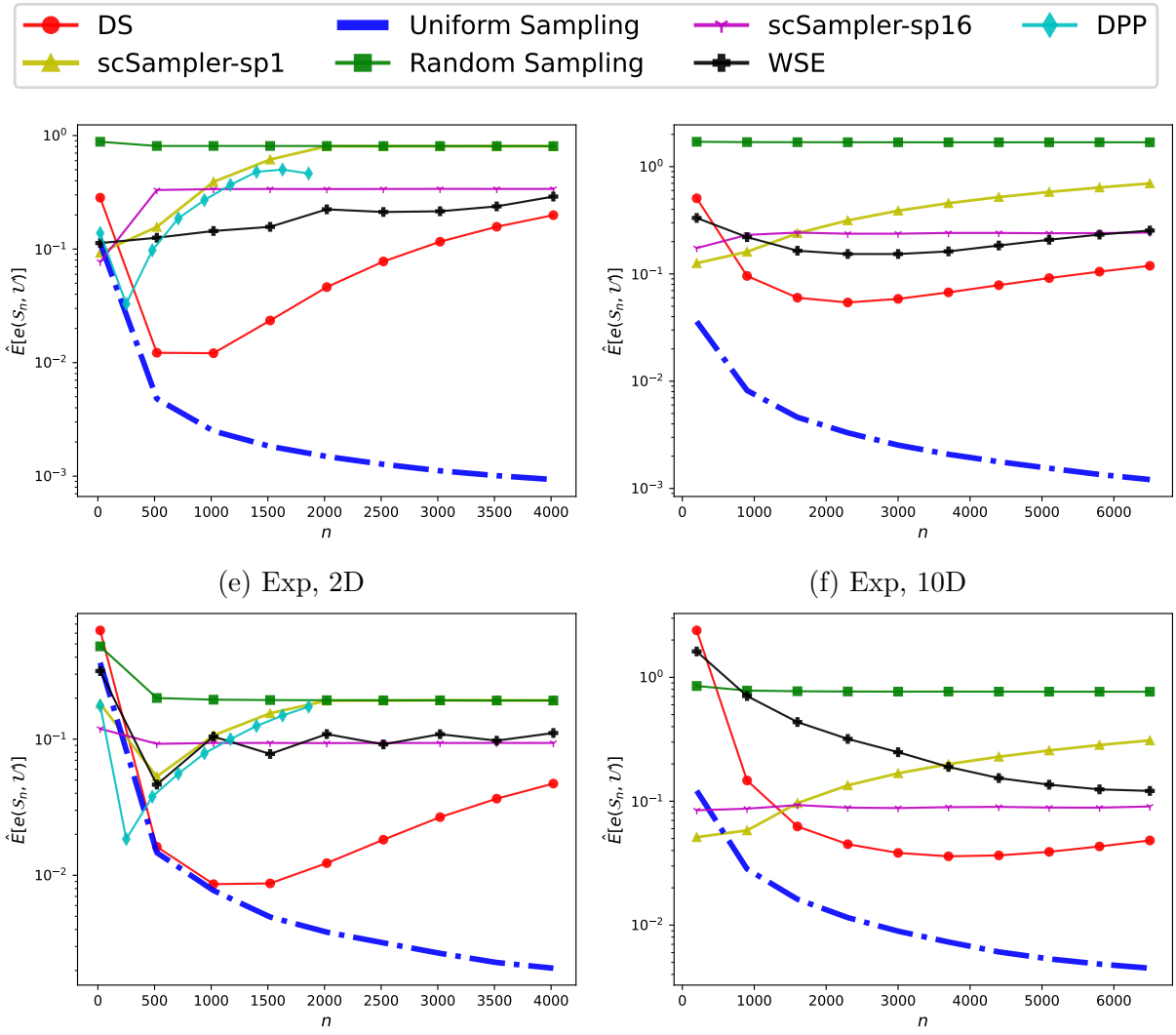


(a) Normal, 2D

(b) Normal, 10D

(c) Gamma, 2D

(d) Gamma, 10D



(e) Exp, 2D

(f) Exp, 10D

(g) MGM, 2D

(h) MGM, 10D

Figure 2.7. Averaged $e(\mathcal{S}_n, \mathcal{U})$, as a function of n, using replicated data, for subsamples selected by DS, random sampling from D ('Random Sampling'), scSampler-sp1, scSampler-sp16, WSE, and DPP. Uniform Sampling serves as a reference (the closer an algorithm is to the Uniform Sampling curve, the better).

2.5.3. Computation Time Comparisons

Table 2.1 provides runtimes for the DS, scSampler-sp1, scSampler-sp16, DPP, and WSE [2] algorithms. We implemented the DS algorithm in Python 3 and tested the runtimes of the above-mentioned algorithms on the Quest High Performance Computing Cluster of Northwestern University. The runtimes were for the normal examples (runtime is relatively invariant to the underlying distribution) with varying data set sizes. The reported runtime statistics in Table 2.1 are in seconds and were averaged over 100 independent replications, although there was not much replicate-to-replicate variability. 'NA' indicates that a subsampling algorithm took longer than 1 hour to run (for each replicate), in which case the experiment was terminated and the approach was considered to be computationally prohibitive for that case.

From Table 2.1, DS and scSampler-sp16 are far more efficient than other methods, often multiple orders of magnitude faster. One should keep in mind, however, that the uniformity performance of scSampler-sp16 was overall far inferior to DS in Figures 2.6 and 2.7. Notice also that for $n \geq 1000$, the runtime of DS remains relatively unchanged with varying n, while the runtime of scSampler-sp16 (and of scSampler-sp1 and DPP) grows super linearly with n increasing. Consequently, for the largest n in Table 2.1, DS becomes faster than scSampler-sp16. Moreover, for larger scale examples with even larger n, we can expect DS to be substantially faster than even scSampler-sp16 (in addition to

[2]We wrapped the published WSE code (Yuksel 2016) in Python and tested its runtime directly from Python.

having substantially better performance). Notice also that DPP and WSE were prohibitively slow for even the smallest size examples in Table 2.1, and were too slow to be included in the table for the larger size examples.

		DS	scSampler-sp1	scSampler-sp16	DPP	WSE
$q = 10$ $N = 10^5$	$n = 1,000$	1.17×10^1	1.29×10^1	8.34×10^{-1}	NA	3.09×10^3
	$n = 8,000$	1.16×10^1	9.79×10^1	6.34×10^0	NA	3.05×10^3
	$n = 20,000$	1.14×10^1	2.33×10^2	1.52×10^1	NA	2.92×10^3
$q = 10$ $N = 10^6$	$n = 1,000$	1.17×10^2	1.59×10^2	7.84×10^0	NA	NA
	$n = 8,000$	1.17×10^2	1.29×10^3	6.15×10^1	NA	NA
	$n = 20,000$	1.15×10^2	3.01×10^3	1.60×10^2	NA	NA
$q = 20$ $N = 10^6$	$n = 1,000$	1.40×10^2	2.31×10^2	1.02×10^1	NA	NA
	$n = 8,000$	1.40×10^2	1.72×10^3	7.69×10^1	NA	NA
	$n = 20,000$	1.40×10^2	$3.42 \times 10^3+$	1.97×10^2	NA	NA
$q = 40$ $N = 10^6$	$n = 1,000$	1.73×10^2	3.36×10^2	1.32×10^1	NA	NA
	$n = 8,000$	1.73×10^2	$2.64 \times 10^3+$	1.02×10^2	NA	NA
	$n = 20,000$	1.74×10^2	NA	2.54×10^2	NA	NA

Table 2.1. Runtime comparisons of subsampling algorithms. The runtime is reported in seconds and averaged over 100 replications. 'NA' means the subsampling algorithm took longer than 1 hour to finish for all tested replications. + on the right side of a number indicates the real statistic was no smaller than the reported statistic (replicates that took longer than 1 hour were terminated and excluded from the average, in which case the reported numbers are somewhat lower than actual runtimes).

