Active Learning for Optimizing Drug Doses and Timing of Administration

by

Xiaotian Zhang

(Under the Direction of Abhyuday Mandal and Qian Xiao)

Abstract

In modern pharmaceutical studies, researchers have found that both the doses and timing of administration of drug components in combinatorial drugs have significant impact on treatment efficacy. However, these studies often have constraints, such as the minimum separation time and maximum number of drug components. To find the best drug combinations, simultaneous optimization for both the doses and timing of administration is required, where one-shot experimental designs are often inefficient. In this work, we propose a novel active learning procedure along with a new Gaussian process model to efficiently identify the optimal configurations within only a few experimental trials. The superiority of the proposed method is illustrated via different simulation studies and real case study.

Index words: [Drug dose, Timing of administration, Active learning, Gaussian process model]

Active Learning for Optimizing Drug Doses and Timing of Administration

by

Xiaotian Zhang

B.S., Beijing Normal University, China, 2020

A [Thesis] Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the
Degree.

[Master] of [Science]

Athens, Georgia

2023

Active Learning for Optimizing Drug Doses and Timing of Administration

by

Xiaotian Zhang

Major Professor:	Abhyuday Mandal
Committee:	Qian Xiao
	Ray Bai

Electronic Version Approved:

Ron Walcott
Dean of the Graduate School
The University of Georgia
May 2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Modern pharmaceutical studies have found that both the dose and timing of drugs have significant impact on treatment effect. Classic pharmacokinetic models and pharmacodynamic models depict the impact of dose and time on overall efficacy by affecting drug concentration (Holford & Sheiner, 1981). Dose adjustment and timing administration have been practically used to get an optimal treatment result. For example, dosage and timing optimization of chemotherapy are necessary to ensure efficacy meanwhile reducing the risk of side-effects for hemodialysis (Janus et al., 2010).

The influence of each drug is not the only factor. In addition, the interaction effect is another decisive factor. There are many theories focusing on drug combination analysis. For example, Berenbaum, 1977 gives a detailed review of many previous methods for determining the nature of drug interactions. More and more mathematical models and theoretical methods for multiple drug combination experimental designs and real data analysis also have been widely applied in biomedical science including median-effect equation (Chou, 2006), network modeling (Iadevaia et al., 2010), and evolutionary modeling (Mumenthaler et al., 2011). For complex situation of multiple drugs, which has many potential interactions, like six antiviral drugs of seven dose levels each, it is hard to go through all the possible combinations. In this case, fractional factorial design (Jaynes et al., 2013) and orthogonal array composite design (OACD) (Luna et al., 2022) can be used to help identify optimal drug combinations, which we will discuss in the next chapter.

The sequence of drugs also significantly influences the therapy effect a lot, which can be explained by interactions between drug combinations (Shah &

Schwartz, 2000). In a drug combination experiment of three anticancer drugs: bortezomib, camptothecin, and doxorubicin (Ding et al., 2015), the result shows that not only the drug dose and ratio but also the drug sequence and the intervals between each drug administration are important to ensure the desired efficacy. Table 1.1 reports another order-of-addition experiment data of five drugs from two blocks, which shows different sequence influences the drug inhibition effect (Xiao & Xu, 2021).

Table 1.1: Drug sequence and corresponding response

block	sequence	response	block	sequence	response
1	DBACE	4.93	2	DBCEA	5.53
1	BACDE	13.63	2	BADEC	7.72
1	DABEC	15.57	2	ABDCE	10.96
1	DCEAB	18.47	2	BDCAE	12.09
1	EDABC	19.5	2	DAECB	13.84
1	ABEDC	20.23	2	ADEBC	16.25
1	BDECA	21.47	2	AECDB	16.37
1	AEBCD	21.59	2	DCABE	17.97
1	ACDBE	23.55	2	ECDAB	19.71
1	ADCEB	23.61	2	EDBCA	20.35

The situations can be more complicated in real life. There might be a large pool of drug components but only several of them can be used in the limited treatment time period. Suppose only one drug can be used every day and we plan to complete the treatment in a week, we can only choose seven drugs. In this case, there are more possible combinations of different components and action time constraint. In such a situation, the researcher needs to screen the drug combinations to determine which set produces the best result, while also calculating the impact of the drug sequence, dose and time setting on the response.

Although here are already some previous models dealing with important component screening problem, regression model for order-of-addition experiment, or optimization with constraints, it is difficult to reach a good result in theses complex situations. In this paper, we solve these generalized problems using a Gaussian process model with mapping functions and an active learning method to do screening design and optimization. The model effect is shown in different simulation cases.

1.2 Literature Review

Models for order-of-addition experiment:

There are two kinds of linear models applied for order-of-addition problems: pair-wise ordering (PWO) model (Mee, 2020; Van Nostrand, 1995; Voelkel, 2019), component-position (CP) model (Yang et al., 2021). The PWO model considers the relative positions of each pair of components while CP model cares about the absolute positions. Based on these two models, Piepho and Williams, 2021 proposed response surface (RS) regression model, which considers both relative and absolute positions of the components using polynomials, and uses component position numbers as predictor variables.

Compared with some linear and non-linear models, kriging model gives more accurate predictions using less number of runs (Xiao et al., 2019). Based on this good property, recent researches have applied kriging models for both quantitative and qualitative factors (Deng et al., 2017; Zhang et al., 2020). For order-of-addition inputs, universal kriging (UK) model and mapping-based universal krigging (MUK) model (Xiao & Xu, 2021) have stronger prediction power and are more robust with less parameters and less runs.

Another mapping-based additive Gaussian process (MaGP) model (Xiao, Wang, et al., 2022) was proposed to fit and predict quantitative-sequence (QS) factors. Based on the mapping parameter we choose, the mapping function can have different dimensions, of which both 2-dimensional MaGP and full-dimensional MaGP models have smaller RMSEs than generalized PWO and CP models for all cases. The author has shown the outcome of some practical problems using proposed MaGP model, for example, traveling salesman problem. There are also some other methods to deal with optimization over sequence order (Rego et al., 2011) (Kumar & Jadon, 2014).

Models for screening:

There are some effect models for screening. Morris and Mitchell, 1995 proposed a screening method composed of individually randomized one-factor-at-a-time (OFAT) designs to determine which inputs have important effects on an output. This model can find the important variables, and can detect curvature and interactions without explicitly modeling the effects. Sobol', 1990 proposed a Monte Carlo algorithm for estimating the sensitivity of a function with respect to arbitrary groups of variables. OFAT design estimates local sensitivity of the computer model about the factors' global effects together while the Monte Carlo-based designs focus on the global sensitivity. When the number

of components is large, innovative design and active learning (IDEAL) method can achieve accurate predictions on the most efficacious drug combination, drug doses, and drug sequence with minimal experimental effort (A. Wang et al., 2020). Based on OFTA design and Sobol's method, Xiao, Joseph, et al., 2022 proposed another Maximum One-Factor-At-A-Time (MOFAT) design for screening.

There are many novel drug screening methods proposed to optimize the combinations for a maximum drug efficacy output. Iterative polynomial regression is a highly efficient model using polynomials as approximation to drug response surface (Lee et al., 2017). Model-based prediction model assessed pairwise interactions to predict efficacy of drug combinations (Zimmer et al., 2016). The microfluidic techniques increase the throughput of drug combination test platforms. It also enables automatic generation and mixing of continuous dual-drug or tri-drug concentration gradients, which facilitates the precise screening of dose combinations (Kim et al., 2012). Based on those drug screening models, B. Wang et al., 2021 illustrates a three-step workflow that could maximize the overall optimization efficiency and gives detailed discussion on more methodologies.

1.3 Motivation

We propose a novel active learning procedure, along with a new Gaussian process model. Our method can identify the optimal drug combination configurations within only a few experimental trials, making it an efficient approach to this problem. We demonstrate the superiority of our approach through simulation studies.

1.4 Outline of Thesis

The outline of this thesis is comprised of several chapters. In Chapter 2, an overview of previous models is presented, introducing linear and non-linear models for order-of-addition experiment. Chapter 3 is the core of the thesis, we propose a novel Gaussian Process model with a mapping function designed for dose and time variables, and illustrate parameter estimation method. In Chapter 4 we introduce an active learning approach for optimizing the GP model and initial design criterion based on some good design methods. In order to speed up the computation, we give calculations of analytical gradients for both parameter estimation and active learning method. Chapter 5 shows the

simulation studies of component screening with constraint, providing evidence of the effectiveness of the proposed model. Chapter 6 summarizes the current result and discussion of future study.

Chapter 2

Previous models for order-of-addition experiment

2.1 Linear models

There are two types of linear models proposed for fitting sequence variables PWO and CP model. The PWO model considers the relative positions of each pair of components while CP model cares about the absolute positions. Previous studies show that these two linear models are not contradictory but complementary, since it's hard to decide which model is better (Yang et al., 2021).

2.1.1 Pairwise ordering model

One linear model uses PWO factors $c_{i,j}$ for all pairs of component orders, $0 \leq i < j \leq m$. Each factor $c_{i,j}$ equals to 1 or -1, corresponding to whether component i is added before j or not. For each run, if i is used before j, then $c_{i,j} = 1$. Otherwise, $c_{i,j} = -1$.

The PWO model is expressed as:

$$y = \beta_0 + \sum_{i<j} \beta_{i,j} c_{i,j} + \epsilon \tag{2.1}$$

where $\epsilon \sim N(0, \sigma^2)$ is a random error. A triplets order-of-addition model including the interactions involved in three distinct components was proposed as an expanded PWO model (Mee, 2020), which is given by:

$$y = \beta_0 + \sum_{i<j} \beta_{i,j} c_{i,j} + \sum_{i<j<k} (\beta_{i,j*i,k} c_{i,j} c_{i,k} + \beta_{i,j*j,k} c_{i,j} c_{j,k}) + \epsilon \quad (2.2)$$

where $\epsilon \sim N(0, \sigma^2)$ is a random error. This model contains more parameters compared to the simple PWO model, so it gives more flexibility while also needs more run times to estimate. Table 4.1 is an example of PWO factors and interactions for 3 component full design:

Table 2.1: Full design for 3 components

Sequence	X_{12}	X_{13}	X_{23}	$X_{12}X_{13}$	$X_{12}X_{23}$	$X_{13}X_{23}$	$X_{12}X_{13}X_{23}$
123	1	1	1	1	1	1	1
132	1	1	-1	1	-1	-1	-1
213	-1	1	1	-1	-1	1	-1
231	-1	-1	1	1	-1	-1	1
312	1	-1	-1	-1	-1	1	1
321	-1	-1	-1	1	1	1	-1

Order-of-Addition Orthogonal Arrays (OofA-OAs)

Voelkel, 2019 constructed a design OofA-OAs for PWO model. An OofA-OA in n runs, m components and strength t is denoted as $OA_{n,m,t}$, if every subset of size $n \times t$ PWO factors, the frequencies are proportional to those of the full factor matrix. This design is fully efficient for PWO model.

The OofA-OA design in m components with n runs has the properties for (I_{ij}, I_{kl}):

1. If $i \neq k$, $i \neq l$, $j \neq k$, and $j \neq l$, the factors are orthogonal.

2. If $i = k$ or $j = l$, the inner product of the factors is $n/3$.

3. If $i = l$ or $j = k$, the inner product of the factors is $-n/3$.

Table 2.2 is an example of $OA(12, 4, 2)$.

Table 2.2: OofA-OA(12,4,2)

Run	01	02	03	04	I_{01}	I_{02}	I_{03}	I_{12}	I_{13}	I_{23}
1	0	1	3	2	1	1	1	1	1	-1
2	0	2	1	3	1	1	1	-1	1	1
3	0	3	1	2	1	1	1	1	-1	-1
4	1	0	2	3	-1	1	1	1	1	1
5	1	2	3	0	-1	-1	-1	1	1	1
6	1	3	2	0	-1	-1	-1	1	1	-1
7	2	0	3	1	1	-1	1	-1	-1	1
8	2	1	0	3	-1	-1	1	-1	1	1
9	2	3	0	1	1	-1	-1	-1	-1	1
10	3	0	2	1	1	1	-1	-1	-1	-1
11	3	1	0	2	-1	1	-1	1	-1	-1
12	3	2	1	0	-1	-1	-1	-1	-1	-1

2.1.2 Component position model

We can also assume that the decisive factor is each individual component's absolute position. This makes sense because the earlier administrated drug is given a longer period to exert its therapeutic benefits. This effect is observed regardless of which drugs come before or after, as each drug interacts with the body's biological systems in a unique manner.

Here is the component position (CP) model:

$$y = \mu_0 + \sum_{c=0}^{m-1} \sum_{j=1}^{m} x_c^{(j)} \tau_c^{(j)} + \epsilon \tag{2.3}$$

where μ_0 is overall mean, $x_c^{(j)} = 1$, if component c is used at position j or $x_c^{(j)} = 0$, $\tau_c^{(j)}$ is the effect of component c used at position j, and $\epsilon \sim N(0, \sigma^2)$ is a random error.

Component orthogonal array

Yang et al., 2021 also proposed a design called: component orthogonal array (COA): each run is a permutation of all m components, and for any two columns, each possible component combines equally often. Under the CP model, this new design shows a 100% D-efficiency, which is the same as full design.

An $n \times m$ matrix with entries from $(0, 1, ..., m-1)$ is a COA, denoted by $COA(n, m)$, if each row is a permutation of $(0, 1, ..., m-1)$, and for any

two columns, each level combination (i, j), with $i \neq j$ appears equally often. Here is an example of $COA(12, 4)$:

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 2 & 3 & 1 \\ 0 & 3 & 1 & 2 \\ 1 & 0 & 3 & 2 \\ 1 & 3 & 2 & 0 \\ 1 & 2 & 0 & 3 \\ 2 & 3 & 0 & 1 \\ 2 & 0 & 1 & 3 \\ 2 & 1 & 3 & 0 \\ 3 & 2 & 1 & 0 \\ 3 & 1 & 0 & 2 \\ 3 & 0 & 2 & 1 \end{bmatrix}. \tag{2.4}$$

The above array has following properties:

1. Each row is permutation of 0,1,2,3;

2. Each order appears three times in each column;

3. For every two columns, each (i, j) appears only one time, $i \neq j$.

2.2 Kriging models

There are several modified kringing models used for drug study. Kriging models are parsimonious and robust under various experimental designs and can efficiently identify drug interactions (Xiao et al., 2019). The typical kriging model is:

$$Y(x) = \mu + Z(x) + \epsilon, \tag{2.5}$$

μ is mean parameter, $Z(x)$ is a stationary Gaussian process with zero mean and covariance Φ, and $\epsilon \sim N(0, \tau^2)$.

$$\phi(x_i, x_j) = Cov(Z(x_i), Z(x_j)) = \sigma^2 \Pi_{l=1}^{k} K(h_l; \theta_l), \tag{2.6}$$

σ^2 is variance parameter, $l = 1, , , k$, and $K(h_l; \theta_l)$ is a kernel function of h_l with parameter θ_l. There are several common choices of the kernel function For Gaussian kernel function:

$$K(h_l; \theta_l) = exp(-(h_l \theta_l)^2/2), \tag{2.7}$$

and for Matern kernel function:

$$K(h_l;\theta_l) = exp(-\sqrt{2\nu}\theta_l h_l)\frac{p!}{(2p)!}\sum_{i=0}^{p}\frac{(p+i)!}{i!(p-i)!}(\sqrt{8\nu}\theta_l h_l)^{p-i}. \quad (2.8)$$

The covariance between ith and jth output can be written as:

$$Cov(y_i, y_j) = \sigma^2\Pi_{l=1}^{k}K(h_l;\theta_l) + \tau^2\delta_{ij}, \quad (2.9)$$

where $\delta_{ij} = 1$ if $i = j$ or $\delta_{ij} = 0$ otherwise.

