Cassandra Lisitza

Robustness of Maximin Space-Filling Designs

Abstract

In this report, we first have a review of the maximin space-filling design method that is often applied and discussed in the literature (for example, Müller, 2007). Then we discuss the robustness of the maximin space-filling design against model misspecification via numerical simulation. For this purpose, we generated spatial data sets on a $n \times n$ grid, and design points were selected from the n^2 locations. The predictions at the unsampled locations were made based on the observations at these design points. Then the mean of the squared prediction errors were estimated as a measure of the robustness of the designs against possible model misspecification. Surprisingly, according to the simulation results, we find that the maximin space-filling designs may be robust against possible model misspecification in the sense that the mean of the squared prediction error did not increase a notable amount when the model is misspecified. Although the results were obtained based on simple models, this result is very inspiring. It will guide further numerical and theoretical studies that will be done as future work.

Key Words: Maximin space-filling designs, model misspecification, robustness, spatial data generation

1 Introduction

Space-filling designs are designs suitable for computational problems, which allow us to simulate data and perform statistical tests on data as opposed to using real data and performing repeated experimental testing. It imposes less of a socioeconomic burden, as the data collection process can be expensive. Further, the data collection process can be very time consuming, and hence computer simulations are an efficient means of reaching the same end goal. In the context of this problem, we simulate data and perform statistical tests to determine whether the maximin space-filling design is robust against model misspecification.

Space-filling designs extend selected design points approximately evenly throughout the given area. Space-filling designs can be difficult to establish in practice, and thus, space-filling designs are generally useful in computer simulations where we can ensure we have a nearly purely deterministic system, as we will exemplify. There are several space-filling designs: U-optimal, maximin, minimax and coffee-house to name but a few [2]. We will focus on the maximin space-filling design. As the name suggests, maximin designs are designs that push design points to the boundaries, contrary to minimax designs in literature such as Pronzato (2017). That is, the design maximizes the smallest distance between any two points in the region X using the standard distance metric. We partition a given region into an $n \times n$ square such that we have n^2 cells.

1.1 Is maximin space-filling design robust against model misspecification?

We will evaluate the robustness of the maximin space-filling design against model misspecification via numerical methods. For that purpose, we will consider two scenarios. The first scenario is that the model proposed by the researcher is the correct model. The second scenario is that the researcher proposed a model but it is only an approximation of the true model. Under both scenarios, spatially correlated data sets from the correct models will be generated on the $n \times n$ grid. We assume that the researcher will use the proposed model to fit the data. However, as we know, the proposed model in the second scenario is incorrect. In this case, the prediction must be biased. We therefore simulate the mean of the squared prediction errors (MSPE), and compare it with the value of MSPE obtained under the first scenario. If the values of MSPE in both cases are close, we then conclude that based on the simulation results, the maximin space-filling design is robust against the model misspecification in the second scenario.

Under the first scenario, we assume the true model is

$$Y = \beta_0 + \mathbf{x}\boldsymbol{\beta} + \varepsilon(x_h, x_v) \tag{1.1}$$

where β_0 and $\boldsymbol{\beta} = (\beta_1, ..., \beta_p)$ are model parameters, $\mathbf{x} = (x_1, ..., x_p) \in \mathbb{R}^p$ are independent variables, and (x_h, x_v) is the coordinate of a grid point where x_h is the horizontal coordinate and x_v is the vertical coordinate. Here $\{\varepsilon(x_h, x_v)\}$ is a Gaussian process with $E(\varepsilon(x_h, x_v)) = 0$, $Var(\varepsilon(x_h, x_v)) = \sigma^2$ and a covariance function

$$E[\varepsilon(\mathbf{x})\varepsilon(\mathbf{x}')] = c(\mathbf{x}, \mathbf{x}'). \tag{1.2}$$

We will choose the values of the model parameters and the covariance function in the simulation. Suppose that the researcher fit the following model after collecting data

$$Y = \theta_0 + \mathbf{x}\boldsymbol{\theta} + \varepsilon(x_h, x_v) \tag{1.3}$$

Assume that the researcher can get the correct correlation function (1.2). Therefore, the researcher used the generalized least squares (GLS) estimators to estimate the regression parameters and made predictions based on the GLS estimates. We will simulate M data sets from model (1.1), each with size n^2, and then simulate the mean squared prediction error (MSPE)

$$E[Y - \hat{Y}_{GLS}]^2$$

where $\hat{Y}_{GLS}$ is the predictor based on the GLS estimates.