The DS algorithm mainly consists of three parts: estimating the density of D up-front from scratch (line 12 in Procedure 1), updating the density periodically (line 23 in

Procedure 1) and selecting subsample points (line 15 and line 26 in Procedure 1). For convenience, we will refer these three parts to 'Initial Density Estimation', 'Density Updating' and 'Subsample Selection' respectively. We observe that for all tested examples, Initial Density Estimation and Density Updating cost around 95% of the total computation time, and the computational costs of 'Initial Density Estimation' and 'Density Updating' are roughly comparable (results are not shown here for brevity). Considering that the density estimation accounts for most of the computational cost, for large N the DS expense could be reduced by randomly sampling some fraction of D to use for the density estimation. This is somewhat similar to the computational strategy used by scSampler-sp16 to reduce computational expense relative to scSampler-sp1. The former randomly partitions D into 16 subsets and applies scSampler-sp1 to each subset, and then combines the 16 subsamples.

2.6. Conclusions

This chapter presents a novel diversity subsampling algorithm, the DS algorithm, that selects an i.i.d. uniform subsample from a data set over the effective support of the empirical data distribution, to the largest extent possible. The asymptotic performances of the DS algorithm were proven, and its advantages over existing diversity subsampling algorithms were demonstrated numerically: Overall, the DS algorithm selects subsamples more similar to a true i.i.d uniform sample than existing algorithms do with much lower computational cost. We have also presented a generalized version of the DS algorithm to select subsamples following any desired target density (or non-negative target function in

general) and proven its asymptotic accuracy. All methods proposed in the chapter are implemented in the FADS (Shang, Apley, and Mehrotra 2022b) Python package.

Bibliography

Asmussen, Søren and Peter W Glynn (2007). *Stochastic simulation: algorithms and analysis*. Vol. 57. Springer Science & Business Media.

Ba, S., W. R. Myers, and W. A. Brenneman (2015). "Optimal sliced latin hypercube designs". In: *Technometrics* 57.4, pp. 479–487.

Bentley, J. L. (1975). "Multidimensional binary search trees used for associative searching". In: *Communications of the ACM* 18.9, pp. 509–517.

Billingsley, P. (1995). *Probability and Measure*. Wiley Series in Probability and Statistics. Wiley. ISBN: 9780471007104. URL: `https://books.google.com/books?id=z39jQgAACAAJ`.

Bıyık, Erdem (2019). *dpp sampler.py*. URL: `https://github.com/Stanford-ILIAD/DPP-Batch-Active-Learning/blob/master/classification_synthetic/dpp_sampler.py`. (accessed: 03.26.2020).

Bıyık, Erdem et al. (2019). "Batch active learning using determinantal point processes". In: *arXiv preprint arXiv:1906.07975*.

Casella, George, Christian P Robert, and Martin T Wells (2004). "Generalized accept-reject sampling schemes". In: *Lecture Notes-Monograph Series*, pp. 342–347.

Chen, Daijun and Shifeng Xiong (2017). "Flexible nested Latin hypercube designs for computer experiments". In: *Journal of Quality Technology* 49.4, pp. 337–353.

Cook, Robert L (1986). "Stochastic sampling in computer graphics". In: *ACM Transactions on Graphics (TOG)* 5.1, pp. 51–72.

Currin, Carla et al. (1988). *A Bayesian approach to the design and analysis of computer experiments*. Tech. rep. ORNL Oak Ridge National Laboratory (US).

Dempster, Arthur P, Nan M Laird, and Donald B Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1, pp. 1–22.

Duan, W. et al. (2017). "Sliced Full Factorial-Based Latin Hypercube Designs as a Framework for a Batch Sequential Design Algorithm". In: *Technometrics* 59.1, pp. 11–22.

Garcia, Vincent, Eric Debreuve, and Michel Barlaud (2008). "Fast k nearest neighbor search using GPU". In: *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*. IEEE, pp. 1–6.

Gut, Allan (2013). *Probability: a Graduate Course*. Vol. 75. Springer.

Han, Insu and Jennifer Gillenwater (2020). "MAP Inference for Customized Determinantal Point Processes via Maximum Inner Product Search". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 2797–2807.

Harville, David A (1997). *Matrix algebra from a statistician's perspective*. Vol. 1. Springer.

Haussmann, Elmar et al. (2020). "Scalable Active Learning for Object Detection". In: *arXiv preprint arXiv:2004.04699*.

Hickernell, J. H., J. L. Bentley, and R. A. Finkel (1977). "An algorithm for finding best matches in logarithmic expected time". In: *ACM Transactions of Mathematical Software* 3.3, pp. 209–226.

Jin, R., W. Chen, and A. Sudjianto (2005). "An efficient algorithm for constructing optimal design of computer experiments". In: *Journal of Statistical Planning and Inference* 134, pp. 268–287.

Jobson, J Dave (2012). *Applied multivariate data analysis: volume II: Categorical and Multivariate Methods*. Springer Science & Business Media.

Johnson, M. E., L. M. Moore, and D. Ylvisaker (1990). "Minimax and maximin distance designs". In: *Journal of Statistical Planning and Inference* 26, pp. 131–148.

Jones, D. R., M. Schonlau, and W. J. Welch (1998). "Efficient global optimization of expensive black-box functions". In: *Journal of Global Optimization* 13, pp. 455–492.

Joseph, R. and Y. Hung (2008). "Orthogonal-maximin Latin Hypercube Designs". In: *Statistica Sinica* 34, pp. 171–186.

Joseph, V Roshan (2016). "Space-filling designs for computer experiments: A review". In: *Quality Engineering* 28.1, pp. 28–35.

Joseph, V Roshan, Evren Gul, and Shan Ba (2015). "Maximum projection designs for computer experiments". In: *Biometrika* 102.2, pp. 371–380.

Joseph, V. R. et al. (2015). "Sequential exploration of complex surfaces using minimum energy designs". In: *Technometrics* 57.1, pp. 64–74.

Kennard, Ronald W and Larry A Stone (1969). "Computer aided design of experiments". In: *Technometrics* 11.1, pp. 137–148.

Ko, Chun-Wa, Jon Lee, and Maurice Queyranne (1995). "An exact algorithm for maximum entropy sampling". In: *Operations Research* 43.4, pp. 684–691.

Kong, Xiangshun, Mingyao Ai, and Kwok Leung Tsui (2017). "Design for sequential follow-up experiments in computer emulations". In: *Technometrics*, pp. 1–9.

Loeppky, Jason L, Leslie M Moore, and Brian J Williams (2010). "Batch sequential designs for computer experiments". In: *Journal of Statistical Planning and Inference* 140.6, pp. 1452–1464.

Mack, YP and Murray Rosenblatt (1979). "Multivariate k-nearest neighbor density estimates". In: *Journal of Multivariate Analysis* 9.1, pp. 1–15.

MacQueen, James et al. (1967). "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA, pp. 281–297.

Mak, Simon and V Roshan Joseph (2018). "Support points". In: *The Annals of Statistics* 46.6A, pp. 2562–2592.

McCool, Michael and Eugene Fiume (1992). "Hierarchical Poisson disk sampling distributions". In: *Proceedings of the conference on Graphics interface*. Vol. 92, pp. 94–105.

Montgomery, Douglas C (2017). *Design and analysis of experiments*. John wiley & sons.

Morris, M. D. and T. J. Mitchell (1995). "Exploratory designs for computational experiments". In: *Journal of Statistical Planning and Inference* 43, pp. 381–402.

Mount, David M. and Sunil Arya (2010). *ANN: A Library for Approximate Nearest Neighbor Searching*. URL: `https://www.cs.umd.edu/~mount/ANN/` (visited on 08/01/2016).

Parzen, Emanuel (1962). "On estimation of a probability density function and mode". In: *The annals of mathematical statistics* 33.3, pp. 1065–1076.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Puzyn, Tomasz et al. (2011). "Investigating the influence of data splitting on the predictive ability of QSAR/QSPR models". In: *Structural Chemistry* 22.4, pp. 795–804.

Qian, P. (2009). "Nested Latin hypercube designs". In: *Biometrika* 96.4, pp. 957–970.

— (2012). "Sliced Latin Hypercube Designs". In: *Journal of the American statistical Association* 107, pp. 393–399.

Qian, Peter ZG and CF Jeff Wu (2009). "Sliced space-filling designs". In: *Biometrika* 96.4, pp. 945–956.

Ren, Pengzhen et al. (2021). "A survey of deep active learning". In: *ACM Computing Surveys (CSUR)* 54.9, pp. 1–40.