2.2.1 UK and MUK model

UK model was proposed for modeling order-of-addition inputs with blocking (Xiao & Xu, 2021). Suppose we have k drugs and m blocks. For each input of the order-of-addition experiment, $w_i = (o_i', b_i')'$, $o_i = (o_{i,1}, ..., o_{i,k})'$ is vector of order numbers and $b_i \in \{0, ..., m-1\}$ is a block level. The UK model can be expressed as:

$$y(w_i) = \mu(b_i) + Z(o_i) + \epsilon_i, \quad (2.10)$$

where $\mu(b_i) = b^T\beta$ is the linear part of block level. β is a vector of coefficients. $Z(o_i)$ is Gaussian process with zero mean and $\epsilon_i \sim N(0, \tau^2)$. The covariance function ϕ is defined as:

$$\phi(o_i, o_j) = cov(Z(o_i), Z(o_j)) = \sigma^2\Pi_{l=1}^{k}K(h_l;\theta_l), \quad (2.11)$$

The kernel function can be chosen as Gaussian or Matern covariance function. Here, $h_l = |o_{i,l} - o_{j,l}|/k$ is the distance measure for the order of $l\text{-}th$ component between $i\text{-}th$ and $j\text{-}th$ run.

The MUK model uses more flexible mapping functions than UK model. A mapping function was introduced to calculate the order distance values. $h_l = |g_l(o_{i,l}) - g_l(o_{j,l})|$, where $l = 1, ..., k$, $i, j = 1, ..., n$, $o_{i,l}$ and $o_{j,l}$ are orders of $l\text{-}th$ element in $i\text{-}th$ input and $j\text{-}th$ input. Since the drug action time is a continuous variable, a cumulative beta function can be a useful mapping function.

$$g_l(o_{i,l}) = \frac{1}{B(\alpha_l, \delta_l)}\int_0^{f(o_{i,l})} t^{\alpha_l - 1}(1-t)^{\delta_l - 1}dt, \quad (2.12)$$

where

$$B(\alpha_l, \delta_l) = \int_0^1 t^{\alpha_l - 1}(1-t)^{\delta_l - 1}. \tag{2.13}$$

$f(o_{i,l}) = (o_{i,l} - 0.5)/k$ is used to change order levels $1, ..., k$ to 0-1 range, α_l and δ_l are parameters that can be estimated by MLE. Different mapping functions can be chosen to adjust to specific time effect, like polynomials and Sigmoid curves.

2.2.2 MaGP model for quantitative-sequence factors

MaGP model was proposed for quantitative-sequence (QS) factors, which can be used to find optimal semi-discrete solutions using only a few experiment trials (Xiao, Wang, et al., 2022). This model still keeps the flexibility and efficiency of GP model. Different from previous Kringing model, the MaGP model can deal with half-continuous and half-sequential variables by designing a mapping function for measuring the distance between order inputs.

The i^{th} input is denoted as $w_i = (x_i', o_i')'$, $x_i = (x_{i,1}, ..., x_{i,k})'$ is a vector of quantitative values and $o_i = (o_{i,1}, ..., o_{i,k})'$ is a vector of sequence numbers for k drugs. For example, if we use three drugs A, B and C in the sequence of A, C and B, the order vector should be $o = (1, 3, 2)$. The MaGP model for QS factor is:

$$Y(w) = \mu + \sum_{h=1}^{k} G_h(w) + \epsilon. \tag{2.14}$$

G_h is independent zero-mean GP with stationary covariance functions. In this case, a special distance measure of sequence inputs needs to be built to formulate covariance functions. MaGP model fix this problem by mapping the order $o_{i,h}$ to a t-dimensional vector $(\delta_{i,h}^{(1)}, ..., \delta_{i,h}^{(t)})$, which is:

$$\begin{bmatrix} 1 \\ 2 \\ \vdots \\ k \end{bmatrix} \rightarrow \begin{bmatrix} \delta_1^{(1)} & \delta_1^{(2)} & \dots & \delta_1^{(t)} \\ \delta_2^{(1)} & \delta_2^{(2)} & \dots & \delta_2^{(t)} \\ \vdots & \vdots & \vdots & \vdots \\ \delta_k^{(1)} & \delta_k^{(2)} & \dots & \delta_k^{(t)} \end{bmatrix}. \tag{2.15}$$

Here the mapping of order number is the same for all components. The model sets $\delta_l^j = 0$ for all $j \geq l$ to avoid over-parametrization. When $t = k - 1$, the mapping is called full mapping which has $k(k-1)/2$ mapping parameters and when $t = 2$, it is called 2-d mapping which has $2k - 3$ mapping parameters.

The choice of t is a trade-off between flexibility and computational cost. When the number of components is large, smaller t can reduce the computation time. The simulation results in Xiao, Wang, et al., 2022 show that both $2d$-MaGP and full-MaGP have a smaller RMSE values than PWO and CP models.

For example, for $k = 4$, the 2-d mapping is:

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ k \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 \\ \delta_2^{(1)} & 0 \\ \delta_3^{(1)} & \delta_3^{(2)} \\ \delta_4^{(1)} & \delta_4^{(2)} \end{bmatrix}, \tag{2.16}$$

while the full mapping is:

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ k \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 \\ \delta_2^{(1)} & 0 & 0 \\ \delta_3^{(1)} & \delta_3^{(2)} & 0 \\ \delta_4^{(1)} & \delta_4^{(2)} & \delta_4^{(3)} \end{bmatrix}. \tag{2.17}$$

Based on this mapping, the covariance function is defined as:

$$\phi_h(w_i, w_j|\sigma_h^2, \theta_h, \delta) = \sigma_h^2 exp\{-\theta_h(x_{i,h} - x_{j,h})^2 - \sum_{l=1}^{t}(\delta_{i,h}^{(l)} - \delta_{j,h}^{(l)})^2\}. \tag{2.18}$$

σ_h^2 is the variance parameter for h-th component, and $\delta_{i,h}$ is a t-dimensional vector corresponding to order of component h in i-th run. All the vales of δ in the mapping matrix are estimated together with other parameters. For each pair of inputs w_i and w_j, the covariance function for MaGP model is:

$$\begin{aligned} \phi(w_i, w_j) &= \sum_{h=1}^{k} \phi_h(w_i, w_j|\sigma_h^2, \theta_h, \delta) + \tau^2 1(w_i = w_j) \\ &= \sum_{h=1}^{k} \sigma_h^2 exp\{-\theta_h(x_{i,h} - x_{j,h})^2 - \sum_{l=1}^{t}(\delta_{i,h}^{(l)} - \delta_{j,h}^{(l)})^2\} + \tau^2 1(w_i = w_j). \end{aligned} \tag{2.19}$$

The full MaGP model has $1 + k(k+3)/2$ parameters while the $2d$-MaGP model has $4k - 2$ parameters. Those parameters can be estimated by MLE funciton:

$$[\sigma^2, \theta, \delta, \tau^2] = argmin\{\log|\Phi| + (y'\Phi^{-1}y) - (1'\Phi^{-1}1)^{-1}(1'\Phi^{-1}y)^2\} \tag{2.20}$$

The MLE analytical expression of μ is:

$$\hat{\mu} = (1'\Phi^{-1}1)^{-1}1'\Phi^{-1}y \tag{2.21}$$

This optimization can be solved by several method in R, like 'nloptr' package. With all the parameters estimated, the prediction mean and variance of input w_* can be expressed as:

$$\hat{Y}(w_*) = \hat{\mu} + \gamma'\Phi^{-1}(y - \hat{\mu}1) \tag{2.22}$$

$$s^2(w_*) = \phi(w_*, w_*) - \gamma'\Phi^{-1}\gamma + \frac{(1 - 1'\Phi^{-1}\gamma)^2}{1'\Phi^{-1}1} \tag{2.23}$$

where γ is covariance vector $(\phi(w_*, w_i))_{n\times 1}$, here $\phi(w_*, w_i) = \sum_{h=1}^{k} \sigma_h^2 exp\{-\theta_h(x_{i,h} - x_{*,h})^2 - \sum_{l=1}^{t}(\delta_{i,h}^{(l)} - \delta_{*,h}^{(l)})^2\}$

When the number of run N is large, it takes a long time to compute covariance function Φ. Here we can use a small trick that only updates the $(n+1)^{th}$ row and column of Φ.

$$\Phi_{n+1} = \begin{bmatrix} \Phi_n & \gamma \\ \gamma & \phi(w_{n+1}, w_{n+1}) \end{bmatrix}, \tag{2.24}$$

$$\Phi_{n+1}^{-1} = \begin{bmatrix} \Phi_n^{-1} + gg^T v & g \\ g^T & v^{-1} \end{bmatrix}, \tag{2.25}$$

where $v = \phi(w_{n+1}, w_{n+1}) - \gamma^T\Phi^{-1}\gamma$, and $g = -v^{-1}\Phi_n^{-1}\gamma$.

Chapter 3

Novel Mapping-based GP Model

In this chapter, we propose a GP model with a flexible mapping function designed for variables dose and time. The crucial point is to set a proper covariance function which can depict the similarity of data points. Different components can have different mapping functions reflecting their own effect feature. For simplicity, we adopt the same mapping function with different parameter values for each component.

3.1 Mapping function

The input data consists of two parts: x and t. We denote the i^{th} input as $w_i = (x_i', t_i')'$, $x_i = (x_{i,1}, ..., x_{i,k})'$ is a vector of dose values and $t_i = (t_{i,1}, ..., t_{i,k})'$ is a vector of action time for k drugs. Both x and t are continuous variables. If it has n runs in total, then X and T are $n \times k$ matrices. The model for output of $w = (x', t')'$ can be expressed as

$$Y(w) = \mu + \sum_{h=1}^{k} G_h(w) + \epsilon \tag{3.1}$$

where $G_h(w)$ is the impact of h^{th} component on the output, $h = 1, ..., k$, which is a Gaussian process with zero mean and stationary covariance function.

Covariance function is a crucial ingredient in GP model, since it measures the similarity between data points. The basic assumption is that those data which are close to each other are likely to have similar targets. Typical covariance functions include: squared exponential covariance function, Matern class, γ-exponential covariance function, piecewise polynomial covariance function,

and so on. Various types of mapping functions can be considered for time effect, like cumulative beta function (Xiao & Xu, 2021).

Here, we adopt a common Sigmoid function: logistic function, which is monotonic and derivable as a part of covariance function. For any two inputs w_i and w_j, the covariance function is:

$$\begin{aligned}\phi(w_i, w_j) &= \sum_{h=1}^{k} \phi_h(w_i, w_j|\sigma_h^2, \theta_h, \eta_h, \nu_h) + \tau^2 1(w_i = w_j) \\ &= \sum_{h=1}^{k} \sigma_h^2 exp\{-\theta_h(x_{i,h} - x_{j,h})^2 - (g(t_{i,h}) - g(t_{j,h}))^2\} + \tau^2 1(w_i = w_j),\end{aligned} \tag{3.2}$$

where

$$g(t_{i,h}) = \frac{1}{1 + e^{-\eta_h(t_{i,h} - \nu_h)}}. \tag{3.3}$$

$|g(t_{i,h}) - g(t_{j,h})|$ is the distance measure for time input vectors. There are $4k + 1$ parameters in this model. Take an example of $\eta_h = 1$ and $\nu_h = 10$, the plot of function $g(t_{i,h})$:

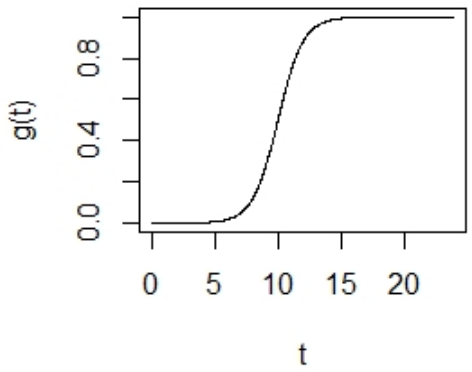

Figure 3.1: Mapping function for time input

Apparently, the distance between two time inputs of a drug is not only influenced by their difference, but also the action time of both drugs. The

values of those parameters will be estimated by MLE, which is discussed in the next section.

3.2 Model estimation

This covariance function contains parameters $\sigma^2 = (\sigma_1^2, ..., \sigma_h^2)'$, $\theta = (\theta_1, ..., \theta_h)'$, $\eta = (\eta_1, ..., \eta_h)'$, $\nu = (\nu_1, ..., \nu_h)'$ and τ^2. To find the maximized log-likelihood is equal to solve the negative log-likelihood function:

$$\log|\Phi| + (y - \mu 1)'\Phi^{-1}(y - \mu 1) \tag{3.4}$$

where $\Phi = (\phi(w_i, w_j))_{n\times n}$, $y = (y_1, ..., y_n)'$ is the response vector. The analytical expression of μ is:

$$\hat{\mu} = (1'\Phi^{-1}1)^{-1}1'\Phi^{-1}y \tag{3.5}$$

Take this into the negative log-likelihood function and we estimate the parameters using:

$$[\sigma^2, \theta, \eta, \nu, \tau^2] = argmin\{\log|\Phi| + (y'\Phi^{-1}y) - (1'\Phi^{-1}1)^{-1}(1'\Phi^{-1}y)^2\} \tag{3.6}$$

With all the parameters estimated, the prediction mean and variance of input w_* can be expressed as:

$$\hat{Y}(w_*) = \hat{\mu} + \gamma'\Phi^{-1}(y - \hat{\mu}1) \tag{3.7}$$

$$s^2(w_*) = \phi(w_*, w_*) - \gamma'\Phi^{-1}\gamma + \frac{(1 - 1'\Phi^{-1}\gamma)^2}{1'\Phi^{-1}1} \tag{3.8}$$

where γ is covariance vector $(\phi(w_*, w_i))_{n\times 1}$.

3.3 Analytical Gradient of Parameter Estimation

We calculate analytical gradient each parameter and use it in R package 'nloptr' to speed up the calculation.

The negative log-likelihood function:$l = \log|\Phi| + (y - \mu 1)'\Phi^{-1}(y - \mu 1)$. Since Φ is invertible, by Jacobi's fomula:

$$\frac{\partial}{\partial x}|\Phi(x)| = |\Phi(x)|tr(\Phi(x)^{-1}\frac{\partial}{\partial x}\Phi(x)) \tag{3.9}$$

Then we can calculate the derivative of function $l(\Phi)$:

$$\frac{\partial l(\Phi)}{\partial \bullet} = tr(\Phi^{-1}\frac{\partial \Phi}{\partial \bullet}) - (y-\hat{\mu}1)'\Phi^{-1}\frac{\partial \Phi}{\partial \bullet}\Phi^{-1}(y-\hat{\mu}1). \tag{3.10}$$

The gradients of each parameter can be calculated:

$$\frac{\partial \phi(w_i, w_j)}{\partial \sigma_h^2} = exp\{-\theta_h(x_{i,h}-x_{j,h})^2 - (g(t_{i,h})-g(t_{j,h}))^2\} \tag{3.11}$$

$$\frac{\partial \phi(w_i, w_j)}{\partial \theta_h} = -\sigma_h^2(x_{i,h}-x_{j,h})^2 exp\{-\theta_h(x_{i,h}-x_{j,h})^2-(g(t_{i,h})-g(t_{j,h}))^2\} \tag{3.12}$$

$$\begin{aligned}\frac{\partial g(t_{i,h})}{\partial \eta_h} &= \frac{(t_{i,h}-\nu_h)e^{-\eta_h(t_{i,h}-\nu_h)}}{(1+e^{-\eta_h(t_{i,h}-\nu_h)})^2} \\ &= g(t_{i,h})^2(t_{i,h}-\nu_h)e^{-\eta_h(t_{i,h}-\nu_h)}\end{aligned} \tag{3.13}$$

$$\begin{aligned}\frac{\partial \phi(w_i, w_j)}{\partial \eta_h} =& -2\sigma_h^2(g(t_{i,h})-g(t_{j,h}))\frac{\partial}{\partial \eta_h}(g(t_{i,h})-g(t_{j,h}))exp\{-\theta_h(x_{i,h}-x_{j,h})^2 \\ & -(g(t_{i,h})-g(t_{j,h}))^2\} \\ =& -2\sigma_h^2(g(t_{i,h})-g(t_{j,h}))(g(t_{i,h})^2(t_{i,h}-\nu_h)e^{-\eta_h(t_{i,h}-\nu_h)} - g(t_{j,h})^2 \\ & (t_{j,h}-\nu_h)e^{-\eta_h(t_{j,h}-\nu_h)})exp\{-\theta_h(x_{i,h}-x_{j,h})^2-(g(t_{i,h})-g(t_{j,h}))^2\}\end{aligned} \tag{3.14}$$

$$\begin{aligned}\frac{\partial g(t_{i,h})}{\partial \nu_h} &= \frac{-\eta_h e^{-\eta_h(t_{i,h}-\nu_h)}}{(1+e^{-\eta_h(t_{i,h}-\nu_h)})^2} \\ &= -g(t_{i,h})^2\eta_h e^{-\eta_h(t_{i,h}-\nu_h)}\end{aligned} \tag{3.15}$$

$$
\begin{aligned}
\frac{\partial \phi(w_i, w_j)}{\partial \nu_h} =& -2\sigma_h^2(g(t_{i,h}) - g(t_{j,h}))\frac{\partial}{\partial \nu_h}(g(t_{i,h}) - g(t_{j,h}))exp\{-\theta_h(x_{i,h} - x_{j,h})^2 \\
& - (g(t_{i,h}) - g(t_{j,h}))^2\} \\
=& 2\sigma_h^2\eta_h(g(t_{i,h}) - g(t_{j,h}))(g(t_{i,h})^2 e^{-\eta_h(t_{i,h}-\nu_h)} - g(t_{j,h})^2 e^{-\eta_h(t_{j,h}-\nu_h)}) \\
& exp\{-\theta_h(x_{i,h} - x_{j,h})^2 - (g(t_{i,h}) - g(t_{j,h}))^2\}
\end{aligned} \tag{3.16}
$$

Chapter 4

Active Learning Approach

Optimization methods often require more function evaluations than we have, which can be a big challenge to global optimization. To overcome this difficulty, Jones et al., 1998 proposed a response surface method using an approximating functions to construct an efficient global optimization (EGO) algorithm with a credible stopping rule. This method can both exploit the approximating surface and improve the approximation. It has been shown that this methodology is especially good at modeling linear and non-linear functions.