Next, we consider the second scenario. Suppose that the researcher proposed model (1.3) to fit the data. However, in this case, the true model (which we will generate data from) is

$$Y = \theta_0 + \mathbf{x}\boldsymbol{\theta} + \psi(\mathbf{x}) + \varepsilon(x_h, x_v) \tag{1.4}$$

where $\psi(\mathbf{x})$ is a small deviation between the true model and the proposed model (1.3).

Assume that the researcher didn't realize the model misspecification and still used model (1.3) to fit the collected data, but they can obtain the correct correlation function (1.2). Therefore, the researcher still used the generalized least squares estimators to estimate the regression parameters and made predictions based on the GLS estimates. We simulate M data sets from model (1.4), each with size n^2, and then simulate the mean squared prediction error $E[Y-\hat{Y}_{GLS}]^2$ where $\hat{Y}_{GLS}$ is the predictor based on the GLS estimates of model (1.3).

In Section 2, we will briefly introduce how to generate spatially correlated data by applying the Markov chain Monte Carlo algorithm, and then we will simulate the mean squared prediction errors for different values of n in Section 3.

2 Generating Spatial Data Sets

Markov chain Monte Carlo (MCMC) is known as one of the "ten most important algorithms" of the 20th century [4]. The goal of MCMC is to simulate a random variable X whose distribution is the target distribution π. Two

algorithms are often used: Metropolis-Hastings algorithm and Gibbs sampler. We will apply the Gibbs sampler to generate spatial data sets.

We are going to simulate spatial datasets based on models such as Model (1.1) or Model (1.4) on a grid. The datasets include the independent variables x_h and x_v as described above, $\mathbf{x} = (x_1, ..., x_p)$, and the dependent variable (Y). Since the deterministic part of an observation can be easily calculated once the model parameters are determined, we only need to simulate the random error variables, which are assumed to be correlated Gaussian. To apply the Gibbs sampler to generate the Gaussian process $\mathbf{x} = (x_h, x_v)$, we need to find the conditional distribution (i.e., the conditional mean and conditional variance since the process is Gaussian) of $\varepsilon(\mathbf{x})$ given all the other $\varepsilon(\mathbf{x}')$ for $\mathbf{x} \neq \mathbf{x}'$.

We denote the covariance matrix as Σ and its jth row as $\Sigma_{j.}$. Moreover, by deleting the jth row and the jth column of Σ we obtain a submatrix denoted as $\Sigma_{-j,-j}$. Let $\boldsymbol{\varepsilon} = (\varepsilon(\mathbf{x}_1), ..., \varepsilon(\mathbf{x}_{100}))$. By deleting the jth entry of the vector $\boldsymbol{\varepsilon}$ we obtain $\boldsymbol{\varepsilon}_{-j}$. Then the conditional mean and conditional variance of $\varepsilon(\mathbf{x}_j)$ (conditional on $\boldsymbol{\varepsilon}_{-j}$) are

$$\Sigma_{j.}\Sigma_{-j,-j}^{-1}\boldsymbol{\varepsilon}_{-j}$$

and

$$1 - \Sigma_{j.}\Sigma_{-j,-j}^{-1}\Sigma_{j.}^{T}$$

separately. We then find the conditional means and variances for all the entries in $\boldsymbol{\varepsilon}$.

The R code used to generate the datasets was modified based on Chyzh [1]. We assume that $Var(\varepsilon(x_h, x_v)) = 1$ and the covariance function is

$$E[\varepsilon(\mathbf{x})\varepsilon(\mathbf{x}')] = c(\mathbf{x}, \mathbf{x}')$$

for any two points $\mathbf{x} = (x_h, x_v)$ and $\mathbf{x}' = (x_h', x_v')$.