Rennen, G. et al. (2010). "Nested maximin Latin hypercube designs". In: *Structural and Multidisciplinary Optimization* 41.3, pp. 371–395.

Reynolds, Douglas A (2009). "Gaussian mixture models." In: *Encyclopedia of biometrics* 741.659-663.

Rosenblatt, Murray (1956). "Remarks on some nonparametric estimates of a density function". In: *The Annals of Mathematical Statistics*, pp. 832–837.

Sacks, J. et al. (1989). "Design and analysis of computer experiments". In: *Statistical Science* 4.4, pp. 409–435.

Santner, Thomas J, Brian J Williams, and William I Notz (2013). *The design and analysis of computer experiments*. Springer Science & Business Media.

Shang, Boyang and Daniel W Apley (2020). *FSSF: Generate Fully-Sequential Space-Filling Designs Inside a Unit Hypercube*. `https://CRAN.R-project.org/package=FSSF` [Accessed: 06.13.2022].

— (2021). "Fully-sequential space-filling design algorithms for computer experiments". In: *Journal of Quality Technology* 53.2, pp. 173–196.

Shang, Boyang, Daniel W Apley, and Sanjay Mehrotra (2022a). "Diversity Subsampling: Custom Subsampling from Data Sets". In: *TBD* 00.0, pp. 100–200.

— (2022b). *Fast Diversity Subsampling from a Data Set*. `https://pypi.org/project/FADS/` [Accessed: 06.02.2022].

Sherman, Jack and Winifred J Morrison (1950). "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix". In: *The Annals of Mathematical Statistics* 21.1, pp. 124–127.

Silveira, Amanda Lemes and Paulo Jorge Sanches Barbeira (2022). "A fast and low-cost approach for the discrimination of commercial aged cachaças using synchronous fluorescence spectroscopy and multivariate classification". In: *Journal of the Science of Food and Agriculture*.

Slutsky, Evgeny (1925). "Uber stochastische asymptoten und grenzwerte". In: *Metron* 5.3, pp. 3–89.

Sobol, J. M. (1967). "On the distribution of points in a cube and the approximate evaluation of integrals". In: *USSR Computational Mathematics and Mathematical Physics* 7.4, pp. 86–112.

Song, Dongyuan et al. (2022a). *scSampler*. `https://github.com/SONGDONGYUAN1994/scsampler` [Accessed: 02.09.2022].

— (Apr. 2022b). "scSampler: fast diversity-preserving subsampling of large-scale single-cell transcriptomic data". In: *Bioinformatics*. btac271. ISSN: 1367-4803. DOI: `10.1093/bioinformatics/btac271`. eprint: `https://academic.oup.com/bioinformatics/advance-article-pdf/doi/10.1093/bioinformatics/btac271/43385173/btac271.pdf`. URL: `https://doi.org/10.1093/bioinformatics/btac271`.

Székely, Gábor J and Maria L Rizzo (2004). "Testing for equal distributions in high dimension". In: *InterStat* 5, pp. 1–6.

Székely, GJ (2003). "E-Statistics: The energy of statistical samples". In: *Bowling Green State University, Department of Mathematics and Statistics Technical Report* 3.05, pp. 1–18.

Tan, Matthias HY (2013). "Minimax designs for finite design regions". In: *Technometrics* 55.3, pp. 346–358.

Tang, Boxin (1998). "Selecting Latin hypercubes using correlation criteria". In: *Statistica Sinica*, pp. 965–977.

Terrell, George R and David W Scott (1992). "Variable kernel density estimation". In: *The Annals of Statistics*, pp. 1236–1265.

Van der Vaart, Aad W (2000). *Asymptotic statistics*. Vol. 3. Cambridge university press.

Wang, Zi et al. (2018). "Active model learning and diverse action sampling for task and motion planning". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 4107–4114.

Wu, Dongrui (2018). "Pool-based sequential active learning for regression". In: *IEEE transactions on neural networks and learning systems* 30.5, pp. 1348–1359.

Yang, JY, Min-Qian Liu, and Dennis KJ Lin (2014). "Construction of nested orthogonal Latin hypercube designs". In: *Statistica Sinica* 24.1.

Yu, Hwanjo and Sungchul Kim (2010). "Passive sampling for regression". In: *2010 IEEE International Conference on Data Mining*. IEEE, pp. 1151–1156.

Yuksel, Cem (2015). “Sample elimination for generating Poisson disk sample sets”. In: *Computer Graphics Forum*. Vol. 34. 2. Wiley Online Library, pp. 25–32.

— (2016). *cySampleElim.h*. URL: `https://github.com/cemyuksel/cyCodeBase/blob/master/cySampleElim.h`. (accessed: 09.17.2020).

APPENDIX A

Additional Materials for Chapter 1

A.1. Detailed Pseudo-code for FSSF algorithm

Algorithm 3 Details is the detailed version of Algorithm 3.

Algorithm 3 Details Details of FSSF-b Algorithm

Input: $n_{\max}$, N, q

if $q < 8$ **then**
    $K \leftarrow 20$
else
    $K \leftarrow 40$
end if
Generate candidate set $\mathcal{C} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$ as a Sobol sequence
Set indicator of whether or not to (re)run the KNN algorithm *startover* $= 1$
for $n = N, N-1, \cdots, 4, 3$ **do**
    if *startover* $= 1$ **then**
        $Re \leftarrow \{1, 2, \cdots, N\} \setminus \{i_N, i_{N-1}, \cdots, i_{n+1}\}$
        $\hat{n} \leftarrow |Re|$
        if $\hat{n} \leq K$ **then**
            $K \leftarrow \hat{n} - 1$
        end if
        $Rn \leftarrow K\mathbf{1}_{\hat{n}}$
        $\boldsymbol{D} \leftarrow \mathbf{0}_{\hat{n} \times K}$
        $\boldsymbol{J} \leftarrow \mathbf{0}_{\hat{n} \times K}$
        for $i = 1, 2, \cdots, \hat{n}$ **do**
            for $j = 1, 2, \cdots, K$ **do**
                Find n_j^i such that $\boldsymbol{x}_{Re[n_j^i]}$ is the jth nearest neighbor of $\boldsymbol{x}_{Re[i]}$ among $\{\boldsymbol{x}_{Re[k]}\}_{\substack{k=1 \\ k \neq i}}^{\hat{n}}$

▷ Continued on the next page.

$J_{ij} \leftarrow n_j^i$

$D_{ij} \leftarrow d(\boldsymbol{x}_{Re[i]}, \boldsymbol{x}_{Re[n_j^i]})$

end for

end for

Create an empty linked list $D \leftarrow \{\}$

Let Id be an array of empty linked list with size $\hat{n}$

for $i = 1, 2, \cdots, \hat{n}$ **do**

for $j = 1, 2, \cdots, K$ **do**

Insert i after the first element of linked list $Id[J_{ij}]$

end for

end for

Construct a balanced binary tree Tr using array $(D_{11}, D_{21}, \cdots, D_{\hat{n}1})$, with each child node storing both the distance and the corresponding index, i.e. each child node has index D_{i1} and value i, for some $i \in \{1, 2, \cdots, \hat{n}\}$. The keys are sorted in non-decreasing order and do not have to be unique.

$startover = 0$

end if

Find the smallest index in the balanced binary tree Tr, store the corresponding value as $\tilde{i}_n^1$.

$\tilde{i}_n^2 \leftarrow J_{\tilde{i}_n^1 1}$

if $J_{\tilde{i}_n^2 1} \neq \tilde{i}_n^1$ **then**

$d_1 \leftarrow D_{\tilde{i}_n^2 1}$

▷ Continued on the next page.

39: **else**

40: $d_1 \leftarrow D_{\tilde{i}_n^2 2}$

41: **end if**

42: $d_2 \leftarrow D_{\tilde{i}_n^1 2}$

43: Set $\tilde{i}_n = \tilde{i}_n^1, DeleteKey = D_{\tilde{i}_n^1 1}$ if $d_1 \geq d_2$, and $\tilde{i}_n = \tilde{i}_n^2, DeleteKey = D_{\tilde{i}_n^2 1}$ otherwise.

44: Insert $\tilde{i}_n$ before the first element of linked list D

45: $i_n \leftarrow Re[\tilde{i}_n]$

46: Find and Remove the corresponding child node with index $DeleteKey$ from the balanced binary tree Tr.