4.1 Efficient Global Estimation

The expected improvement (EI) criterion is a weighted function of all possible improvements by corresponding density value. Take $f_{min} = min(y^{(1)}, ..., y^{(n)})$ as the current best value and the improvement at each point as $I = max(f_{min} - y, 0)$. Then we simply take the expected value: $E[I] = E[max(f_{min} - y, 0)]$. Y is a random variable following $N(\hat{Y}, s^2)$. EI criterion for dosage-time inputs $w_i = (x_i', t_i')'$ can be calculated by:

$$EI(w) = (y_{min} - \hat{Y}(w))\Phi(\frac{y_{min} - \hat{Y}(w)}{s(w)}) + s(w)\phi(\frac{y_{min} - \hat{Y}(w)}{s(w)}) \quad (4.1)$$

ϕ and Φ are the standard normal density and distribution function. In the beginning, the initial Kriging model is built based on some initial points, which we will discuss later in this chapter. Then, an iterative process is executed. In each iteration, a new candidate is selected to maximize the EI function.

$$w_{n+1} = \arg maxEI(w) \tag{4.2}$$

The first part in equation 4.1 is maximized by reaching the minimum $\hat{Y(w)}$, and the second part is maximized by enlarging $s(w)$. Minimizing the prediction $\hat{Y(w)}$ has the benefit to exploit around the current best solution, while maximizing standard deviation $s(w)$ has the benefit to explore different space. We solve this optimization by R package 'genound', which is an optimization method that combines evolutionary search algorithms with derivative-based Newton or quasi-Newton methods (Mebane Jr, 2019). The selected next point is then used to update the kringing model. The iteration is stopped if the expected improvement is less than 1% of the best current function value (Jones et al., 1998; Xiao, Wang, et al., 2022).

The algorithm can be divided into four steps:

1. Fit the GP model using the current n observations and corresponding responses.

2. Select next design w_{n+1} which maximizes the EI function.

3. Refit the GP model based on the previous and new observation.

4. Repeat this calculation until values of EI function make improvements not more than a threshold.

4.2 Analytical Gradient of Expected Improvement

Here we calculate the analytical gradient and use it in R package 'genound' to speed up the optimization.

$$\nabla EI(w) = -\nabla\hat{Y}(w) \times \Phi(\frac{y_{min} - \hat{Y}(w)}{s(w)}) + \nabla s(w)\phi(\frac{y_{min} - \hat{Y}(w)}{s(w)}) \tag{4.3}$$

Given prediction mean $\hat{Y}(w_*) = \hat{\mu} + \gamma'\Phi^{-1}(y - \hat{\mu}1)$, and $\gamma = (\phi(w_*, w_i))_{n\times 1}$,

$$\nabla\hat{Y}(w_*) = \nabla\gamma'\Phi^{-1}(y - \hat{\mu}1) \tag{4.4}$$

$$\nabla s(w_*) = \frac{1}{2s(w*)}\nabla s^2(w_*) = -\frac{1}{s(w*)}(\nabla\gamma'\Phi^{-1}\gamma + \frac{(1 - 1'\Phi^{-1}\gamma)\nabla\gamma'\Phi^{-1}1}{1'\Phi^{-1}1}) \tag{4.5}$$

The gradient of x and t are respectively shown as :

$$\frac{\nabla\phi(w_*, w_i)}{\nabla x_{*,h}} = -2\theta_h\sigma_h^2(x_{*,h}-x_{i,h})exp\{-\theta_h(x_{*,h}-x_{i,h})^2-(g(t_{*,h})-g(t_{i,h}))^2\} \tag{4.6}$$

$$\begin{aligned}\frac{\nabla\phi(w_*, w_i)}{\nabla t_{*,h}} = &-2\eta_h\sigma_h^2(g(t_{*,h}) - g(t_{i,h}))g(t_{*,h})^2exp\{-\eta_h(t_{*,h} - \nu_h) \\ &- \theta_h(x_{*,h} - x_{i,h})^2 - (g(t_{*,h}) - g(t_{i,h}))^2\}\end{aligned} \tag{4.7}$$

Then $\nabla\gamma'$ can be expressed as:

$$\frac{\nabla\gamma'}{\nabla x_*} = \begin{pmatrix} \frac{\nabla\phi(w_*,w_1)}{\nabla x_{*,1}} & \cdots & \frac{\nabla\phi(w_*,w_n)}{\nabla x_{*,1}} \\ \vdots & \vdots & \vdots \\ \frac{\nabla\phi(w_*,w_1)}{\nabla x_{*,n}} & \cdots & \frac{\nabla\phi(w_*,w_n)}{\nabla x_{*,n}} \end{pmatrix}, \tag{4.8}$$

$$\frac{\nabla\gamma'}{\nabla t_*} = \begin{pmatrix} \frac{\nabla\phi(w_*,w_1)}{\nabla t_{*,1}} & \cdots & \frac{\nabla\phi(w_*,w_n)}{\nabla t_{*,1}} \\ \vdots & \vdots & \vdots \\ \frac{\nabla\phi(w_*,w_1)}{\nabla t_{*,n}} & \cdots & \frac{\nabla\phi(w_*,w_n)}{\nabla t_{*,n}} \end{pmatrix}. \tag{4.9}$$

Next calculate $\nabla\hat{Y}(w_*)$ and $\nabla s(w_*)$ using $\nabla\gamma'$, and then derive $\nabla EI(w)$.

4.3 Optimal Initial Design

As mentioned before, the initial Kriging model is built based on some initial points, which are generated by experimental design methods. Before designing initial points of X and T, we introduce a kind of typical design method, which we will use later.

4.3.1 Space filling design and Latin Hypercube design (LHD)

In simple language, a space-filling design has points everywhere in the whole region with as few gaps or holes as possible (Joseph, 2016). The goal is to extract maximum information about the non-linear surface so that kriging models can provide the best possible approximation.

Space-filling design places the design points to cover the experimental region well meanwhile make them as far apart as possible. The first property is called minimax distance design and the second is called maximin distance design (Johnson et al., 1990).

Suppose all the design points can be scaled in a hypercube $\chi = [0, 1]^p$. The design has n points $X = [x_1, ..., x_n]$. For any point $z \in [0, 1]^p$, we calculate the Euclidean distances and find the nearest design point as $\min_i d(z, x_i)$. The worst point in the whole region has the largest distance to its nearest point and the distance can be noted as $\max_\chi \min_i d(z, x_i)$. We want to find the design which has the minimum of this distance value, which is:

$$\min_X \max_{z \in \chi} \min_i d(z, x_i). \tag{4.10}$$

Also, the minimum distance among those points is $\min_{i,j} d(x_i, x_j)$. The space filling design is found by maximizing the minimum distance:

$$\max_X \min_{i,j} d(x_i, x_j). \tag{4.11}$$

Latin Hypercube design (LHD)(McKay et al., 2000) is a good choice for space-filling design, since it avoids the computation replication caused by sparsity in the whole space for lower-dimension projections. The range of each dimension is divided into n equally spaced intervals. Then, only one coordinate of a design point is sampled from each of the interval. For example, in a two-dimension square, there is one and only one point in each row and column.

Morris and Mitchell, 1995 gives a criterion for choosing the maximin Latin hypercube design(MmLHD):

$$\min_X \{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{d^k(x_i, x_j)}\}^{1/k}. \tag{4.12}$$

The parameter k is a positive integer. The figure 4.1 shows an example of Maximin Latin Hypercube design for $n = 7$.

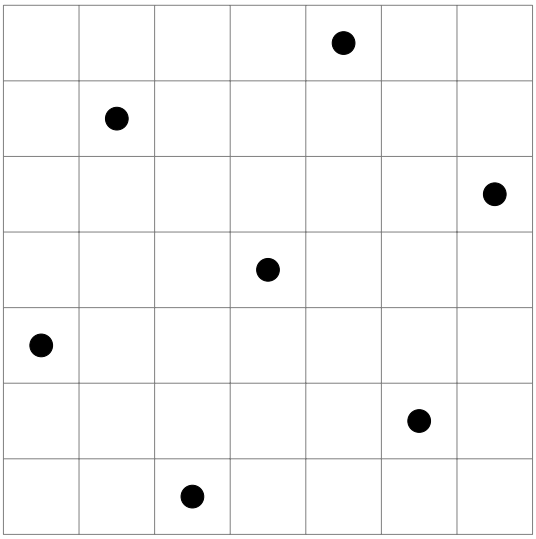

Figure 4.1: Maximin Latin Hypercube design for $n = 7$

4.3.2 Initial design for dose and time

The initial design consists of two parts: time matrix T and dose matrix X. Good initial designs can efficiently save running times and generate better solutions. In this section, we hope to find sequence-balanced and space-filling initial design for T, then construct optimal design for X based on T.

Initial design for T

Here we define a v_p criterion to optimize initial T matrix:

$$\nu_p = (\rho_1 \sum_{i=1}^{k} \sum_{j=1}^{k} \frac{1}{(t_{i,j}+1)^p} + \rho_2 \sum_{i=2}^{n} \sum_{j=1}^{i-1} \frac{1}{(d_{i,j}+1)^p})^{\frac{1}{p}}, \tag{4.13}$$

where $t_{i,j}$ is the number of appearances that t_i is smaller than t_j and $d_{i,j}$ is the Euclidean distance between $i\text{-}th$ input and $j\text{-}th$ input. We take the same parameter values as $p = 15$, $\rho_1 = 0.2$, and $\rho_2 = 0.8$, which means considering more on the second part.

ν_p is to be minimized, which equals to maximizing both minimum $t_{i,j}$ and $d_{i,j}$. The first part $\sum_{i=1}^{k} \sum_{j=1}^{k} 1/(t_{i,j}+1)^p$ guarantees the balance of the design and the second part $\sum_{i=2}^{n} \sum_{j=1}^{i-1} 1/(d_{i,j}+1)^p$ seeks the space-filling properties.

This optimization can be calculated by standard threshold accepting (TA) algorithm (Dueck & Scheuer, 1990; Xiao & Xu, 2018). It starts with a random matrix and then find a better neighbor design by exchanging two random rows of T. Here is an example of initial optimized T matrix of 5 components ranging from 0 to 24.

$$\begin{pmatrix} 11.92 & 8.13 & 14.04 & 15.93 & 0.00 \\ 1.93 & 19.44 & 23.97 & 0.00 & 20.54 \\ 4.38 & 12.16 & 2.15 & 0.00 & 8.14 \\ 0.00 & 8.95 & 0.51 & 18.93 & 16.59 \\ 17.98 & 0.00 & 6.94 & 5.42 & 7.79 \end{pmatrix}$$

The t_{ij} values of this initial T matrix is:

Table 4.1: t_{ij} values of initial T design

t_{12}	t_{13}	t_{14}	t_{15}	t_{21}	t_{23}	t_{24}	t_{25}	t_{31}	t_{32}
3	3	2	3	2	3	3	2	2	2
t_{34}	t_{35}	t_{41}	t_{42}	t_{43}	t_{45}	t_{51}	t_{52}	t_{53}	t_{54}
2	3	3	2	3	3	2	3	2	2

Initial design for X

Next find optimal initial design of X based on T. We define a C_p criterion to optimize X matrix:

$$C_p = (\sum_{i=2}^{n} \sum_{j=1}^{i-1} \frac{1}{(\rho_1' l_{i,j} + \rho_2' d_{i,j} + 1)^p})^{\frac{1}{p}}, \tag{4.14}$$

where $l_{i,j}$ is Euclidean distance between $i\text{-}th$ input and $j\text{-}th$ input of X and $d_{i,j}$ is the Euclidean distance between $i\text{-}th$ input and $j\text{-}th$ input of T. Here we take $p = 15$, $\rho_1' = \rho_2' = 0.5$.

C_p is to be minimized which is equal to maximizing the distance between each two runs for both T and X. Considering the space-filling property of the dose design, we start with a Latin Hypercube design (LHD) (Ba et al., 2015) using R package 'FastMmLHD' (H. Wang et al., 2020). This optimization can also be calculated by TA algorithm. The neighbor matrix is defined by exchanging two random rows of X.

After finding the initial optimized X, we need to rescale each row of X to $[0, 1]$ range for further use. Here is the example of initial optimized X matrix of 5 components ranging from 0 to 1 based on the previous T matrx:

$$\begin{pmatrix} 0.00 & 0.25 & 0.50 & 0.75 & 1.00 \\ 0.25 & 0.75 & 1.00 & 0.50 & 0.00 \\ 0.50 & 1.00 & 0.25 & 0.00 & 0.75 \\ 0.75 & 0.50 & 0.00 & 1.00 & 0.25 \\ 1.00 & 0.00 & 0.75 & 0.25 & 0.50 \end{pmatrix}$$

Chapter 5

Simulation Studies

Usually, the practical situation is complicated. There are many drug components which can be effective or ineffective and we need to make an optimal plan. Every two of them can have a positive or negative interaction affected by action time and dose. Also, there may be a time constraint, for example, every two drugs cannot be used together in one hour.

Those different situations can be summarized by three separate problems as below. We give solutions to the three simulated cases. In the first case, there are five drugs in total. Four of them have positive effect and the other one is negative. We expect the simulation result shows which one has negative effect to be omitted. In the second case, five drugs are all set to have positive impact and there is a time constraint that the time interval between every two drugs is no less than 3 hours. In the third case, there are total eight drugs. Some of them are more effective while the others are less effective. We expect to pick an optimal combination.

Since drug effect is a convex function of both duration of time and dose. Specifically, as time increases, the response value of treatment effect always goes up at first and then decreases and it is a monotonic increasing convex function of dose given a fixed action time. Here we use a quadratic function to simulate time part and log function for dose part. It is usually believed that greater or less efficacy is possible with the combination than with any drug alone. For simplicity, we describe interaction as a function decided by doses of every two drugs and the time interval.