3 Simulate the Mean of Squared Prediction Errors

As a special example of model (1.1), we consider the following model

$$Y = 1 + 2x_1 + 1.5x_2 + \varepsilon(x_h, x_v) \tag{3.1}$$

.

For the specified model, $\beta_0 = 1, \beta_1 = 2, \beta_2 = 1.5$ We generate the points on a grid and calculate the distance between any two given points and define the covariance function as

$$Cov(y_1, y_2) = \frac{1}{1+d} \tag{3.2}$$

where d is the spatial distance between y_1 and y_2. In general, the distance between two points x and y is $d(x,y) = ((x_1-x_2)^2+(y_1-y_2)^2)^{1/4}$, where x_1, x_2 are on the horizontal coordinate x_h and y_1, y_2 are on the vertical coordinate x_v. By applying the simulation method described in Section 2, $M = 500$ datasets were generated. The next step is to obtain a maximin design on the 10×10 grid. An R package *Maximin* was developed by (Sun) 2021. We applied this package, and the R code is available in the Appendix.

With the maximin space-filling design obtained, a sample with sample size $n = 10$ will be chosen from each generated dataset. We then found the generalized linear estimates for the model parameters of Model (1.3) based on the samples, and then made predictions $\hat{Y}_{GLS}$ at the grid points that were not chosen. We then compute $(y - \hat{y}_{GLS})^2$ for all 500 samples and the average of the squared differences. For this simulation, an average MSPE of 83.48097 was obtained.

This process was repeated for grids with sizes $n^2 = 25$ and $n^2 = 400$ which yielded average MSPEs 1.94135 and 464.8311, respectively. The results are summarized in the table below:

Grid Size	Average MSPE
$n^2 = 25$	1.94135
$n^2 = 100$	83.48097
$n^2 = 400$	464.8311

Table 1: The average MSPE for Different Sample Sizes for Model 5

As we can see, the average MSPE increases as the sample size increases, which is what we would expect. Now we want to move on to our second scenario, our misspecified model (1.4), to determine whether the space-filling design is robust against model misspecification.

As a special case of model (1.4), we consider the model below to be the true model

$$Y = 1 + 2x_1 + 1.5x_2 + 0.01x_1^2 + 0.003x_2^2 + \varepsilon(x_h, x_v) \tag{3.3}$$

Like before, we start by considering a grid size of $n^2 = 100$ and use the R package *Maximin* to construct a space-filling design, which is exactly the same as we constructed for model (3.1) with the same grid size.

We then generated datasets based on model (3.3). Once we have all the points generated on a grid, we calculate the distance between any two points and define our covariance function as before in (3.2). We then apply the Gibbs sampler as before to find the conditional distribution to generate the Gaussian

process. Similar to what was done for model (3.1), 500 samples were selected from the generated datasets according to the maximin space-filling design. GLS estimates of the model parameters were obtained and predictions were made at the grid points that were not selected.

We then computed $(y - \hat{y}_{GLS})^2$ the same as before for all of the datasets and then computed the average. For this simulation, an average MSPE of 83.32013 was calculated.

In a similar fashion, we repeated this process for grid sizes $n^2 = 25$ and $n^2 = 400$ to obtain average MSPEs 1.91884 and 462.1736, respectively.

Grid Size	Average MSPE
$n^2 = 25$	1.91884
$n^2 = 100$	83.32013
$n^2 = 400$	462.1736

Table 2: The average MSPE for Different Sample Sizes for Model 6

Similar to what we saw in our first scenario, we see a steady increase in the average MSPE as the sample size increases. Let us now compare the results from our two scenarios.

Grid Size	Average MSPE Model 5	Average MSPE Model 6	Difference
$n^2 = 25$	1.94135	1.91884	0.02251
$n^2 = 100$	83.48097	83.32013	0.16084
$n^2 = 400$	464.8311	462.1736	2.6575

Table 3: Comparing the Results from Scenario 1 and Scenario 2

For a grid size of $n^2 = 25$, we have a positive difference of 0.02251 indicating that the MSPE values are similar. There is no increase of the MSPE when the model is misspecified. For $n = 100$, we have a difference of 0.16084 which again indicates that the MSPE doesn't increase for the misspecified model. Finally, when $n = 400$ was considered, we see very similar MSPE values, which again shows that slight model misspecification does not have an effect on the MSPE.