47: $Rn[\tilde{i}_n] \leftarrow 0$

48: **for** $u \in Id[\tilde{i}_n]$ **do**

49: $Rn[u] = Rn[u] - 1$

50: **if** $Rn[u] = 1$ **then**

51: set $startover = 1$

52: break from the innermost loop.

53: **end if**

54: Find $j \in \{1, 2, \cdots, K\}$ such that $J_{uj} = \tilde{i}_n$

55: **if** $j = 1$ **then**

56: In the balanced binary tree Tr, try to find child nodes whose indices equal D_{u1}, and whose value equals u

57: **if** such locations do not exit **then**

58: Go to the next iteration for the innermost loop ▷ Continued on the next page.

59: **end if**

60: remove this child node from the balanced binary tree Tr

61: insert the new child node with index D_{u2} and value u into the balanced binary tree Tr.

62: **end if**

63: Shift J_{uj} to the last element of $J_{u,}$

64: Shift D_{uj} to the last element of $D_{u,}$

65: **end for**

66: **end for**

67: $\{i_1, i_2\} \leftarrow R \setminus \{Re[D[1]], \cdots, Re[D[|D|]]\}$

Output: Ordered FSSF design sequence $S_{n_{\max}} = \left(\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \cdots, \boldsymbol{x}_{i_{n_{\max}}}\right)$

A.2. Mathematical Link between IMSE of GP and the Space-filling Design

The Minimax Design proposed by Johnson, L. M. Moore, and Ylvisaker (1990) is defined as $\arg\min_{S_n \subset \Omega} h_{\max}(S_n)$, where $h_{\max}(S_n)$ is defined in Equation (1.2). Johnson, L. M. Moore, and Ylvisaker (1990) showed a version mathematical link between IMSE of GP and the Minimax Design. We here provide a slightly different derivation to show the link between IMSE of GP and the Minimax Design under the unbiasedness assumption (see Sacks et al. (1989) for the derivation of Equation (1.9), which is derived under unbiasedness assumption on the predictor).

Proposition 1. *The solution of problem* $\arg\min_{S_n \subset \Omega} IMSE(S_n)$ *can be approximated by the solution of the Minimax problem* $\arg\min_{S_n \subset \Omega} h_{max}(S_n)$.

PROOF. In this proof, we use the same notation and same assumptions as in Chapter 1. Some proof ideas are borrowed from Johnson, L. M. Moore, and Ylvisaker (1990).

We observe in Equation (1.6), function $R(\boldsymbol{x}, \boldsymbol{x}')$ only depends on $d_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}')$. So for convenience, we define

$$R(d(\boldsymbol{x}, \boldsymbol{x}')) = e^{-d(\boldsymbol{x}, \boldsymbol{x}')}. \tag{A.1}$$

Note that in Equation (A.1), $d(\boldsymbol{x}, \boldsymbol{x}')$ may depend on $\boldsymbol{\theta}$, we drop the subscript $\boldsymbol{\theta}$ for simplicity. In addition, we define

$$d(\boldsymbol{x}, S_n) = \min_{i=1,\cdots,n} d(\boldsymbol{x}, \boldsymbol{x}_i), \tag{A.2}$$

where $S_n = \{\boldsymbol{x}_i\}_{i=1}^n$.

It is easy to observe that $R(d(\boldsymbol{x}, \boldsymbol{x}'))$ in Equation (A.1) is a decreasing function of $d(\boldsymbol{x}, \boldsymbol{x}')$ the range of which is $(0, 1]$. $\forall k \in \mathbb{Z}_+$, $R^k(d(\boldsymbol{x}, \boldsymbol{x}'))$ has the similar properties with $R(d(\boldsymbol{x}, \boldsymbol{x}'))$. We show the link between IMSE of GP with minimax design using $R^k(d(\boldsymbol{x}, \boldsymbol{x}'))$ as the correlation coefficient when $k \to +\infty$ and $n \to +\infty$.

As in Equation (1.10),

$$\begin{aligned} \mathrm{IMSE}(S_n) &= \int_\Omega \mathrm{MSE}(\boldsymbol{x}, S_n)\, \mathrm{d}\boldsymbol{x} \\ &= \sigma^2|\Omega| - \sigma^2 \int_\Omega \begin{pmatrix} 1 & \boldsymbol{r}^T(\boldsymbol{x}) \end{pmatrix} \begin{pmatrix} 0 & \mathbf{1}_{1\times n} \\ \mathbf{1}_{n\times 1} & \boldsymbol{R} \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ \boldsymbol{r}(\boldsymbol{x}) \end{pmatrix} \mathrm{d}\boldsymbol{x}. \end{aligned} \tag{A.3}$$

Let $\lambda_{\min}, \lambda_{\max}$ be the smallest and largest eigenvalue of $\begin{pmatrix} 0 & \mathbf{1}_{1\times n} \\ \mathbf{1}_{n\times 1} & \boldsymbol{R} \end{pmatrix}^{-1}$. Then by Rayleigh Principle,

$$
\begin{aligned}
&\lambda_{\min}(1+\sum_{i=1}^{n} R^{2k}(d(\boldsymbol{x}_i,\boldsymbol{x}))) \\
&\leq \begin{pmatrix} 1 & \boldsymbol{r}^T(\boldsymbol{x}) \end{pmatrix} \begin{pmatrix} 0 & \mathbf{1}_{1\times n} \\ \mathbf{1}_{n\times 1} & \boldsymbol{R} \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ \boldsymbol{r}(\boldsymbol{x}) \end{pmatrix} \qquad \text{(A.4)} \\
&\leq \lambda_{\max}(1+\sum_{i=1}^{n} R^{2k}(d(\boldsymbol{x}_i,\boldsymbol{x}))).
\end{aligned}
$$

Now we consider the case when $k \to +\infty$. Clearly, $R^k(d(\boldsymbol{x},\boldsymbol{x}')) \to 1$ if $d(\boldsymbol{x},\boldsymbol{x}') = 0$ and $R^k(d(\boldsymbol{x},\boldsymbol{x}')) \to 0$ if $d(\boldsymbol{x},\boldsymbol{x}') > 0$. Thus, $\boldsymbol{R} \to \boldsymbol{I}_{n\times n}$, i.e., the identity matrix in $\mathbb{R}^{n\times n}$, as $k \to +\infty$.

Using the properties of Schur complement, the eigenvalues of matrix $\begin{pmatrix} 0 & \mathbf{1}_{1\times n} \\ \mathbf{1}_{n\times 1} & \boldsymbol{R} \end{pmatrix}^{-1}$ can be easily found as $\lambda_1 = \cdots \lambda_{n-1} = 1$, $\lambda_n = \frac{2}{1-\sqrt{1+4n}} < 0$ and $\lambda_{n+1} = \frac{2}{1+\sqrt{1+4n}} \in (0,1)$. So as $k \to +\infty$, $\lambda_{\min} \to \frac{2}{1-\sqrt{1+4n}} < 0$ and $\lambda_{\max} \to 1$. Taking $n \to +\infty$, we also have $\lambda_{\min} \to 0-$.