Suppose we have five main drugs in our simulation cases. g is the vector of effect coefficients. The dose of each drug x_i ranges from 0 to 1 and action time t_i is from 0 to 24, $i = 1, , , k$. u is the duration of each drug, which is the time length from adding the drug to the end. In the following three cases, the

treatment effect is tested at 30th hour, thus the time effect is a function of $u_i = (30 - t_i)/24$. k is the number of components. h, l represent serial numbers of every two used drugs. $s_{h,l}$ denotes the interaction coefficient between drug h and l, which can be shown in a upper $k \times k$ matrix. The absolute values of s among the drugs in our cases are generated by a uniform distribution $U(5, 30)$. Positive values represent for synergistic effect and negative for antagonist effect. The drug effect function involving action time and dosage is:

$$F(x,t) = \sum_{i=1}^{k} g_i \ln(1 + x_i) \times (2.4u_i - 2u_i^2) + \sum_{h,l=1}^{k} \frac{x_h^2 x_l^2}{1 + (|t_h - t_l|)/24} s_{h,l}$$

5.1 Case 1 dose and time optimization

This case can be seen as general optimization problem for continuous variables. Four of them have positive effect and the other one is negative. The algorithm can be summarized as:

Initialize number of runs N.

Set an empty vector EI and an empty matrix W.

for $i = 1$ to N **do**

Maximize $f(w)$ by Genetic optimization using derivatives and denote the optimal result w_{opt}.

$ei[i] = f(w_{opt})$

$W = rbind(W, w_{opt})$

Refit MaGP model bades on W and $y(W)$.

end for

$f(w)$ is EI value of new vector w based on all the current observations.

Set the effect coefficient $g = (26.8, 23.2, 32.4, 30, -15)$ and the interaction coefficient matrix $s = \begin{pmatrix} 0 & -17.4 & -19.8 & 6.9 & -12.4 \\ & 0 & 8.1 & -16.8 & -13.2 \\ & & 0 & -28.4 & 7.2 \\ & & & 0 & -11.6 \\ & & & & 0 \end{pmatrix}$.

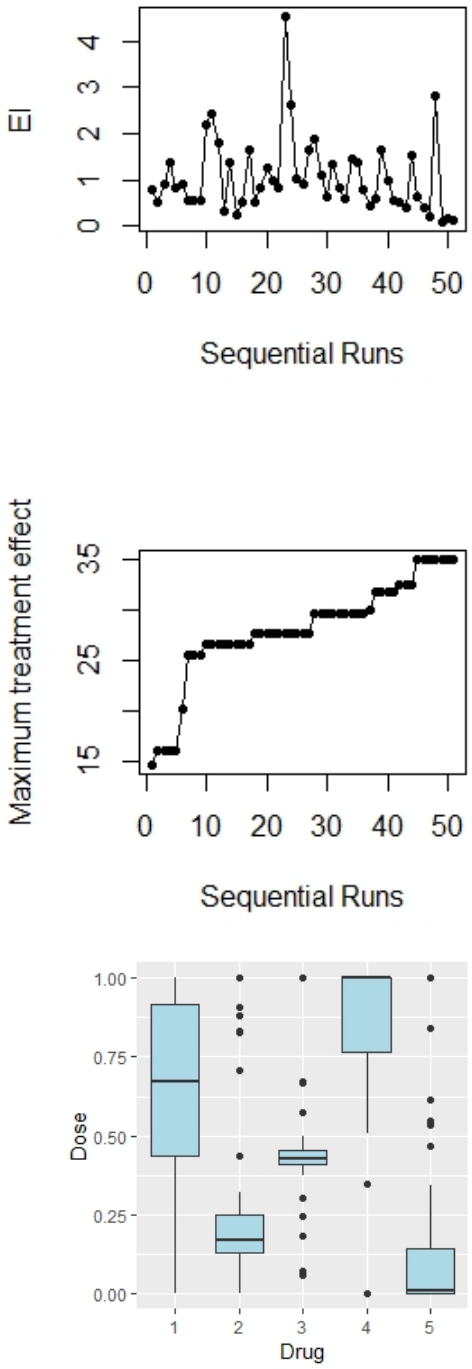

Figure 5.1: EI, response values, and dose box-plot of case 1

5.1 shows it runs 51 times before meeting the stop criterion. As the number of runs increases, the maximum effect reaches 35.02. The corresponding dosage of five drugs is $x = (1.00, 0.24, 0.39, 1.00, 0)$. The box-plot of optimal doses for five drugs per run shows the median optimal value of 5th drug is 0, thus this negative effect one should not be used. For the other four positive drugs, the optimal time setting is $t = (10.40, 13.48, 14.60, 16.39)$

5.2 Case 2 optimization with time constraint

The time interval between every two drugs is no less than 3 hours. The setting of coefficient g and interaction matrix s are the same as case 1. If the time between any of two drugs is less than three hours, we give EI value a large penalty. If the optimized dose value of is 0, we set the corresponding time value is 24 for simplicity.

The algorithm can be summarized as:

Initialize number of runs N.

Set an empty vector EI and an empty matrix W.

for $i = 1$ to N **do**

Maximize $f(w)$ by Genetic optimization using derivatives and denote the optimal result w_{opt}.

$ei[i] = f(w_{opt})$

if $|t_i - t_j| < 3, ei = ei - 10000$

$W = rbind(W, w_{opt})$

Refit MaGP model bades on W and $y(W)$.

end for

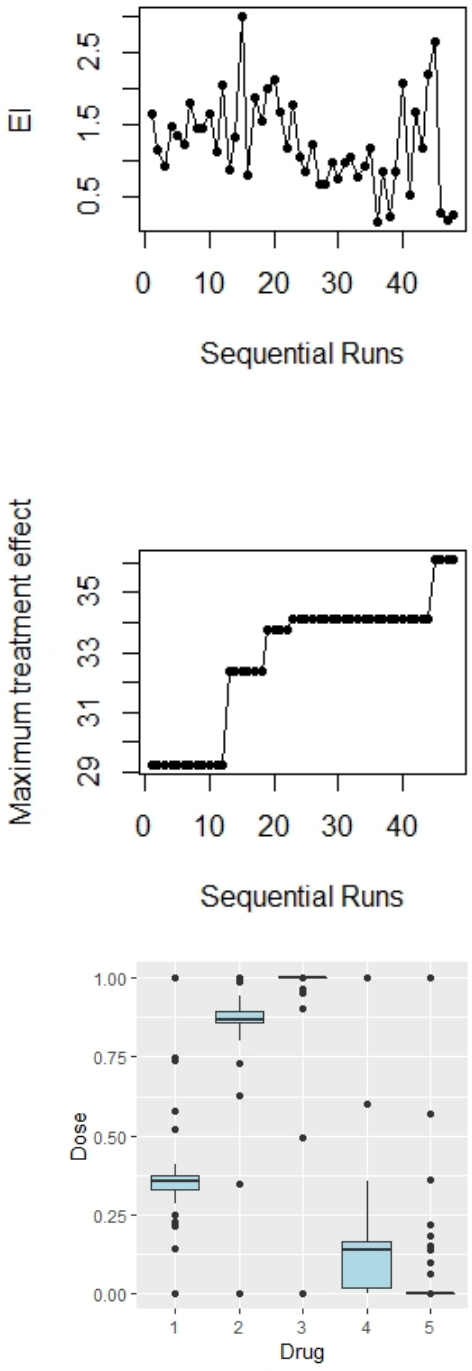

Figure 5.2: EI, response values, and dose box-plot of case 2

The result shows it has 48 sequential runs in total. As the number of runs increases, the maximum effect reaches 36.57. The corresponding action time of five drugs is $t = (10.54, 14.47, 19.45, 22.54, 24)$ and dose setting of five drugs is $x = (0.29, 1, 1, 0.16, 0)$, which shows that first four drugs are used.

5.3 Case 3 optimization with time constraint and component screening

The problem with time constraint and component screening can be divided into two parts. First we do the same sequential run as case 2, then we select the important components whose medium values in the box-plot are not zero. Then for the next step, we only do optimization for those selected components with time constraint.

Based on the previous two cases, we add several more drugs which have different effect. Here we use eight drugs in total with different coefficient settings $g = c(26.8, 23.2, 32.4, 30, 15, 9.6, 3.7, 0)$, and interaction coefficient matrix:

$$s = \begin{pmatrix} 0 & -17.4 & -19.8 & 6.9 & -12.4 & 3.5 & 1.7 & 0 \\ & 0 & 8.1 & -16.8 & -13.2 & -2.8 & 2.5 & 0 \\ & & 0 & -28.4 & 7.2 & -7.9 & -1.8 & 0 \\ & & & 0 & -11.6 & -6.4 & -0.9 & 0 \\ & & & & 0 & -1.6 & 1.2 & 0 \\ & & & & & 0 & -0.4 & 0 \\ & & & & & & 0 & 0 \\ & & & & & & & 0 \end{pmatrix}$$

.

The algorithm can be summarized as:

Initialize number of runs N.

Set an empty vector EI and an empty matrix W.

for $i = 1$ to N **do**

Maximize $f(w)$ by Genetic optimization using derivatives and denote the optimal result w_{opt}.

$ei[i] = f(w_{opt}) - p \times \sum_{i=1}^{n} x_i$

if $|t_i - t_j| < 3, ei = ei - 10000$

$W = rbind(W, w_{opt})$

Refit MaGP model bades on W and $y(W)$.

end for

p is the penalty of the sum of all dose values. Here we set $p = 0.15$. The box-plot shows the distribution of optimized dose values of each drug in each run indicating drug 2,3,4,5 are significant components, since the medium values of other ones are 0. Next we omit those components and find optimized setting for other four significant ones with time constraint. The algorithm is the same as case 2. The following figures show the result.

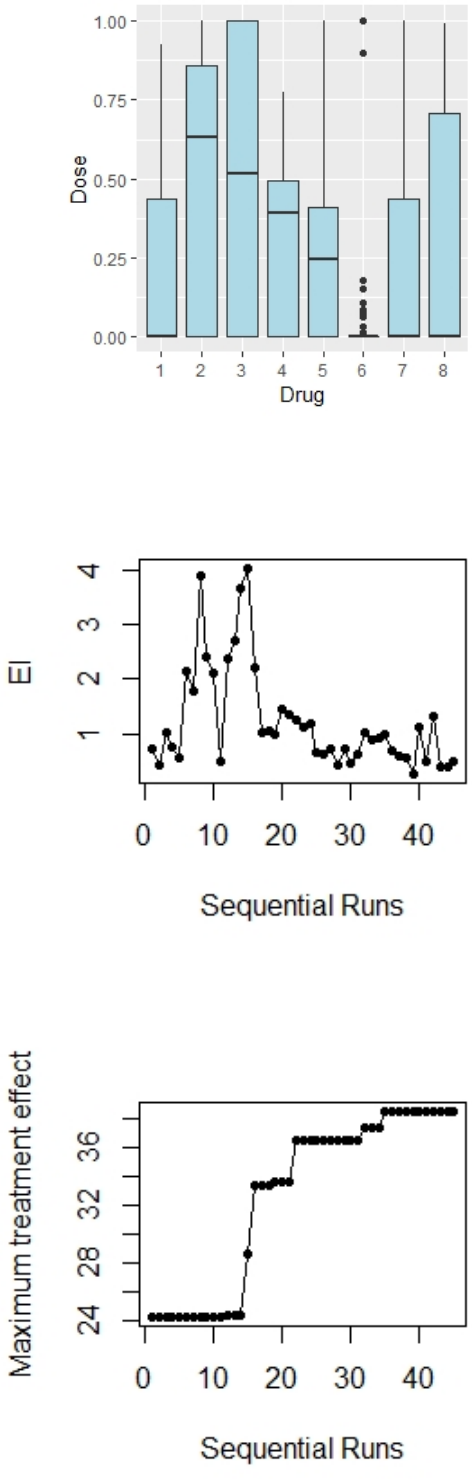

Figure 5.3: Dose boxp-lot, EI, and response values of case 3

It runs 45 times before meeting the stop criterion. The maximum effect reaches 38.49. The corresponding action time of four drugs is $t = (18.63, 14.76, 22.70, 9.59)$ and dose setting of four drugs is $x = (1, 1, 0.19, 0.75)$.

The three cases are designed progressively to achieve optimization of timing and dose with time constraint and component screening. From each of the maximum treatment effect plot, the response variable improves greatly compared to the initial values. Each sequential run of optimization takes about one minute to run in R.

Chapter 6

Discussion

Compared to previous study, this paper solve a more complex problem to find best drug combinations with component screening and time constraint. We propose a novel Gaussian process model with mapping functions, use an active learning method, and generate initial data using optimal initial design for time and dose separately. The simulation studies show the result of the optimization algorithm progressively and the final case is divided into two steps: first we do the screening, then find optimal result under time constraint. This novel GP is efficient since it can find a pretty good result under the constraints based on only several runs of initial design.

This novel GP model can also be applied to other research. For example, we can use it to find optimized solution for the famous problem: traveling salesman, which is to make optimized plan for the salesman to travel and complete business work among many cities. Each city has a daily profit and cost, and there will be a penalty if the completion time of the business passes the due time. After completing the work in each city, he will earn a fixed profit and a variable income for everyday, while there is also a cost each day for completing the task. The salesman can make his own schedule to decide which cities he will go and how long he will stay in each selected city. This is an extension of the original traveling salesman problem. We need to optimize both beginning time and numbers of days for staying with due time constraints and component screening. This can be solved by the method the same as case 3. First we screen out the most profitable cities and then decide the optimal beginning time and how long to stay.

One potential problem is that since we add the time constraint and penalty for component screening to EI value in each sequential run, the convergent property of the model may not hold. This leads to the problem that the total se-

quential running times before meeting the stop criterion are not stable enough. This can be a further topic of improvement for the novel GP model research.

Appendix A

Output Data

Case 1

Initial design

	X1	X2	X3	X4	X5	T1	T2	T3	T4	T5	y
1	0.25	0.75	1.00	0.50	0.00	14.206	0.000	4.054	14.904	10.014	13.556
2	1.00	0.00	0.75	0.25	0.50	13.624	23.062	1.977	0.000	9.770	-0.186
3	0.75	0.50	0.00	1.00	0.25	13.563	4.674	20.928	0.000	20.642	5.071
4	0.50	1.00	0.25	0.00	0.75	3.975	15.782	23.125	16.910	0.000	10.588
5	0.00	0.25	0.50	0.75	1.00	0.000	17.792	6.047	18.582	21.984	6.045

Sequential output

	X1	X2	X3	X4	X5	T1	T2	T3	T4	T5	y
1	0.341	0.906	0.247	0.001	0.002	8.872	6.951	20.225	19.341	10.066	14.564
2	0.308	0.882	0.185	0.001	0.001	16.270	10.059	23.816	13.837	10.378	15.917
3	0.001	0.832	0.072	0.001	0.839	15.328	20.890	17.558	15.556	14.877	-1.276
4	0.383	0.829	0.059	0.688	1.000	17.262	6.860	24.000	18.211	24.000	3.817
5	0.001	0.194	0.442	0.001	0.001	4.478	18.286	6.222	12.354	6.905	7.839
6	0.384	0.320	0.481	0.804	0.001	4.523	17.745	18.595	23.664	5.950	20.158
7	0.410	0.237	0.576	0.834	0.001	16.068	7.170	18.228	16.196	8.746	25.542
8	0.463	0.001	0.674	0.912	0.001	4.127	10.185	16.922	22.510	24.000	16.120
9	0.395	0.437	0.666	1.000	0.001	3.684	2.482	23.218	0.019	8.236	-0.194
10	0.515	0.195	0.455	0.794	0.098	17.310	15.754	20.693	15.383	17.233	26.632
11	0.438	0.264	0.304	0.833	0.143	18.501	7.305	23.999	23.725	23.908	19.235
12	0.705	1.000	0.483	0.803	0.001	18.962	24.000	22.083	16.257	9.248	20.328
13	0.318	0.295	0.482	0.766	0.085	21.559	23.412	18.765	8.988	18.403	21.566
14	0.651	0.199	0.464	0.730	0.001	10.930	10.078	23.374	21.749	18.711	23.570
15	0.411	0.193	0.452	0.794	0.001	20.144	10.562	20.556	12.653	8.787	25.068
16	0.443	0.199	0.476	0.834	0.001	22.299	3.989	22.524	19.253	18.519	21.542