4 Summary

In this work, we generated spatial data under two scenarios. The first scenario is based on Model (1.1) and the second scenario based on Model (1.4). We then simulated the mean of squared prediction errors under both scenarios and varied the grid sizes. We saw that the second scenario which considered the

misspecified model yielded similar MSPE values compared to that of the first scenario. Although this simulation is simple, it shows a very inspiring result. That is, the maximin space-filling design might be robust against model misspecification in the sense that the MSPE values do not change remarkably even when the model is misspecified. To further verify this result, more numerical simulations will be needed, and we will try to prove the result theoretically as future work.

References

[1] Chyzh, O., (2018, May 22). *Estimate an LGM.* GitHub. https://github.com/ochyzh/LSGM-demos/blob/master/estimate_lsgm.rmd

[2] Muller, G. (2007). *Collecting spatial data: optimum design of experiments for random fields.* Springer Science & Business Media.

[3] Pronzato, L. (2017) Minimax and maximin space-filling designs: some properties and methods for con- struction. *Journal de la Societe Française de Statistique, Societe Française de Statistique et Societe Mathematique de France,* 158 (1), 7-36. hal-01496712

[4] Richey, M. (2010). The Evolution of Markov Chain Monte Carlo Methods. *The American Mathematical Monthly,* 117(5), 383–413. https://doi.org/10.4169/000298910x485923

[5] Sun, F., Gramacy, R.B. (2021) Space-Filling Design under Maximin Distance. https://cran.r-project.org/web/packages/maximin/maximin.pdf

[6] Yildirim, I. (2012). Bayesian Inference: Gibbs Sampling. *University of Rochester Department of Brain and Cognitive Sciences*, 1-6. http://www.mit.edu/ ilkery/papers/GibbsSampling.pdf

Appendix

```
### Space-Filling Designs
sink(".output.txt",append=TRUE)

# We will need the following packages
#install.packages('tgp')
#install.packages('lhs',repos='/private/var/folders/xv/tyk6ldpd70bgwkr3k5yy8nq40000gn/T/RtmpVbCoNV/downloaded_packages'

#install.packages('maximin',repos='/private/var/folders/xv/tyk6ldpd70bgwkr3k5yy8nq40000gn/T/RtmpVbCoNV/downloaded_packa

#update.packages("kableExtra")
#install.packages("kableExtra",dependencies = TRUE,repos='https://cloud.r-project.org/',type='binary')

library(tgp)
library(lhs)
library(maximin)
library(hms)
library(kableExtra)


# 3.3 Simulate the mean of squared prediction errors

##### n=10

# Step 1
# We use the following code to construct a space-filling design.
nn <- 10
p <- 2
Xorig <- randomLHS(5, p)
x1 <- seq(0, 1, length.out=nn)
Xcand <- expand.grid(replicate(p, x1, simplify=FALSE))
names(Xcand) <- paste0("x", 1:2)
T <- nrow(Xcand)
Xsparse <- maximin.cand(n=nn, Xcand=Xcand, Tmax=T, Xorig=Xorig,
                        init=NULL, verb=FALSE, tempfile=NULL)
index=Xsparse$inds
space_design=Xcand[index,]
space_design # We make observations at these locations
maxmd <- as.numeric(format(round(max(na.omit(Xsparse$mis)), 5), nsmall=5))

# Step 2
set.seed(400)
n=100
fixedpars<-c(1,2,1.5)

# Create the grid
x.h<-seq(0,1,length.out = 10)
x.v<-seq(0,1,length.out = 10)
coord.points=expand.grid(x.h, x.v)

coord<-expand.grid(x1=x.h, y1=x.v, x2=x.h, y2=x.v)

dist<-((coord$x1-coord$x2)^2+(coord$y1-coord$y2)^2)^(1/4)

covar=function(d)
{
  coy1y2=1/(1+d) ## Covariance between two variables y1, y2 only depends on d, the spatial distance between y1 and y2
}
covmatrix=matrix(covar(dist), nrow=100, ncol=100)
eigenv=eigen(covmatrix) ## Positive semidefinite matrix
```