Therefore as $k \to +\infty$ and $n \to +\infty$, $\begin{pmatrix} 0 & \mathbf{1}_{1\times n} \\ \mathbf{1}_{n\times 1} & \boldsymbol{R} \end{pmatrix}^{-1}$ is asymptotically positive semi-definite. Hence by Equation (A.3),

$$\arg\min_{S_n\subset\Omega} \mathrm{IMSE}(S_n) = \arg\max_{S_n\subset\Omega} \int_\Omega \begin{pmatrix} 1 & \boldsymbol{r}^T(\boldsymbol{x}) \end{pmatrix} \begin{pmatrix} 0 & \mathbf{1}_{1\times n} \\ \mathbf{1}_{n\times 1} & \boldsymbol{R} \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ \boldsymbol{r}(\boldsymbol{x}) \end{pmatrix} \mathrm{d}\boldsymbol{x}. \tag{A.5}$$

In addition, as $k \to +\infty$,

$$\begin{aligned} &\sum_{i=1}^n R^{2k}(d(\boldsymbol{x}_i,\boldsymbol{x}))) \\ =&\sum_{i=1}^n \frac{R^{2k}(d(\boldsymbol{x}_i,\boldsymbol{x})))}{R^{2k}(d(S_n,\boldsymbol{x})))} R^{2k}(d(S_n,\boldsymbol{x}))) \\ =&R^{2k}(d(S_n,\boldsymbol{x}))) \sum_{i=1}^n [\mathbf{1}_{d(\boldsymbol{x}_i,\boldsymbol{x})=d(S_n,\boldsymbol{x})} + o(1)\mathbf{1}_{d(\boldsymbol{x}_i,\boldsymbol{x})>d(S_n,\boldsymbol{x})}]. \end{aligned} \tag{A.6}$$

Thus, as $k \to +\infty$ and $n \to +\infty$, Equation (A.4) becomes

$$\begin{aligned} 0 \le &\begin{pmatrix} 1 & \boldsymbol{r}^T(\boldsymbol{x}) \end{pmatrix} \begin{pmatrix} 0 & \mathbf{1}_{1\times n} \\ \mathbf{1}_{n\times 1} & \boldsymbol{R} \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ \boldsymbol{r}(\boldsymbol{x}) \end{pmatrix} \\ \le &1 + R^{2k}(d(S_n,\boldsymbol{x}))) \sum_{i=1}^n [\mathbf{1}_{d(\boldsymbol{x}_i,\boldsymbol{x})d(S_n,\boldsymbol{x})} + o(1)\mathbf{1}_{d(\boldsymbol{x}_i,\boldsymbol{x})>d(S_n,\boldsymbol{x})}]). \end{aligned} \tag{A.7}$$

Considering Equation (A.5), we have $\arg\min_{S_n\subset\Omega} \mathrm{IMSE}(S_n)$ can be approximated by

$$\arg\max_{S_n\subset\Omega} \int_{\boldsymbol{x}\in\Omega} R^{2k}(d(S_n,\boldsymbol{x}))) \sum_{i=1}^n \mathbf{1}_{d(\boldsymbol{x}_i,\boldsymbol{x})d(S_n,\boldsymbol{x})}\, \mathrm{d}\boldsymbol{x}. \tag{A.8}$$

By Equation (A.2), we have $\forall \boldsymbol{x} \in \Omega$, $\sum_{i=1}^{n} \mathbf{1}_{d(\boldsymbol{x}_i,\boldsymbol{x})d(S_n,\boldsymbol{x})} \geq 1$. And since $R()$ is a decreasing function, we have

$$\int_{\boldsymbol{x}\in\Omega} R^{2k}(d(S_n,\boldsymbol{x}))) \sum_{i=1}^{n} \mathbf{1}_{d(\boldsymbol{x}_i,\boldsymbol{x})d(S_n,\boldsymbol{x})} \, \mathrm{d}\boldsymbol{x} \geq |\Omega| R^{2k}(\max_{\boldsymbol{x}\in\Omega} d(S_n,\boldsymbol{x})) \tag{A.9}$$

So we can maximize $R^{2k}(\max_{\boldsymbol{x}\in\Omega} d(S_n,\boldsymbol{x}))$ to approximate Equation (A.8), which is equivalent to solve the Minimax problem, i.e.,

$$\arg\min_{S_n\subset\Omega} \max_{\boldsymbol{x}\in\Omega} d(S_n,\boldsymbol{x}) = \arg\min_{S_n\subset\Omega} h_{\max}(S_n). \tag{A.10}$$

□

APPENDIX B

Additional Materials for Chapter 2

B.1. Density Estimation for the DS algorithm

The DS algorithm (Procedure 1) can be used with any density estimation method, whether or not the estimated density integrates to 1, for instance, fixed or variable bandwidth KDE (Rosenblatt 1956; Parzen 1962; Terrell and Scott 1992), KNN density estimation (Mack and Rosenblatt 1979), and Gaussian Mixture Model (GMM) density estimation (Reynolds 2009). Taking both density estimation accuracy and computational efficiency into consideration, we found that GMM density estimation with diagonal covariance matrices was overall the most effective for use in the DS algorithm. In all of our examples, we estimate the parameters of the GMM model using the popular Expectation-Maximization (EM) technique (Dempster, Laird, and Rubin 1977; Reynolds 2009).

The GMM approach models the density as

$$f(\boldsymbol{x}) = \sum_{k=1}^{M} c_k f_k(\boldsymbol{x}|\boldsymbol{\theta}_k), \tag{B.1}$$

where $c_k \geq 0,\ \forall k = 1, \cdots, M$ and $\sum_{k=1}^{M} c_k = 1$. Here M is the number of components for the GMM model, and $f_k(\boldsymbol{x}|\boldsymbol{\theta}_k)$ is the gaussian p.d.f with mean $\boldsymbol{\mu}_k$ and a diagonal covariance matrix $\text{Diag}(\sigma_{k1}^2, \cdots, \sigma_{kq}^2)$. We denote $\boldsymbol{\theta}_k = (c_k, \boldsymbol{\mu}_k, \sigma_{k1}, \cdots, \sigma_{kq})$ and write

$$f_k(\boldsymbol{x}|\boldsymbol{\theta}_k) = \prod_{j=1}^{q} \frac{1}{\sqrt{2\pi}\sigma_{kj}} e^{-\frac{\left(x_j - \mu_{kj}\right)^2}{2\sigma_{kj}^2}}. \tag{B.2}$$

Given initial guesses for $\boldsymbol{\theta}_1^{(0)}, \cdots, \boldsymbol{\theta}_M^{(0)}$, we apply the EM algorithm a fixed number (denoted *niter*) of iterations to estimate the parameters $\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_M$. The algorithm is outlined as Procedure 2. See, for example Reynolds 2009, for detailed discussions and derivations for GMM density estimation.

Procedure 2 GMM density estimation using EM for the DS algorithm (Procedure 1)

Input: N, q, $D = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$, $niter$, M, $\boldsymbol{\theta}_1^{(0)}, \cdots, \boldsymbol{\theta}_M^{(0)}$

1: **for** $t \in \{1, \cdots, niter\}$ **do**
2: **for** $k \in \{1, \cdots, M\}$ **do**
3: **for** $i \in \{1, \cdots, N\}$ **do**
4: $p_{ik}^{(t-1)} \leftarrow \frac{c_k^{(t-1)} f_k(\boldsymbol{x}_i|\boldsymbol{\theta}_k^{(t-1)})}{\sum_{j=1}^{M} c_j^{(t-1)} f_j(\boldsymbol{x}_i|\boldsymbol{\theta}_j^{(t-1)})}$
5: **end for**
6: **for** $j \in \{1, \cdots, q\}$ **do**
7: $\mu_{kj}^{(t)} \leftarrow \frac{\sum_{i=1}^{k} p_{ik}^{(t-1)} x_{ij}}{\sum_{i=1}^{k} p_{ik}^{(t-1)}}$
8: $\sigma_{kj}^{(t)} \leftarrow \sqrt{\frac{\sum_{i=1}^{k} p_{ik}^{(t-1)} x_{ij}^2}{\sum_{i=1}^{k} p_{ik}^{(t-1)}} - \left(\mu_{kj}^{(t)}\right)^2}$
9: **end for**
10: $c_k^{(t)} \leftarrow \frac{1}{N} \sum_{i=1}^{N} p_{ik}^{(t-1)}$
11: **end for**
12: **end for**
13: **for** $k = 1, \cdots, M$ **do**
14: **for** $i = 1, \cdots, N$ **do**
15: Compute $f_k(\boldsymbol{x}_i|\boldsymbol{\theta}_k^{(niter)})$ according to Equation (B.2)
16: **end for**
17: **end for**
18: **for** $i = 1, \cdots, N$ **do**
19: $\hat{f}_{\text{GMM}}(\boldsymbol{x}_i) \leftarrow \sum_{k=1}^{M} c_k^{(niter)} f_k(\boldsymbol{x}_i|\boldsymbol{\theta}_k^{(niter)})$
20: **end for**

Output: Density estimation at data points in D, $\{\hat{f}_{\text{GMM}}(\boldsymbol{x}_i), \text{ for } i = 1, 2, \cdots, N\}$

For all of our examples, we chose $niter = 10$ and $M = 32$ for the 'Initial Density Estimation' (line 12 of Procedure 1) of DS and used an existing implementation by Pedregosa et al. 2011 for the GMM density estimation. We set all GMM density estimation parameters as their default values except for $niter$ and M. The 'Density Updating' of the DS algorithm (line 23 of Procedure 1) is methodologically similar to the 'Initial Density Estimation', except that for better efficiency, we use the estimated values of $\{\boldsymbol{\theta}_k\}_{k=1}^{M}$ from the previous update as the initial guesses in the current updating procedure, and we set

$niter = 1$. Note that for the 'Density Updating' procedure, we fit the GMM model to only the data points in D that have not been previously selected.