17	0.552	0.131	0.501	1.000	0.280	22.921	0.829	21.596	11.860	6.901	20.140
18	0.567	0.139	0.419	1.000	0.098	20.377	10.425	16.912	19.863	1.638	27.681
19	1.000	0.150	0.373	1.000	0.077	7.670	4.366	20.852	21.250	4.747	27.051
20	1.000	1.000	0.245	0.897	0.110	8.254	3.453	20.493	14.683	6.885	9.515
21	0.380	0.041	0.425	0.875	0.001	17.946	2.828	17.040	11.229	15.610	23.605
22	0.624	0.105	0.426	0.761	0.001	6.084	12.172	18.336	15.497	10.806	24.412
23	0.337	0.154	0.412	1.000	0.044	16.184	20.862	19.789	20.400	15.179	23.090
24	0.438	0.001	0.429	0.671	0.001	23.696	14.469	14.628	17.209	1.189	21.885
25	0.542	0.127	0.429	1.000	0.057	21.755	4.743	13.812	14.007	13.337	25.385
26	0.554	0.143	0.427	1.000	0.033	13.738	8.739	21.531	18.153	8.813	27.282
27	0.672	0.147	0.392	1.000	0.015	16.143	23.085	22.976	21.280	10.262	26.746
28	1.000	0.126	0.449	1.000	0.320	19.291	16.880	13.617	20.708	20.944	29.580
29	0.699	0.157	0.450	1.000	0.319	12.881	5.787	21.618	19.616	20.851	23.565
30	0.916	0.144	0.429	0.349	0.045	21.109	2.715	10.973	12.423	9.080	21.962
31	1.000	0.149	0.443	1.000	0.021	7.445	14.138	1.482	15.848	16.045	24.339
32	1.000	0.185	0.457	1.000	1.000	19.034	1.243	22.455	13.114	1.469	16.014
33	0.886	0.290	0.479	1.000	0.344	10.231	8.532	7.963	23.144	0.001	24.768
34	0.541	0.098	0.407	1.000	0.012	23.999	4.619	15.352	23.953	11.515	21.684
35	0.965	0.149	0.449	1.000	0.154	0.717	12.918	13.899	17.493	8.959	18.876
36	0.669	0.175	0.490	0.001	0.142	9.805	19.788	13.545	18.563	0.072	18.087
37	1.000	0.118	0.448	1.000	0.001	15.278	20.005	17.472	23.996	8.925	29.977
38	0.904	0.171	0.426	1.000	0.001	11.433	17.518	21.229	19.113	8.154	31.757
39	1.000	0.181	0.434	1.000	0.615	20.754	7.943	11.469	16.621	1.259	29.245
40	0.915	0.159	0.387	1.000	0.466	8.288	24.000	13.585	22.101	0.032	26.057
41	0.892	0.175	0.438	0.624	0.001	11.682	19.595	14.774	0.001	15.083	17.556
42	0.859	0.001	0.419	1.000	0.001	17.742	21.415	13.725	19.027	16.615	32.394
43	0.821	0.084	0.434	1.000	0.550	14.164	23.930	15.413	17.059	23.265	24.460
44	0.876	0.160	0.455	1.000	0.001	12.159	17.388	21.269	19.134	16.991	31.129
45	1.000	0.238	0.387	1.000	0.001	10.400	13.479	14.601	16.392	23.454	35.016
46	1.000	0.286	1.000	1.000	0.001	13.505	13.712	2.426	19.424	15.953	7.055
47	0.870	0.186	0.422	0.518	0.001	10.608	6.296	7.828	13.987	13.519	24.298
48	1.000	0.707	0.429	1.000	0.001	9.483	0.288	21.561	18.676	17.246	18.224
49	0.916	0.048	0.417	0.507	0.535	23.810	20.855	8.383	22.644	1.039	19.207
50	0.844	0.085	0.408	1.000	0.001	14.029	19.347	3.409	18.185	8.661	29.133
51	0.835	0.103	0.410	1.000	0.001	16.896	11.822	7.508	20.255	22.090	31.237

Case 2

Initial design

	X1	X2	X3	X4	X5	T1	T2	T3	T4	T5	y
1	0.00	0.25	0.50	0.75	1.00	0.000	12.774	0.057	7.026	21.410	-3.665
2	0.75	0.50	0.00	1.00	0.25	12.733	0.691	15.981	0.000	7.551	1.380
3	0.50	1.00	0.25	0.00	0.75	18.809	17.490	0.000	10.641	4.373	5.861
4	1.00	0.00	0.75	0.25	0.50	13.995	0.000	3.698	23.626	16.918	6.042
5	0.25	0.75	1.00	0.50	0.00	1.107	14.599	18.927	18.798	0.000	26.627

Sequential output

	X1	X2	X3	X4	X5	T1	T2	T3	T4	T5	y
1	0.404	0.944	1.000	0.001	0.001	13.339	17.409	24.000	24.000	24.000	29.248
2	0.522	0.001	1.000	0.023	0.003	17.870	24.000	24.000	12.706	24.000	14.735
3	1.000	0.985	1.000	0.358	0.001	19.324	16.324	23.987	0.013	24.000	5.247
4	0.346	0.626	0.001	0.001	0.001	21.859	0.001	24.000	24.000	24.000	2.838
5	0.363	0.001	1.000	0.601	0.568	3.145	24.000	13.108	23.983	7.418	12.542
6	0.001	0.801	1.000	0.001	0.001	24.000	13.671	22.566	24.000	24.000	25.850
7	0.379	0.728	1.000	0.001	0.002	2.120	16.919	20.570	24.000	24.000	25.364
8	0.001	0.818	1.000	0.001	0.001	24.000	22.645	13.931	24.000	24.000	27.540
9	0.291	1.000	0.901	0.001	0.063	2.505	19.840	23.679	24.000	8.799	25.266
10	0.393	0.943	1.000	0.261	0.001	3.518	18.434	24.000	14.933	24.000	28.703
11	0.215	0.899	1.000	0.192	0.001	7.711	21.835	2.670	11.636	24.000	19.333
12	0.577	0.913	1.000	0.160	0.001	2.457	19.485	12.706	23.715	24.000	26.269
13	0.335	0.885	1.000	0.168	0.001	4.441	16.350	20.587	12.496	24.000	32.342
14	0.225	0.858	1.000	0.131	0.001	2.603	8.713	17.143	20.809	24.000	29.707
15	0.366	0.864	0.951	0.010	0.152	12.631	17.548	20.851	24.000	0.001	31.395
16	0.321	0.859	0.966	1.000	0.001	17.790	6.206	20.827	0.006	24.000	-0.446
17	0.146	0.867	1.000	0.010	0.220	3.602	23.396	20.384	24.000	6.710	26.305
18	0.354	0.854	1.000	0.181	0.001	1.583	16.852	9.239	23.281	24.000	26.865
19	0.357	0.858	1.000	0.165	0.001	10.412	19.773	14.782	23.836	24.000	33.727
20	0.352	0.831	1.000	0.139	0.001	4.434	23.320	20.077	11.473	24.000	28.770
21	0.383	0.832	1.000	0.186	0.001	1.018	23.521	15.746	12.733	24.000	26.995
22	0.371	0.853	1.000	0.172	0.001	1.521	20.002	23.174	16.861	24.000	26.525
23	0.357	0.881	1.000	0.140	0.001	9.091	20.037	16.547	23.207	24.000	34.110
24	0.365	0.867	1.000	0.010	0.360	4.830	20.566	17.562	24.000	23.984	27.774
25	0.359	0.873	1.000	0.001	1.000	7.226	15.180	18.607	24.000	3.518	26.312
26	0.350	0.887	1.000	0.109	0.001	2.262	14.291	22.073	18.272	24.000	28.337
27	0.382	0.864	1.000	0.117	0.001	4.532	16.507	19.545	22.610	24.000	31.541
28	0.361	0.866	1.000	0.119	0.001	7.827	20.778	23.851	16.601	24.000	28.839
29	0.376	0.863	1.000	0.141	0.001	4.670	16.368	22.667	19.522	24.000	29.125

30	0.367	0.861	1.000	0.140	0.001	5.623	19.915	23.888	15.994	24.000	28.140
31	0.363	0.872	1.000	0.123	0.001	7.079	22.320	16.096	19.230	24.000	31.959
32	0.362	0.875	1.000	0.118	0.001	9.513	16.879	20.361	23.990	24.000	33.118
33	0.359	0.876	1.000	0.088	0.001	1.882	14.359	22.159	18.852	24.000	27.415
34	0.347	0.881	1.000	0.139	0.001	3.119	14.489	23.372	17.514	24.000	27.671
35	0.338	0.869	1.000	0.157	0.001	4.349	21.342	13.821	16.821	24.000	31.312
36	0.749	0.867	1.000	0.168	0.001	23.386	10.808	14.263	18.029	24.000	27.769
37	0.356	0.868	1.000	0.158	0.001	6.257	20.633	17.626	23.633	24.000	32.431
38	0.335	0.349	1.000	0.148	0.001	4.611	0.001	17.552	21.499	24.000	18.186
39	0.355	0.874	1.000	1.000	0.001	9.092	13.226	23.121	20.104	24.000	6.903
40	0.736	0.893	1.000	0.151	0.001	7.013	23.875	20.381	15.571	24.000	25.039
41	0.409	0.849	0.001	0.143	0.001	6.454	23.999	24.000	19.435	24.000	12.010
42	0.365	0.890	1.000	0.148	0.001	10.222	19.051	22.051	16.051	24.000	32.947
43	0.332	0.916	1.000	0.165	0.001	5.092	15.896	18.897	23.204	24.000	33.737
44	0.297	1.000	1.000	0.142	0.001	3.301	22.324	19.122	15.654	24.000	32.963
45	0.286	1.000	1.000	0.156	0.001	10.539	14.474	19.447	22.539	24.000	36.569
46	0.229	1.000	0.496	0.010	0.183	4.194	12.737	23.996	24.000	0.955	19.273
47	0.250	1.000	1.000	0.001	0.100	5.751	15.606	22.424	24.000	1.850	31.086
48	0.001	1.000	1.000	0.219	0.141	24.000	21.087	17.297	14.213	0.171	35.402

Case 3

Step 1

Initial design

	X1	X2	X3	X4	X5	X6	X7	X8	T1	T2	T3	T4	T5	T6	T7	T8	y
1	0.7	0.6	0.0	0.9	0.4	0.1	1.0	0.3	23.5	12.2	11.5	6.4	11.6	0.0	17.2	21.2	22.0
2	0.1	0.4	0.7	1.0	0.9	0.6	0.3	0.0	1.0	4.2	0.0	14.3	5.6	10.4	17.1	16.5	1.8
3	0.0	0.1	0.3	0.4	0.6	0.7	0.9	1.0	0.0	22.8	1.9	16.4	14.8	22.6	6.5	0.2	16.8
4	0.4	1.0	0.6	0.0	0.3	0.9	0.7	0.1	21.7	9.2	4.4	13.5	0.0	15.2	0.3	16.8	19.5
5	0.6	0.9	0.1	0.3	1.0	0.4	0.0	0.7	18.6	6.7	22.5	4.9	13.8	0.0	6.1	4.7	10.9
6	0.3	0.7	1.0	0.6	0.1	0.0	0.4	0.9	7.2	11.3	19.4	12.0	9.9	6.7	23.3	0.0	29.0
7	1.0	0.0	0.9	0.1	0.7	0.3	0.6	0.4	14.2	0.0	3.0	4.9	5.9	14.6	1.8	14.4	10.3
8	0.9	0.3	0.4	0.7	0.0	1.0	0.1	0.6	0.0	9.7	23.1	9.1	23.0	17.9	2.1	7.3	16.7

Sequential Output

	X1	X2	X3	X4	X5	X6	X7	X8	T1	T2	T3	T4	T5	T6	T7	T8	y
1	0.0	0.0	0.1	0.0	0.0	0.2	0.0	0.8	24.0	24.0	13.3	24.0	24.0	17.6	24.0	0.8	3.4
2	0.8	0.0	0.0	0.6	0.2	0.0	0.0	0.0	0.6	24.0	24.0	11.2	14.2	24.0	24.0	24.0	10.3
3	0.0	0.8	0.0	0.5	0.0	0.0	0.0	0.0	24.0	22.8	24.0	15.7	24.0	24.0	24.0	24.0	13.7
4	0.0	0.8	0.0	0.1	0.0	0.0	0.0	0.0	24.0	9.1	24.0	1.7	24.0	24.0	24.0	24.0	7.9
5	0.0	0.8	0.0	0.0	0.0	0.0	0.0	0.0	24.0	20.8	24.0	24.0	24.0	24.0	24.0	24.0	8.3
6	0.0	0.9	0.0	0.5	0.2	0.0	0.6	0.0	24.0	7.6	24.0	0.1	3.3	24.0	11.9	24.0	5.4
7	0.5	0.7	0.4	0.0	0.0	0.0	0.0	0.4	14.6	4.2	0.1	24.0	24.0	24.0	24.0	22.0	8.6
8	0.8	0.8	1.0	0.0	0.0	0.0	0.0	0.0	15.8	18.8	10.7	24.0	24.0	24.0	24.0	24.0	20.5
9	0.9	0.7	1.0	0.0	0.0	0.0	0.0	0.0	1.3	16.9	12.4	24.0	24.0	24.0	24.0	24.0	12.1
10	0.8	0.8	1.0	0.5	0.0	0.9	0.0	0.2	13.7	19.9	10.7	22.9	1.1	16.8	24.0	7.2	19.6
11	0.4	0.0	1.0	0.5	0.3	0.0	0.8	0.0	0.7	24.0	15.3	23.0	18.4	24.0	5.0	24.0	17.9
12	0.0	0.0	1.0	0.6	0.0	0.0	0.9	0.0	24.0	24.0	14.2	23.3	24.0	24.0	17.6	24.0	16.9
13	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	24.0	24.0	13.3	24.0	24.0	24.0	24.0	24.0	15.8
14	0.0	0.0	1.0	0.4	0.0	0.0	0.0	0.0	24.0	24.0	13.8	17.3	24.0	24.0	24.0	24.0	19.3
15	0.0	0.0	1.0	0.0	0.0	0.1	0.0	0.0	24.0	24.0	20.3	24.0	24.0	5.1	24.0	24.0	14.7
16	0.0	0.0	1.0	0.5	0.0	0.0	0.0	0.3	24.0	24.0	17.6	13.9	24.0	24.0	24.0	21.9	18.4
17	0.0	0.6	0.5	0.5	0.5	0.1	1.0	0.9	24.0	13.1	20.8	17.3	10.1	1.5	23.9	5.5	26.7
18	0.8	0.6	0.5	0.5	0.5	0.0	0.5	0.0	23.5	15.7	12.6	8.6	0.4	4.8	20.5	24.0	23.9
19	0.0	0.6	1.0	0.4	0.4	0.0	0.0	0.4	24.0	23.5	16.1	12.7	4.5	0.8	24.0	19.4	27.2
20	0.0	0.0	1.0	0.3	0.4	0.0	0.0	0.0	24.0	24.0	12.7	22.3	6.9	24.0	24.0	24.0	21.0
21	0.0	0.0	0.0	0.4	0.4	0.0	0.8	0.0	24.0	24.0	24.0	21.2	11.5	24.0	0.4	24.0	9.9
22	0.0	0.6	0.5	0.5	0.4	0.2	0.3	0.4	24.0	23.8	8.8	18.8	4.7	0.0	14.9	11.8	20.8
23	0.6	0.2	1.0	0.5	0.4	1.0	0.0	1.0	23.4	5.2	20.4	11.9	8.4	16.8	24.0	1.9	18.4
24	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	0.1
25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	0.1
26	0.5	0.8	0.0	0.5	0.4	0.0	0.0	0.7	21.8	16.5	24.0	13.5	6.3	24.0	24.0	9.3	20.9
27	0.4	0.9	1.0	0.7	1.0	0.1	1.0	0.0	0.4	22.1	18.5	8.0	14.7	3.6	11.6	24.0	22.7
28	0.0	0.9	1.0	0.4	0.4	0.0	1.0	0.7	24.0	23.0	7.4	13.4	10.4	24.0	16.4	19.5	29.6

29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	0.1
30	0.5	0.9	1.0	0.7	0.4	0.0	0.1	0.8	19.1	22.1	13.9	0.3	10.3	24.0	3.3	7.2	17.5
31	0.0	1.0	1.0	0.4	0.3	0.0	0.0	0.9	24.0	16.3	23.3	10.8	4.7	24.0	24.0	19.9	31.2
32	0.8	0.6	1.0	0.8	1.0	0.0	1.0	0.9	17.8	23.9	7.9	10.9	20.9	24.0	1.7	13.9	9.6
33	0.0	1.0	1.0	0.3	0.4	0.1	0.0	0.7	24.0	22.5	13.2	5.1	1.9	8.8	24.0	16.5	30.3
34	0.8	1.0	1.0	0.4	0.3	0.0	1.0	0.6	19.7	16.4	13.3	10.3	4.2	24.0	23.3	7.2	29.2
35	0.0	1.0	1.0	0.4	0.4	0.0	0.0	0.0	24.0	23.6	19.3	10.0	16.3	24.0	24.0	24.0	34.0
36	0.0	1.0	0.4	0.6	0.5	0.0	0.0	0.8	24.0	11.2	23.2	15.1	5.4	24.0	24.0	0.4	20.6
37	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	0.1
38	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	0.1
39	0.4	0.8	0.0	0.3	0.3	0.0	0.0	0.6	16.8	23.7	24.0	9.7	6.7	24.0	24.0	13.8	17.0
40	0.0	0.9	0.5	0.4	0.5	0.0	0.0	0.9	24.0	22.3	3.1	9.1	13.3	24.0	24.0	17.1	17.5
41	0.0	1.0	1.0	0.0	0.4	0.0	0.6	0.7	24.0	15.2	11.8	24.0	6.1	24.0	23.1	20.1	36.6
42	0.0	1.0	0.1	0.5	0.4	0.0	0.4	0.0	24.0	10.4	4.0	17.2	23.1	24.0	13.5	24.0	18.3
43	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	0.1
44	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	0.1
45	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	24.0	0.1

Step 2

Initial design

	X2	X3	X4	X5	T2	T3	T4	T5	y
1	0.667	0.000	1.000	0.333	0.000	16.313	17.177	14.481	10.482
2	0.000	0.333	0.667	1.000	15.772	6.619	0.000	15.071	5.986
3	0.333	1.000	0.000	0.667	0.000	1.181	3.769	3.002	3.669
4	1.000	0.667	0.333	0.000	22.235	19.022	0.000	1.515	20.708

Sequential Output

	X2	X3	X4	X5	T2	T3	T4	T5	y
1	1.000	0.654	0.361	0.001	22.955	19.738	6.819	24.000	24.229
2	1.000	0.003	0.980	0.263	24.000	24.000	17.165	20.741	10.432
3	0.692	0.595	0.406	0.001	0.535	21.673	14.739	24.000	14.076
4	0.690	0.828	0.384	0.001	11.091	15.578	2.289	24.000	22.514
5	1.000	0.001	0.352	0.001	0.032	24.000	3.260	24.000	-2.038
6	0.788	0.985	0.411	0.282	22.836	10.269	3.175	16.674	24.088
7	0.929	1.000	1.000	0.428	23.925	9.349	18.370	13.149	8.474
8	0.756	0.693	0.429	0.391	23.964	17.620	4.614	20.813	23.005
9	0.675	0.802	1.000	0.161	23.457	19.363	3.678	16.347	11.741
10	0.531	0.854	0.496	0.381	1.177	14.918	6.181	21.732	18.594
11	0.372	0.547	0.403	0.116	9.645	15.746	4.279	23.261	17.221
12	0.851	0.782	0.362	1.000	15.634	23.256	20.251	6.139	24.305
13	0.838	0.778	0.323	1.000	22.947	16.914	10.051	2.033	24.210
14	0.833	0.713	0.365	1.000	14.742	22.851	4.944	1.743	17.620
15	0.918	0.816	0.140	0.317	12.257	22.557	18.906	15.792	28.639
16	0.930	0.834	0.271	0.454	15.363	12.259	22.445	18.910	33.291
17	0.001	0.848	0.263	0.496	24.000	23.672	16.022	19.384	18.751
18	0.901	0.907	0.273	0.393	23.329	17.004	20.328	1.651	28.888
19	0.877	0.882	0.247	0.749	14.824	11.824	22.541	18.915	33.584
20	0.897	1.000	0.243	0.630	13.528	22.530	19.530	16.530	33.413
21	0.891	0.886	0.255	0.697	22.814	11.872	18.208	14.935	32.091
22	0.896	1.000	0.249	1.000	14.975	11.975	22.297	18.562	36.461
23	0.884	1.000	0.041	1.000	13.733	19.866	16.740	23.770	34.977
24	0.917	1.000	0.226	1.000	14.458	23.614	20.599	17.599	31.661
25	0.867	1.000	0.163	0.881	23.994	14.897	20.967	17.967	34.567
26	0.855	0.971	0.212	1.000	13.878	16.886	19.898	23.959	36.086
27	0.837	0.912	0.206	1.000	23.998	12.937	18.943	15.943	32.594
28	0.831	0.576	0.188	0.967	15.621	0.689	22.740	19.738	13.138
29	1.000	0.987	0.001	0.871	20.197	15.693	24.000	23.328	33.079
30	0.001	1.000	0.222	0.947	24.000	14.668	23.072	19.776	29.712

31	1.000	1.000	0.168	0.695	17.917	14.894	23.510	2.988	35.607
32	0.955	1.000	0.190	0.731	15.121	18.387	22.169	10.190	37.390
33	1.000	1.000	0.192	0.746	15.701	19.937	23.072	9.295	36.232
34	1.000	1.000	0.180	0.725	23.511	15.995	20.397	11.356	36.128
35	1.000	1.000	0.189	0.749	18.626	14.761	22.699	9.594	38.494
36	1.000	1.000	0.192	0.749	16.479	22.480	19.480	11.703	34.555
37	0.001	1.000	0.181	0.745	24.000	16.584	21.990	13.479	27.494
38	0.001	1.000	0.177	0.714	24.000	16.728	23.226	2.661	21.247
39	1.000	0.001	0.224	0.865	13.463	24.000	16.634	22.679	12.560
40	1.000	1.000	0.745	0.759	14.738	23.031	19.954	9.703	17.586
41	0.152	1.000	0.173	0.485	5.194	18.504	22.159	14.199	24.303
42	1.000	1.000	0.174	0.312	14.107	11.107	18.989	21.990	37.292
43	0.978	0.001	0.135	0.291	17.648	24.000	20.903	23.941	14.255
44	1.000	0.081	0.211	0.840	20.125	4.095	16.649	13.589	13.415
45	1.000	1.000	0.221	1.000	19.188	15.770	22.707	12.423	38.331

Appendix B

R code

Case 1

```
    # initial design T
k = 5
T1 = matrix(runif(k**2,0,24), nrow=k)

# neighbor matrix
neighbour1 = function(xc){
  xn = xc
  ra = sample(k,1)
  r = sample.int(k,2)
  xn[ra,r[1]]=xc[ra,r[2]]
  xn[ra,r[2]]=xc[ra,r[1]]
  return(xn)
}

## vp function
p = 15
rho1 = 0.2
rho2 = 0.8
OF1 = function(xx){
  t = matrix(rep(1,k**2), nrow=k)
  i = 1
  while (i < k+1){
    j = 1
    while (j < k+1){
      l = 1
      while (l < k+1){
        if (xx[l,i]<xx[l,j]){t[i,j]=t[i,j]+1}
```

```
        l = l+1
      }
      j = j+1
    }
    i = i+1
  }
  diag(t)=NA
  sum1 = sum(apply(1/t**p, 1, sum, na.rm=TRUE))
  sum2 = 0
  i = 2
  while (i < k+1){
    j = 1
    while (j < i){
      sum2 = sum2 + 1/(sqrt(sum((xx[i,]-xx[j,])^2))+1)**p
      j = j+1
    }
    i = i+1
  }
  vp = (rho1*sum1+rho2*sum2)**(1/p)
  return(vp)
}

algo1 = list(nS = 3000L,
             vT = 0,
             neighbour = neighbour1,
             x0 = T1,
             printBar = FALSE)

library(NMOF)
sol1 = TAopt(OF1, algo = algo1)
T_int = sol1$xbest


# initial design X
library(LHD)
X0 = FastMmLHD(5,5)

## cp function
p = 15
rho3 = 0.5
rho4 = 0.5
OF2 = function(xx){
```

```
  sum3 = 0
  i = 2
  while (i < k+1){
    j = 1
    while (j < i){
      sum3 = sum3 + 1/(rho3*sqrt(sum((xx[i,]-xx[j,])^2))+
      rho4*sqrt(sum((T_int[i,]-T_int[j,])^2))+1)**p
      j = j+1
    }
    i = i+1
  }
  cp = sum3**(1/p)
  return(cp)
}

# neighbor matrix
neighbour2 = function(xc){
  xn = xc
  r = sample.int(5,2)
  xn[r[1],]=xc[r[2],]
  xn[r[2],]=xc[r[1],]
  return(xn)
}

algo2 = list(nS = 3000L,
             vT = 0,
             neighbour = neighbour2,
             x0 = X0,
             printBar = FALSE)

sol2 = TAopt(OF2, algo = algo2)

# rescale to [0,1]
fn = function(x){x/4}
X_int = apply(sol2$xbest,2,fn)


# coefficient and interaction
g = c(26.8,23.2,32.4,30,-15)
s1 = c(0,0,0,0,0)
s2 = c(-17.4,0,0,0,0)
s3 = c(-19.8,8.1,0,0,0)
```

```
s4 = c(6.9,-16.8,-28.4,0,0)
s5 = c(-12.4,-13.2,7.2,-11.6,0)
s = cbind(s1,s2,s3,s4,s5)
y1 = rep(0,k)
y2 = rep(0,k)

# initial response
y1 = (log(1+X_int)*(2.4*(30-T_int)/24-2*((30-T_int)/24)**2))%*%g
u = (1:5)
v = t(combn(u,2))
l = 1
while (l < (nrow(v)+1)){
  y2 = y2 + s[v[l,1],v[l,2]]*(1/(1+abs(T_int[,v[l,1]]-
  T_int[,v[l,2]])/24))*(X_int[,v[l,1]]**2)*(X_int[,v[l,2]]**2)
  l = l+1
}
y = -y1 - y2


# GP model estimation
EI = NA
for (N in (1:50)){

  library(stats)
  x0 = c(rep(50,k),rep(10,k),rep(5,k),rep(10,k),10)
  Lb = c(rep(0.1,k),rep(0.1,k),rep(0.1,k),rep(0,k),0.1)
  Ub = c(rep(150,k),rep(100,k),rep(10,k),rep(24,k),100)

  # upper matrix index
  rcoord <- cbind(rep(seq_len(k+N-2), times = rev(seq_len(k+N-2))),
    unlist(lapply(
      X = rev(seq_len(k+N-2)),
      FUN = function(nn, nm) seq_len(nn) + nm - nn, nm = k+N-1)))

  # covariance matrix
  library('nloptr')
  kern =function(xx){
    Phi = matrix(rep(0,(k+N-1)^2), nrow=k+N-1)

    sig2 = xx[1:k]
    theta = xx[(k+1):(2*k)]
    eta = xx[(2*k+1):(3*k)]
```

```
  nu = xx[(3*k+1):(4*k)]
  tau2 = xx[4*k+1]

  m = 1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),
  2,eta,'*'))) - 1/(1+exp(-sweep(sweep(T_int[rcoord[, 2],],2,nu),
  2,eta,'*')))

  n = sweep(exp(-sweep((X_int[rcoord[, 1],]-X_int[rcoord[, 2],])
  **2,2,theta,'*')),2,sig2,'*')

  Phi[rcoord] = apply(exp(-m**2)*n, 1, sum)
  Phi = Phi + t(Phi)
  diag(Phi) = sum(sig2) + tau2

  return (Phi)
}

# MLE estimation and gradient
eval_f_list = function(xx) {

  # MLE estimation
  fy1 = t(y)%*%solve(kern(xx))%*%y
  fy2 = 1/(t(rep(1,k+N-1))%*%solve(kern(xx))%*%rep(1,k+N-1))*
  (t(rep(1,k+N-1))%*%solve(kern(xx))%*%y)**2
  fx = log(det(kern(xx))) + fy1 - fy2

  sig2 = xx[1:k]
  theta = xx[(k+1):(2*k)]
  eta = xx[(2*k+1):(3*k)]
  nu = xx[(3*k+1):(4*k)]
  tau2 = xx[4*k+1]

  grad = rep(0,4*k+1)
  p_si = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_th = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_et = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_nu = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_ta = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)

  h = 1
  while (h < (k+1)){
    m = 1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),
```

```
2,eta,'*'))) - 1/(1+exp(-sweep(sweep(T_int[rcoord[, 2],],
2,nu), 2,eta,'*')))

n = (X_int[rcoord[, 1],]-X_int[rcoord[, 2],])**2

p_si[rcoord] = (exp(-m**2)*exp(-sweep(n,2,theta,'*')))[,h]
p_si = p_si + t(p_si)
diag(p_si)=1

p_th[rcoord] = -(sweep(n,2,sig2,'*')*
exp(-sweep(n,2,theta,'*'))*exp(-m**2))[,h]
p_th = p_th + t(p_th)

# part of gradient calculation
r = sweep(T_int[rcoord[, 1],],2,nu)*exp(-sweep(sweep
(T_int[rcoord[, 1],],2,nu), 2,eta,'*'))

s = sweep(T_int[rcoord[, 2],],2,nu)*exp(-sweep(sweep
(T_int[rcoord[, 2],],2,nu), 2,eta,'*'))

rr = (1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],]
,2,nu), 2,eta,'*'))))**2*r - (1/(1+exp(-sweep(sweep(
T_int[rcoord[, 2],],2,nu), 2,eta,'*'))))**2*s

p_et[rcoord] = (-2*sweep(m,2,sig2,'*')*rr*exp
(-sweep(n,2,theta,'*'))*exp(-m**2))[,h]
p_et = p_et + t(p_et)

ss = (1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),
2,eta,'*'))))**2*(exp(-sweep(sweep(T_int[rcoord[, 1],],
2,nu),2,eta,'*'))) - (1/(1+exp(-sweep(sweep
(T_int[rcoord[, 2],],2,nu),2,eta,'*'))))**2*(exp
(-sweep(sweep(T_int[rcoord[, 2],],2,nu),2,eta,'*')))

p_nu[rcoord] = (2*sweep(m,2,(sig2*eta),'*')*ss*exp
(-sweep(n,2,theta,'*'))*exp(-m**2))[,h]
p_nu = p_nu + t(p_nu)

# prediction mean
mu_hat = as.vector(1/(t(rep(1,k+N-1))%*%solve
(kern(xx))%*%rep(1,k+N-1))*t(rep(1,k+N-1))%*%
solve(kern(xx))%*%y)
```

```
    # gradient of sig2
    grad[h] = sum(diag((solve(kern(xx))%*%p_si)))
    -t(y-mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_si%*%
    solve(kern(xx))
    %*%(y-mu_hat*rep(1,k+N-1))

    # gradient of theta
    grad[h+k] = sum(diag((solve(kern(xx))%*%p_th)))-
    t(y-mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_th%*%
    solve(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))

    # gradient of eta
    grad[h+2*k] = sum(diag((solve(kern(xx))%*%p_et)))
    -t(y-mu_hat*rep(1,k+N-1))%*%solve(kern(xx))
    %*%p_et%*%solve(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))

    # gradient of nu
    grad[h+3*k] = sum(diag((solve(kern(xx))%*%p_nu)))-t(y-mu_hat*
    rep(1,k+N-1))%*%solve(kern(xx))%*%p_nu%*%solve(kern(xx))
    %*%(y-mu_hat*rep(1,k+N-1))

    h = h+1
  }

  diag(p_ta) = 2*tau2

  # gradient of tau2
  grad[4*k+1] = sum(diag((solve(kern(xx))%*%p_ta)))-t(y-
  mu_hat*rep(1,k+N-1))%*%   solve(kern(xx))%*%p_ta%*%solve(kern(xx))
  %*%(y-mu_hat*rep(1,k+N-1))

  return( list( "objective" = fx,
                "gradient"  = grad ) )
}

# optim parameters
opts = list("algorithm"="NLOPT_LD_LBFGS",
            "xtol_rel"=1.0e-6)
res = nloptr(x0 = x0,
             eval_f = eval_f_list,
             lb = Lb,
```

```
              ub = Ub,
              opts = opts)

# parameter estimated values
sig2 = res$solution[1:k]
theta = res$solution[(k+1):(2*k)]
eta = res$solution[(2*k+1):(3*k)]
nu = res$solution[(3*k+1):(4*k)]
tau2 = res$solution[4*k+1]

# cov matrix and mean
Phi = kern(res$solution)
mu = as.vector(1/(t(rep(1,k+N-1))%*%solve(Phi)%*%rep(1,k+N-1))
*t(rep(1,k+N-1))%*%solve(Phi)%*%y)



# EI function
fw = function(w){
  m = sweep(1/(1+exp(-sweep(sweep(T_int,2,nu), 2,eta,'*'))),2,
  1/(1+exp(-eta*(w[(k+1):(2*k)]-nu))))
  n = exp(-sweep((sweep(X_int,2,w[1:k]))**2,2,theta,'*'))
  gama = apply(sweep(n,2,sig2,'*')*exp(-m**2),1,sum)

  # prediction mean
  y_hat = as.vector(mu + t(gama)%*%solve(Phi)%*%(y-
  mu*rep(1,(k+N-1))))

  # covariance
  sw = sum(sig2)+tau2-t(gama)%*%solve(Phi)%*%gama+(1-
  t(rep(1,(k+N-1)))%*%solve(Phi)%*%gama)
  **2/(t(rep(1,(k+N-1)))%*%
  solve(Phi)%*%rep(1,(k+N-1)))
  sw = sqrt(as.vector(sw))

  # EI value
  EI = (min(y)-y_hat)*pnorm((min(y)-y_hat)/sw,mean=0,sd=1)+
  sw*dnorm((min(y)-y_hat)/sw,mean=0,sd=1)
  return(EI)
}

# EI gradient
fw_grad = function(w){
```

```
m = sweep(1/(1+exp(-sweep(sweep(T_int,2,nu), 2,eta,'*'))),2,
1/(1+exp(-eta*(w[(k+1):(2*k)]-nu))))
n = exp(-sweep((sweep(X_int,2,w[1:k]))**2,2,theta,'*'))
gama = apply(sweep(n,2,sig2,'*')*exp(-m**2),1,sum)

# prediction mean
y_hat = as.vector(mu + t(gama)%*%solve(Phi)%*%(y-
mu*rep(1,(k+N-1))))

# covariance
sw = sum(sig2)+tau2-t(gama)%*%solve(Phi)%*%gama+
(1-t(rep(1,(k+N-1)))
%*%solve(Phi)%*%gama)**2/(t(rep(1,(k+N-1)))%*%solve(Phi)%*%
rep(1,(k+N-1)))
sw = sqrt(as.vector(sw))

grad_w = rep(0,2*k)
grad_sw = rep(0,2*k)
p_wx = t(2*sweep(sweep(X_int,2,w[1:k]),2,theta*sig2)*n*exp(-m**2))
r = ((1/(1+exp(-eta*(w[(k+1):(2*k)]-nu))))**2)*
exp(-eta*(w[(k+1):(2*k)]-nu))
p_wt = t(2*sweep(m,2,(sig2*eta*r),'*')*n*exp(-m**2))

h = 1
while (h < (k+1)){
  # part of X gradient
  grad_sw[h] = -(1/sw)*(p_wx[h,]%*%solve(Phi)%*%gama+(1-
  t(rep(1,(k+N-1)))%*%solve(Phi)%*%gama)*(p_wx[h,]%*%solve(Phi)
  %*%rep(1,(k+N-1)))/(t(rep(1,(k+N-1)))%*%solve(Phi)
  %*%rep(1,(k+N-1))))

  # part of T gradient
  grad_sw[h+k] = -(1/sw)*(p_wt[h,]%*%solve(Phi)%*%gama+(1-
  t(rep(1,(k+N-1)))%*%solve(Phi)%*%gama)*(p_wt[h,]%*%solve(Phi)
  %*%rep(1,(k+N-1)))/(t(rep(1,(k+N-1)))%*%solve(Phi)
  %*%rep(1,(k+N-1))))

  # gradient of X
  grad_w[h] = -p_wx[h,]%*%Phi%*%(y-mu*rep(1,(k+N-1)))*pnorm(
  (min(y)-y_hat)/sw,mean=0,sd=1) + grad_sw[h]*dnorm(
  (min(y)-y_hat)/sw,mean=0,sd=1)
```

```
    # gradient of T
    grad_w[h+k] = -p_wt[h,]%*%Phi%*%(y-mu*rep(1,(k+N-1)))*pnorm(
    (min(y)-y_hat)/sw,mean=0,sd=1) + grad_sw[h+k]*dnorm(
    (min(y)-y_hat)/sw,mean=0,sd=1)

    h = h+1
  }
  return (grad_w)
}

# optim EI
library(rgenoud)
Lb = c(rep(0.001,k),rep(0.001,k))
Ub = c(rep(1,k),rep(24,k))
dom = cbind(Lb, Ub)
res= genoud(fw, nvars=2*k, Domains = dom, boundary.enforcement=1,
pop.size=1000, max.generations=50, max=TRUE, solution.tolerance=0.01,
gr=fw_grad)

# EI value and w matrix
w_opt = res$par
if (N==1){w = matrix(w_opt,c(1,2*k))}
else {w = rbind(w,w_opt)}
EI[N] = res$value

# X matrix and T matrix
X_int = rbind(X_int, w_opt[1:k])
T_int = rbind(T_int,w_opt[(k+1):(2*k)])

# new response value
y2 = 0
u = (1:k)
v = t(combn(u,2))
l = 1
while (l < nrow(v)+1){
  y2 = y2 + s[v[l,1],v[l,2]]*(1/(1+abs(w_opt[(k+1):(2*k)]
  [v[l,1]]-w_opt[(k+1):(2*k)][v[l,2]])/24))*(w_opt[v[l,1]]
  **2)*(w_opt[v[l,2]]**2)
  l = l+1
}
wt_opt = (30-w_opt[(k+1):(2*k)])/24
y1 = sum(g*log(1+w_opt[1:k])*(2.4*wt_opt-2*wt_opt**2))
```

```
  y_new = -y1 - y2
  y = c(y,y_new)


  # stopping criterion
  if (N >= 3){
    ymax = max(-y[1:(N+k-2)])
    if ((EI[N]<0.015*ymax)&(EI[N-1]<0.015*ymax)&
    (EI[N-2]<0.015*ymax)){ break }
  }
}
```