B.2. Estimating the Deviation Point of the DS Algorithm

In this section, we take the DS subsample as an example to illustrate the point that for finite N, any subsampling algorithm will inevitably deviate from uniform sampling after a certain subsample size n, due to points in sparser regions being depleted from D. For a subsample of size n selected by the DS algorithm, denoted by $\mathcal{S}_n = \{\boldsymbol{x}_{j_1}, \cdots, \boldsymbol{x}_{j_n}\}$, with $\{j_1, \cdots, j_n\} \subset \{1, \cdots, N\}$, we use $n_\Omega = \sum_{k=1}^{n} \mathbf{1}_\Omega(\boldsymbol{x}_{j_k})$ to denote the number of selected points in $\mathcal{S}_n$ lying inside Ω. Consider a small region $\mathrm{d}V$ in the lowest density region inside Ω and denote the lowest density function value of $f(\boldsymbol{x})$ inside Ω by $f_{\min,\,\Omega}$. Then among data points in D, there are, on average, approximately $N f_{\min,\,\Omega}|\mathrm{d}V|$ data points inside region $\mathrm{d}V$. Similarly, if $\mathcal{S}_n$ is i.i.d uniformly distributed over Ω, then $\mathcal{S}_n$ contains on average approximately $n_\Omega \frac{|\mathrm{d}V|}{|\Omega|}$ selected points inside region $\mathrm{d}V$. Therefore the deviation from uniformity over Ω of a subsample theoretically must begin no later than when $n_\Omega \frac{|\mathrm{d}V|}{|\Omega|} = N f_{\min,\,\Omega}|\mathrm{d}V|$, i.e. when $n_\Omega = N f_{\min,\,\Omega}|\Omega| \stackrel{\mathsf{def}}{=} n_\Omega^b$. We refer to the DS subsample size, n^b, at which there are n_Ω^b points inside Ω as the 'deviation point' of the DS algorithm.

Figure B.1 shows an example with estimated $n^b \approx 590$ (denoted by the vertical black dash line) using the normal example with $q = 2$. The experimental setup is the same as in Section 2.5.2. From Figure B.1 we see that the average $e(\mathcal{S}_n, \mathcal{U})$ for the DS algorithm starts to deviate from the results of Uniform Sampling at around n^b, as the above theoretical arguments predict.

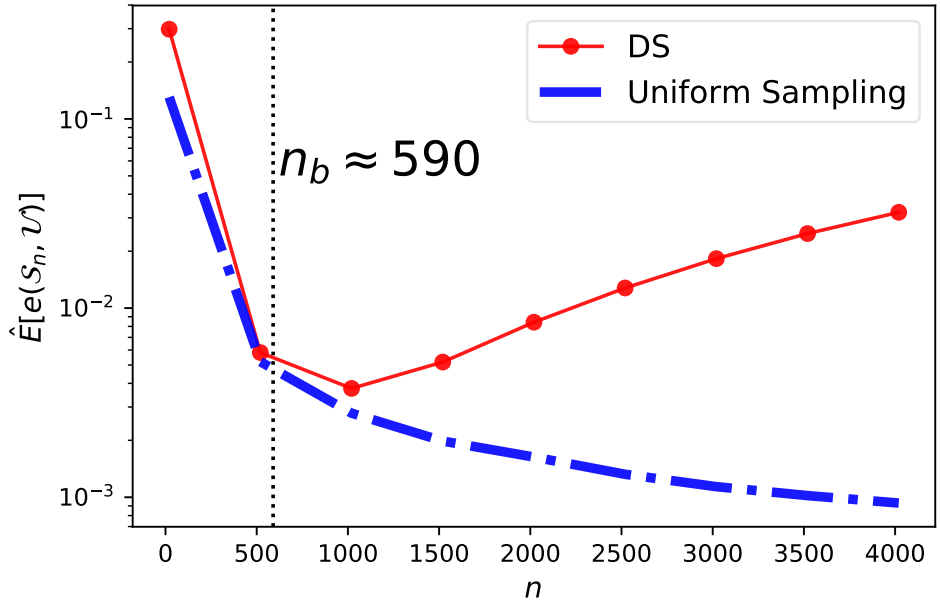
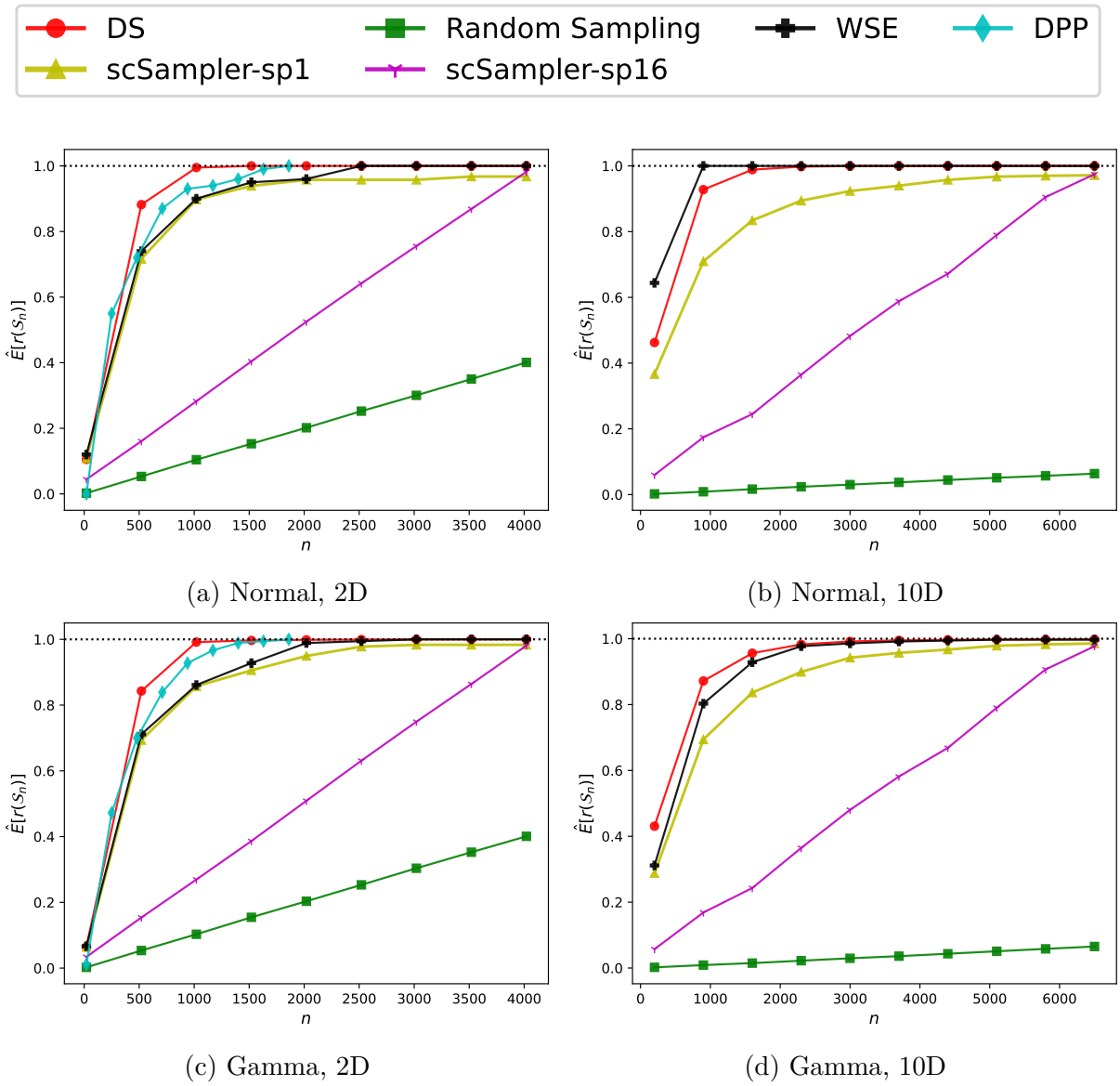


Figure B.1. Estimation of n_b (shown by the vertical dotted line) for the DS algorithm. Uniform Sampling serves as a reference (the closer an algorithm is to the Uniform Sampling curve, the better).