Case 2

```
# initial design T
k = 5
T1 = matrix(runif(k**2,0,24), nrow=k)

# neighbor matrix
neighbour1 = function(xc){
  xn = xc
  ra = sample(k,1)
  r = sample.int(k,2)
  xn[ra,r[1]]=xc[ra,r[2]]
  xn[ra,r[2]]=xc[ra,r[1]]
  return(xn)
}

## vp function
p = 15
rho1 = 0.2
rho2 = 0.8
OF1 = function(xx){
  t = matrix(rep(1,k**2), nrow=k)
  i = 1
  while (i < k+1){
    j = 1
    while (j < k+1){
      l = 1
      while (l < k+1){
        if (xx[l,i]<xx[l,j]){t[i,j]=t[i,j]+1}
        l = l+1
      }
      j = j+1
    }
    i = i+1
  }
  diag(t)=NA
  sum1 = sum(apply(1/t**p, 1, sum, na.rm=TRUE))
  sum2 = 0
  i = 2
  while (i < k+1){
    j = 1
    while (j < i){
```

```
      sum2 = sum2 + 1/(sqrt(sum((xx[i,]-xx[j,])^2))+1)**p
      j = j+1
    }
    i = i+1
  }
  vp = (rho1*sum1+rho2*sum2)**(1/p)
  return(vp)
}

algo1 = list(nS = 3000L,
             vT = 0,
             neighbour = neighbour1,
             x0 = T1,
             printBar = FALSE)

library(NMOF)
sol1 = TAopt(OF1, algo = algo1)
T_int = sol1$xbest


# initial design X
library(LHD)
X0 = FastMmLHD(5,5)

## cp function
p = 15
rho3 = 0.5
rho4 = 0.5
OF2 = function(xx){
  sum3 = 0
  i = 2
  while (i < k+1){
    j = 1
    while (j < i){
      sum3 = sum3 + 1/(rho3*sqrt(sum((xx[i,]-xx[j,])^2))
      +rho4*sqrt(sum((T_int[i,]-T_int[j,])^2))+1)**p
      j = j+1
    }
    i = i+1
  }
  cp = sum3**(1/p)
  return(cp)
```

```
}

# neighbor matrix
neighbour2 = function(xc){
  xn = xc
  r = sample.int(5,2)
  xn[r[1],]=xc[r[2],]
  xn[r[2],]=xc[r[1],]
  return(xn)
}

algo2 = list(nS = 3000L,
             vT = 0,
             neighbour = neighbour2,
             x0 = X0,
             printBar = FALSE)

sol2 = TAopt(OF2, algo = algo2)

# rescale to [0,1]
fn = function(x){x/4}
X_int = apply(sol2$xbest,2,fn)


# coefficient and interaction
k = 5
g = c(26.8,23.2,32.4,30,-15)
s1 = c(0,0,0,0,0)
s2 = c(-17.4,0,0,0,0)
s3 = c(-19.8,8.1,0,0,0)
s4 = c(6.9,-16.8,-28.4,0,0)
s5 = c(-12.4,-13.2,7.2,-11.6,0)
s = cbind(s1,s2,s3,s4,s5)
y1 = rep(0,k)
y2 = rep(0,k)

# initial response
y1 = (log(1+X_int)*(2.4*(30-T_int)/24-2*((30-T_int)/24)**2))%*%g
u = (1:5)
v = t(combn(u,2))
l = 1
while (l < (nrow(v)+1)){
```

```
  y2 = y2 + s[v[l,1],v[l,2]]*(1/(1+abs(T_int[,v[l,1]]-
  T_int[,v[l,2]])/24))*(X_int[,v[l,1]]**2)*(X_int[,v[l,2]]**2)
  l = l+1
}
y = -y1 - y2


# GP model estimation
EI = NA
for (N in (1:70)){

  library(stats)
  x0 = c(rep(50,k),rep(10,k),rep(5,k),rep(10,k),10)
  Lb = c(rep(0.1,k),rep(0.1,k),rep(0.1,k),rep(0,k),0.1)
  Ub = c(rep(150,k),rep(100,k),rep(10,k),rep(24,k),100)

  # upper matrix index
  rcoord <- cbind(rep(seq_len(k+N-2), times = rev(
  seq_len(k+N-2))), unlist(lapply(X = rev(seq_len(k+N-2)),
  FUN = function(nn, nm) seq_len(nn) + nm - nn, nm = k+N-1)))

  # covariance function
  library('nloptr')
  kern =function(xx){
    Phi = matrix(rep(0,(k+N-1)^2), nrow=k+N-1)

    sig2 = xx[1:k]
    theta = xx[(k+1):(2*k)]
    eta = xx[(2*k+1):(3*k)]
    nu = xx[(3*k+1):(4*k)]
    tau2 = xx[4*k+1]

    m = 1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),
    2,eta,'*'))) - 1/(1+exp(-sweep(sweep(T_int[rcoord[, 2],],
    2,nu), 2,eta,'*')))

    n = sweep(exp(-sweep((X_int[rcoord[, 1],]-X_int
    [rcoord[, 2],])**2,2,theta,'*')),2,sig2,'*')

    Phi[rcoord] = apply(exp(-m**2)*n, 1, sum)
    Phi = Phi + t(Phi)
    diag(Phi) = sum(sig2) + tau2
```

```
  return (Phi)
}

# MLE estimation and gradient
eval_f_list = function(xx) {

  # MLE estimation
  fy1 = t(y)%*%solve(kern(xx))%*%y
  fy2 = 1/(t(rep(1,k+N-1))%*%solve(kern(xx))%*%rep(1,k+N-1))
  *(t(rep(1,k+N-1))%*%solve(kern(xx))%*%y)**2
  fx = log(det(kern(xx))) + fy1 - fy2

  sig2 = xx[1:k]
  theta = xx[(k+1):(2*k)]
  eta = xx[(2*k+1):(3*k)]
  nu = xx[(3*k+1):(4*k)]
  tau2 = xx[4*k+1]

  grad = rep(0,4*k+1)
  p_si = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_th = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_et = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_nu = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_ta = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)

  h = 1
  while (h < (k+1)){
    m = 1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),
    2,eta,'*'))) - 1/(1+exp(-sweep(sweep(T_int[rcoord[, 2],],
    2,nu), 2,eta,'*')))

    n = (X_int[rcoord[, 1],]-X_int[rcoord[, 2],])**2

    p_si[rcoord] = (exp(-m**2)*exp(-sweep(n,2,theta,'*')))[,h]
    p_si = p_si + t(p_si)
    diag(p_si)=1

    p_th[rcoord] = -(sweep(n,2,sig2,'*')*exp(-sweep(n,2,theta,
    '*'))*exp(-m**2))[,h]
    p_th = p_th + t(p_th)
```

```
# part of gradient calculation
r = sweep(T_int[rcoord[, 1],],2,nu)*exp(-sweep(sweep(T_int
[rcoord[, 1],],2,nu), 2,eta,'*'))

s = sweep(T_int[rcoord[, 2],],2,nu)*exp(-sweep(sweep(T_int
[rcoord[, 2],],2,nu), 2,eta,'*'))

rr = (1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu), 2,eta,
'*'))))**2*r - (1/(1+exp(-sweep(sweep(T_int[rcoord[, 2],],2,nu),
2,eta,'*'))))**2*s

p_et[rcoord] = (-2*sweep(m,2,sig2,'*')*rr*exp(-sweep(n,2,theta,
'*'))*exp(-m**2))[,h]
p_et = p_et + t(p_et)

ss = (1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),2,eta,
'*'))))**2*(exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),2,eta,
'*'))) - (1/(1+exp(-sweep(sweep(T_int[rcoord[, 2],],2,nu),2,eta,
'*'))))**2*(exp(-sweep(sweep(T_int[rcoord[, 2],],2,nu),2,eta,'*')))
p_nu[rcoord] = (2*sweep(m,2,(sig2*eta),'*')*ss*exp
(-sweep(n,2,theta,'*'))*exp(-m**2))[,h]
p_nu = p_nu + t(p_nu)

# prediction mean
mu_hat = as.vector(1/(t(rep(1,k+N-1))%*%solve(kern(xx))
%*%rep(1,k+N-1))*t(rep(1,k+N-1))%*%solve(kern(xx))%*%y)

# gradient of sig2
grad[h] = sum(diag((solve(kern(xx))%*%p_si)))-t(y-
mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_si%*%solve
(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))

# gradient of theta
grad[h+k] = sum(diag((solve(kern(xx))%*%p_th)))-
t(y-mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_th%*%
solve(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))

# gradient of eta
grad[h+2*k] = sum(diag((solve(kern(xx))%*%p_et)))-t(y-
mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_et%*%solve
(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))
```

```
    # gradient of nu
    grad[h+3*k] = sum(diag((solve(kern(xx))%*%p_nu)))-t(y-
    mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_nu%*%
    solve(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))
    h = h+1
  }
  diag(p_ta) = 2*tau2

  # gradient of tau2
  grad[4*k+1] = sum(diag((solve(kern(xx))%*%p_ta)))-t(y-
  mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_ta%*%
  solve(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))

  return( list( "objective" = fx,
                "gradient"  = grad ) )
}

# optim parameters
opts = list("algorithm"="NLOPT_LD_LBFGS",
            "xtol_rel"=1.0e-6)
res = nloptr(x0 = x0,
             eval_f = eval_f_list,
             lb = Lb,
             ub = Ub,
             opts = opts)

# parameter estimated values
sig2 = res$solution[1:k]
theta = res$solution[(k+1):(2*k)]
eta = res$solution[(2*k+1):(3*k)]
nu = res$solution[(3*k+1):(4*k)]
tau2 = res$solution[4*k+1]

# cov matrix and mean
Phi = kern(res$solution)
mu = as.vector(1/(t(rep(1,k+N-1))%*%solve(Phi)%*%rep(1,k+N-1))
*t(rep(1,k+N-1))%*%solve(Phi)%*%y)


# EI function
fw = function(w){
  m = sweep(1/(1+exp(-sweep(sweep(T_int,2,nu), 2,eta,'*'))),
```

```
2,1/(1+exp(-eta*(w[(k+1):(2*k)]-nu))))
n = exp(-sweep((sweep(X_int,2,w[1:k]))**2,2,theta,'*'))
gama = apply(sweep(n,2,sig2,'*')*exp(-m**2),1,sum)

# prediction mean
y_hat = as.vector(mu + t(gama)%*%solve(Phi)%*%(y-mu*rep(1,
(k+N-1))))

# covariance
sw = sum(sig2)+tau2-t(gama)%*%solve(Phi)%*%gama+(1-t(rep(1,
(k+N-1)))%*%solve(Phi)%*%gama)**2/(t(rep(1,(k+N-1)))%*%solve(Phi)
%*%rep(1,(k+N-1)))
sw = sqrt(as.vector(sw))

# EI value
EI = (min(y)-y_hat)*pnorm((min(y)-y_hat)/sw,mean=0,sd=1)+
sw*dnorm((min(y)-y_hat)/sw,mean=0,sd=1)

#chosen components
if (length(which(w[1:k]>0.01))==5){
  j = 1
  while (j < k){
    l = j+1
    while (l < (k+1)){
      if ((abs(w[j+k]-w[l+k]))<3){EI=EI-1000}
      l = l+1
    }
    j = j+1
  }
} else{
  c0 = which(w[1:k]<0.01)
  j = 1
  while (j < (k+1)){
    if (j==c0) {j = j+1}
    l = 1
    while (l < (k+1)){
      if ((l==c0) | (l==j)){l = l+1}
      if ((abs(w[j+5]-w[l+5]))<3){EI=EI-1000}
      l = l+1
    }
    j = j+1
  }
```

```
  }
  return(EI)
}

# EI gradient
fw_grad = function(w){
  m = sweep(1/(1+exp(-sweep(sweep(T_int,2,nu), 2,eta,
  '*'))),2,1/(1+exp(-eta*(w[(k+1):(2*k)]-nu))))
  n = exp(-sweep((sweep(X_int,2,w[1:k]))**2,2,theta,'*'))
  gama = apply(sweep(n,2,sig2,'*')*exp(-m**2),1,sum)

  # prediction mean
  y_hat = as.vector(mu + t(gama)%*%solve(Phi)%*%(y-mu*
  rep(1,(k+N-1))))

  # covariance
  sw = sum(sig2)+tau2-t(gama)%*%solve(Phi)%*%gama+(1-
  t(rep(1,(k+N-1)))%*%solve(Phi)%*%gama)**2/(t(rep(1,
  (k+N-1)))%*%solve(Phi)%*%rep(1,(k+N-1)))
  sw = sqrt(as.vector(sw))

  grad_w = rep(0,2*k)
  grad_sw = rep(0,2*k)
  p_wx = t(2*sweep(sweep(X_int,2,w[1:k]),2,theta*sig2)
  *n*exp(-m**2))
  r = ((1/(1+exp(-eta*(w[(k+1):(2*k)]-nu))))**2)*exp(-eta*
  (w[(k+1):(2*k)]-nu))
  p_wt = t(2*sweep(m,2,(sig2*eta*r),'*')*n*exp(-m**2))

  h = 1
  while (h < (k+1)){
    # part of X gradient
    grad_sw[h] = -(1/sw)*(p_wx[h,]%*%solve(Phi)%*%gama+
    (1-t(rep(1,(k+N-1)))%*%solve(Phi)%*%gama)*(p_wx[h,]
    %*%solve(Phi)%*%rep(1,(k+N-1)))/(t(rep(1,(k+N-1)))%*%
    solve(Phi)%*%rep(1,(k+N-1))))

    # part of T gradient
    grad_sw[h+k] = -(1/sw)*(p_wt[h,]%*%solve(Phi)%*%gama+
    (1-t(rep(1,(k+N-1)))%*%solve(Phi)%*%gama)*(p_wt[h,]%*%
    solve(Phi)%*%rep(1,(k+N-1)))/(t(rep(1,(k+N-1)))%*%
    solve(Phi)%*%rep(1,(k+N-1))))
```

```
    # gradient of X
    grad_w[h] = -p_wx[h,]%*%Phi%*%(y-mu*rep(1,(k+N-1)))*
    pnorm((min(y)-y_hat)/sw,mean=0,sd=1) + grad_sw[h]*dnorm
    ((min(y)-y_hat)/sw,mean=0,sd=1)

    # gradient of T
    grad_w[h+k] = -p_wt[h,]%*%Phi%*%(y-mu*rep(1,(k+N-1)))*
    pnorm((min(y)-y_hat)/sw,mean=0,sd=1) + grad_sw[h+k]*
    dnorm((min(y)-y_hat)/sw,mean=0,sd=1)
    h = h+1
  }
  return (grad_w)
}

# optim EI
library(rgenoud)
Lb = c(rep(0.001,k),rep(0.001,k))
Ub = c(rep(1,k),rep(24,k))
dom = cbind(Lb, Ub)
res= genoud(fw, nvars=2*k, Domains = dom, boundary.enforcement=1,
pop.size=1000, max.generations=100,max=TRUE,solution.tolerance=
0.01,gr=fw_grad)

# EI value and w matrix
w_opt = res$par
if (N==1){w = matrix(w_opt,c(1,2*k))} else {w = rbind(w,w_opt)}
EI[N] = res$value

# X matrix and T matrix
X_int = rbind(X_int, w_opt[1:k])
T_int = rbind(T_int,w_opt[(k+1):(2*k)])

# new response value
y2 = 0
u = (1:k)
v = t(combn(u,2))
l = 1
while (l < nrow(v)+1){
  y2 = y2 + s[v[l,1],v[l,2]]*(1/(1+abs(w_opt[(k+1):(2*k)][v[l,1]]
  -w_opt[(k+1):(2*k)][v[l,2]])/24))*(w_opt[v[l,1]]**2)*
  (w_opt[v[l,2]]**2)
```

```
    l = l+1
  }
  wt_opt = (30-w_opt[(k+1):(2*k)])/24
  y1 = sum(g*log(1+w_opt[1:k])*(2.4*wt_opt-2*wt_opt**2))
  y_new = -y1 - y2
  y = c(y,y_new)


  # stopping criterion
  if (N >= 3){
    ymax = max(-y[1:(N+k-2)])
    if ((EI[N]<0.015*ymax)&(EI[N-1]<0.015*ymax)&(EI[N-2]
    <0.015*ymax)){ break }
  }
}
```