B.3. Numerical Performance Results for the Low Density Region Sampling Ratio Criterion

Figure B.2 shows the average low-density region sampling ratio (Equation (2.35)) for the six subsampling algorithms, namely random sampling, DS, WSE, scSampler-sp1, scSampler-sp16, and DPP. The Uniform Sampling reference is irrelevant under this criterion, because it only generates samples within Ω. In 8 out of 10 examples (Figures B.2a and B.2c to B.2i), the average low-density region sampling ratio of the DS algorithm converges to 1 faster than (Figures B.2a, B.2c, B.2d and B.2f to B.2h) or comparably to (Figures B.2e and B.2i) the other subsampling algorithms, which demonstrates its effectiveness for selecting data points in the low-density regions when n is small. For the other

two examples (Figures B.2b and B.2j) performances of DS are not far behind the best performing method.

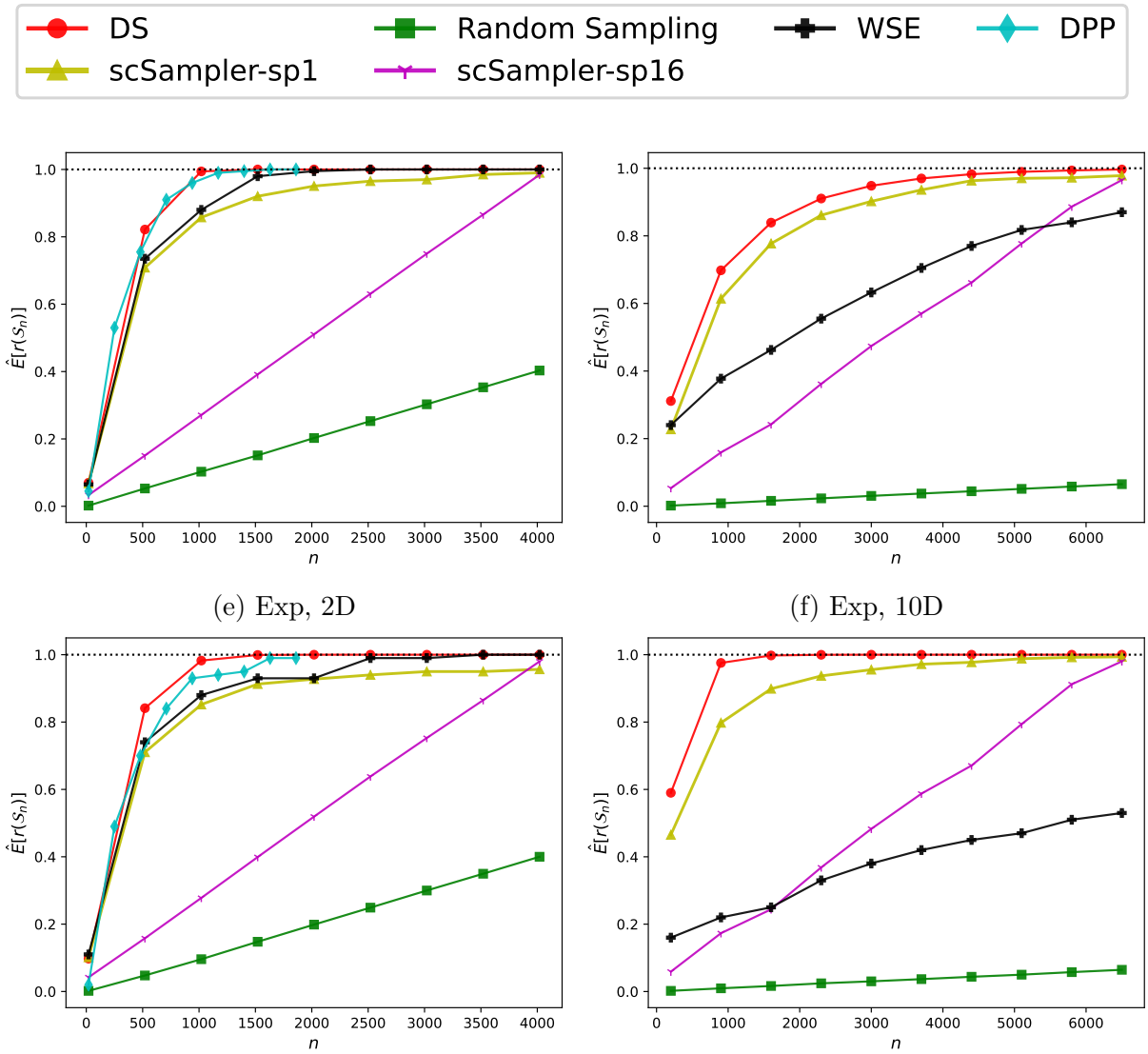


(a) Normal, 2D

(b) Normal, 10D

(c) Gamma, 2D

(d) Gamma, 10D

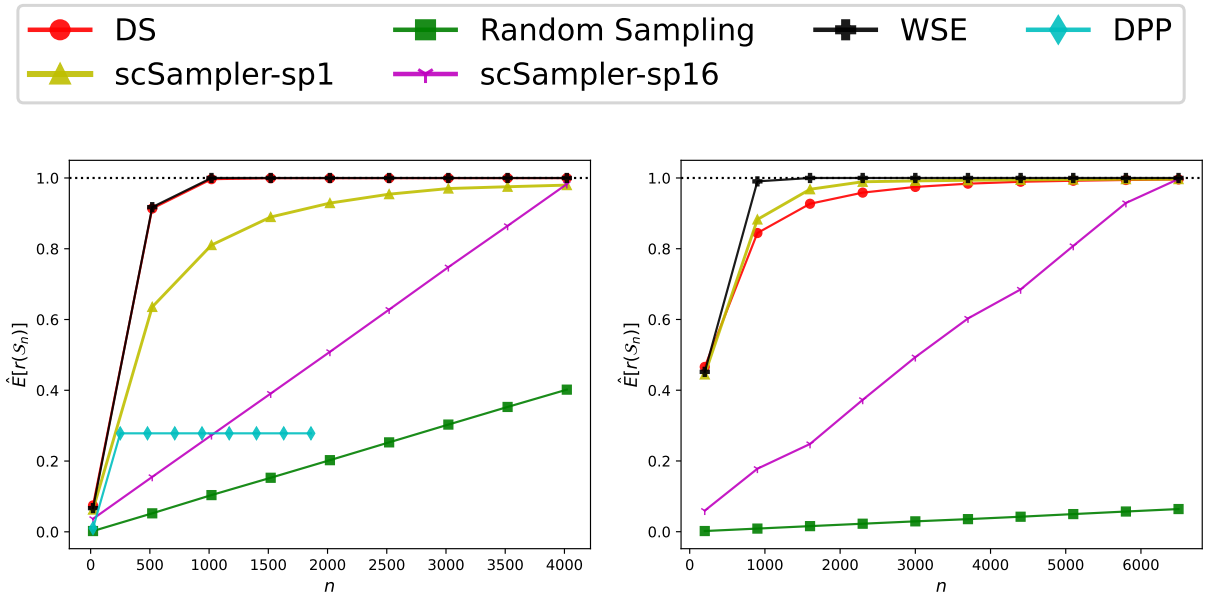


(e) Exp, 2D

(f) Exp, 10D

(g) MGM, 2D

(h) MGM, 10D

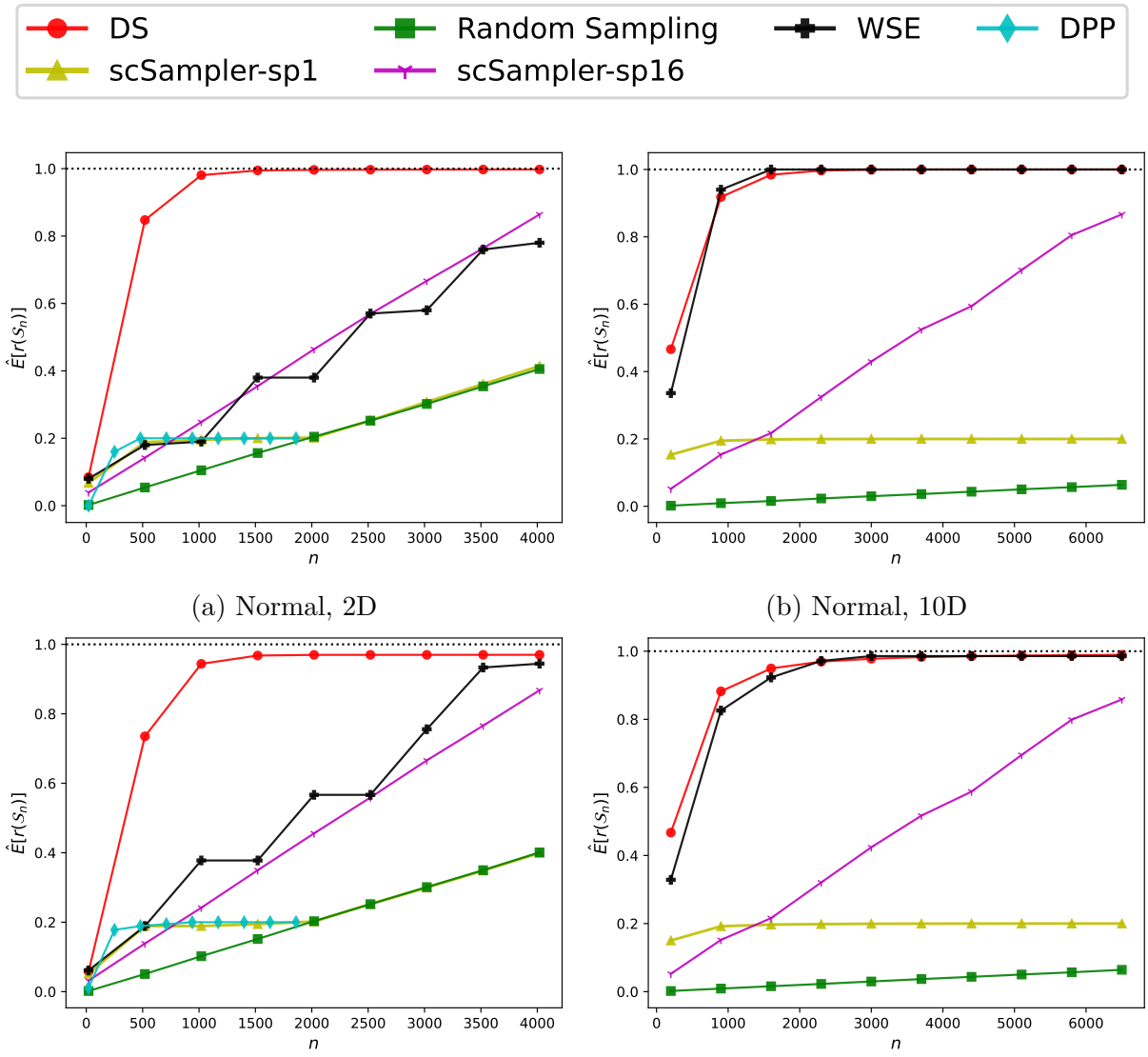


(i) Geometric, 2D

(j) Geometric, 10D

Figure B.2. Averaged $r(\mathcal{S}_n)$ (Equation (2.35)), as a function of n, for subsamples selected by DS, random sampling from D ('Random Sampling'), scSampler-sp1, scSampler-sp16, WSE, and DPP.

Analogous results for the replicated D examples are shown in Figure B.3, from which we see that the results of DS are not adversely affected much when the data sets have many replications, and DS remains among the best performing algorithms for all examples. For some examples (e.g., Figures B.3a, B.3c, B.3e, B.3g and B.3h), DS performs substantially better than the next best performing method.

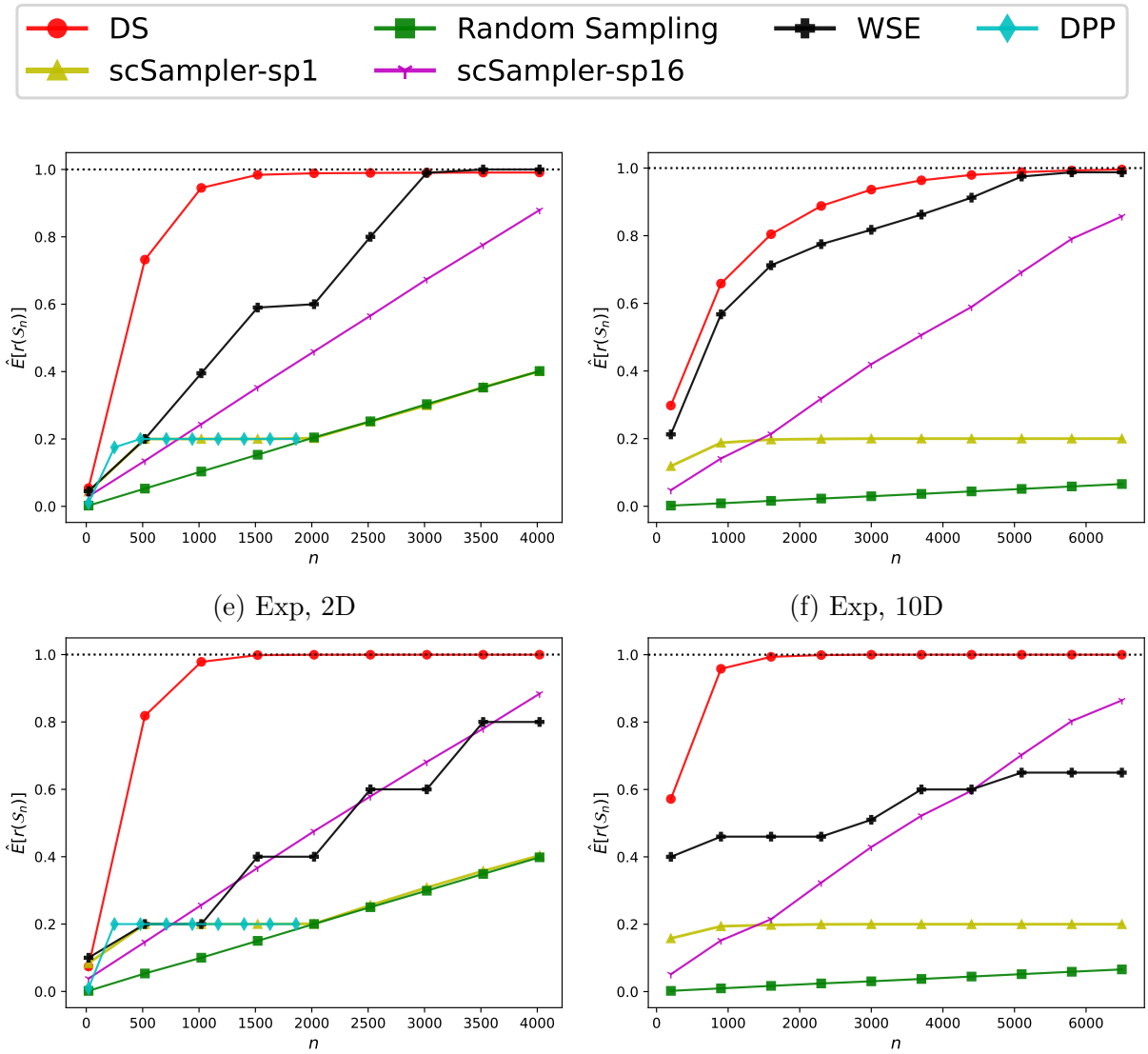


(a) Normal, 2D

(b) Normal, 10D

(c) Gamma, 2D

(d) Gamma, 10D



(e) Exp, 2D

(f) Exp, 10D

(g) MGM, 2D

(h) MGM, 10D

Figure B.3. Averaged $r(\mathcal{S}_n)$ (Equation (2.35)), as a function of n, using replicated data, for subsamples selected by DS, random sampling from D ('Random Sampling'), scSampler-sp1, scSampler-sp16, WSE, and DPP.