Case 3: step 1

```
# initial design T
k = 8
T1 = matrix(runif(k**2,0,24), nrow=k)

# neighbor matrix
neighbour1 = function(xc){
  xn = xc
  ra = sample(k,1)
  r = sample.int(k,2)
  xn[ra,r[1]]=xc[ra,r[2]]
  xn[ra,r[2]]=xc[ra,r[1]]
  return(xn)
}

## vp function
p = 15
rho1 = 0.2
rho2 = 0.8
OF1 = function(xx){
  t = matrix(rep(1,k**2), nrow=k)
  i = 1
  while (i < k+1){
    j = 1
    while (j < k+1){
      l = 1
      while (l < k+1){
        if (xx[l,i]<xx[l,j]){t[i,j]=t[i,j]+1}
        l = l+1
      }
      j = j+1
    }
    i = i+1
  }
  diag(t)=NA
  sum1 = sum(apply(1/t**p, 1, sum, na.rm=TRUE))
  sum2 = 0
  i = 2
  while (i < k+1){
    j = 1
    while (j < i){
```

```
      sum2 = sum2 + 1/(sqrt(sum((xx[i,]-xx[j,])^2))+1)**p
      j = j+1
    }
    i = i+1
  }
  vp = (rho1*sum1+rho2*sum2)**(1/p)
  return(vp)
}

algo1 = list(nS = 3000L,
             vT = 0,
             neighbour = neighbour1,
             x0 = T1,
             printBar = FALSE)

library(NMOF)
sol1 = TAopt(OF1, algo = algo1)
T_int = sol1$xbest


# initial design X
library(LHD)
X0 = FastMmLHD(8,8)

## cp function
p = 15
rho3 = 0.5
rho4 = 0.5
OF2 = function(xx){
  sum3 = 0
  i = 2
  while (i < k+1){
    j = 1
    while (j < i){
      sum3 = sum3 + 1/(rho3*sqrt(sum((xx[i,]-xx[j,])^2))
      +rho4*sqrt(sum((T_int[i,]-T_int[j,])^2))+1)**p
      j = j+1
    }
    i = i+1
  }
  cp = sum3**(1/p)
  return(cp)
```

```
}

# neighbor matrix
neighbour2 = function(xc){
  xn = xc
  r = sample.int(k,2)
  xn[r[1],]=xc[r[2],]
  xn[r[2],]=xc[r[1],]
  return(xn)
}

algo2 = list(nS = 3000L,
             vT = 0,
             neighbour = neighbour2,
             x0 = X0,
             printBar = FALSE)

sol2 = TAopt(OF2, algo = algo2)

# rescale to [0,1]
fn = function(x){x/7}
X_int = apply(sol2$xbest,2,fn)


# coefficient and interaction
g = c(26.8,23.2,32.4,30,15,9.6,3.7,0)
s1 = rep(0,8)
s2 = c(-17.4,0,0,0,0,0,0,0)
s3 = c(-19.8,8.1,0,0,0,0,0,0)
s4 = c(6.9,-16.8,-28.4,0,0,0,0,0)
s5 = c(-12.4,-13.2,7.2,-11.6,0,0,0,0)
s6 = c(3.5,-2.8,-7.9,-6.4,-1.6,0,0,0)
s7 = c(1.7,2.5,-1.8,-0.9,1.2,-0.4,0,0)
s8 = rep(0,8)
s = cbind(s1,s2,s3,s4,s5,s6,s7,s8)
y1 = rep(0,k)
y2 = rep(0,k)

# initial response
y1 = (log(1+X_int)*(2.4*(30-T_int)/24-2*((30-T_int)/24)**2))%*%g
u = (1:k)
v = t(combn(u,2))
```

```
l = 1
while (l < (nrow(v)+1)){
  y2 = y2 + s[v[l,1],v[l,2]]*(1/(1+abs(T_int[,v[l,1]]
  -T_int[,v[l,2]])/24))*(X_int[,v[l,1]]**2)*(X_int[,v[l,2]]**2)
  l = l+1
}
y = -y1 - y2


# GP model estimation
EI = NA
for (N in (1:50)){

  library(stats)
  x0 = c(rep(50,k),rep(10,k),rep(5,k),rep(10,k),10)
  Lb = c(rep(0.1,k),rep(0.1,k),rep(0.1,k),rep(0,k),0.1)
  Ub = c(rep(150,k),rep(100,k),rep(10,k),rep(24,k),100)

  # upper matrix index
  rcoord <- cbind(rep(seq_len(k+N-2), times = rev(
  seq_len(k+N-2))), unlist(lapply(X = rev(seq_len(k+N-2)),
  FUN = function(nn, nm) seq_len(nn) + nm - nn, nm = k+N-1)))

  # covariance function
  library('nloptr')
  kern =function(xx){
    Phi = matrix(rep(0,(k+N-1)^2), nrow=k+N-1)

    sig2 = xx[1:k]
    theta = xx[(k+1):(2*k)]
    eta = xx[(2*k+1):(3*k)]
    nu = xx[(3*k+1):(4*k)]
    tau2 = xx[4*k+1]

    m = 1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),
    2,eta,'*'))) - 1/(1+exp(-sweep(sweep(T_int[rcoord[, 2],],
    2,nu), 2,eta,'*')))

    n = sweep(exp(-sweep((X_int[rcoord[, 1],]-X_int
    [rcoord[, 2],])**2,2,theta,'*')),2,sig2,'*')

    Phi[rcoord] = apply(exp(-m**2)*n, 1, sum)
```

```
  Phi = Phi + t(Phi)
  diag(Phi) = sum(sig2) + tau2
  return (Phi)
}

# MLE estimation and gradient
eval_f_list = function(xx) {

  # MLE estimation
  fy1 = t(y)%*%solve(kern(xx))%*%y
  fy2 = 1/(t(rep(1,k+N-1))%*%solve(kern(xx))%*%rep(1,k+N-1))
  *(t(rep(1,k+N-1))%*%solve(kern(xx))%*%y)**2
  fx = log(det(kern(xx))) + fy1 - fy2

  sig2 = xx[1:k]
  theta = xx[(k+1):(2*k)]
  eta = xx[(2*k+1):(3*k)]
  nu = xx[(3*k+1):(4*k)]
  tau2 = xx[4*k+1]

  grad = rep(0,4*k+1)
  p_si = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_th = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_et = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_nu = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_ta = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)

  h = 1
  while (h < (k+1)){
    m = 1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),
    2,eta,'*'))) - 1/(1+exp(-sweep(sweep(T_int[rcoord[, 2],],
    2,nu), 2,eta,'*')))

    n = (X_int[rcoord[, 1],]-X_int[rcoord[, 2],])**2

    p_si[rcoord] = (exp(-m**2)*exp(-sweep(n,2,theta,'*')))[,h]
    p_si = p_si + t(p_si)
    diag(p_si)=1

    p_th[rcoord] = -(sweep(n,2,sig2,'*')*exp(-sweep(n,2,theta,
    '*'))*exp(-m**2))[,h]
    p_th = p_th + t(p_th)
```

```
# part of gradient calculation
r = sweep(T_int[rcoord[, 1],],2,nu)*exp(-sweep(sweep(T_int
[rcoord[, 1],],2,nu), 2,eta,'*'))

s = sweep(T_int[rcoord[, 2],],2,nu)*exp(-sweep(sweep(T_int
[rcoord[, 2],],2,nu), 2,eta,'*'))

rr = (1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu), 2,eta,
'*'))))**2*r - (1/(1+exp(-sweep(sweep(T_int[rcoord[, 2],],2,nu),
2,eta,'*'))))**2*s

p_et[rcoord] = (-2*sweep(m,2,sig2,'*')*rr*exp(-sweep(n,2,theta,
'*'))*exp(-m**2))[,h]
p_et = p_et + t(p_et)

ss = (1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),2,eta,
'*'))))**2*(exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),2,eta,
'*'))) - (1/(1+exp(-sweep(sweep(T_int[rcoord[, 2],],2,nu),2,eta,
'*'))))**2*(exp(-sweep(sweep(T_int[rcoord[, 2],],2,nu),2,eta,'*')))
p_nu[rcoord] = (2*sweep(m,2,(sig2*eta),'*')*ss*exp
(-sweep(n,2,theta,'*'))*exp(-m**2))[,h]
p_nu = p_nu + t(p_nu)

# prediction mean
mu_hat = as.vector(1/(t(rep(1,k+N-1))%*%solve(kern(xx))
%*%rep(1,k+N-1))*t(rep(1,k+N-1))%*%solve(kern(xx))%*%y)

# gradient of sig2
grad[h] = sum(diag((solve(kern(xx))%*%p_si)))-t(y-
mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_si%*%solve
(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))

# gradient of theta
grad[h+k] = sum(diag((solve(kern(xx))%*%p_th)))-
t(y-mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_th%*%
solve(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))

# gradient of eta
grad[h+2*k] = sum(diag((solve(kern(xx))%*%p_et)))-t(y-
mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_et%*%solve
(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))
```

```
    # gradient of nu
    grad[h+3*k] = sum(diag((solve(kern(xx))%*%p_nu)))-t(y-
    mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_nu%*%
    solve(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))
    h = h+1
  }
  diag(p_ta) = 2*tau2

  # gradient of tau2
  grad[4*k+1] = sum(diag((solve(kern(xx))%*%p_ta)))-t(y-
  mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_ta%*%
  solve(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))

  return( list( "objective" = fx,
                "gradient"  = grad ) )
}

# optim parameters
opts = list("algorithm"="NLOPT_LD_LBFGS",
            "xtol_rel"=1.0e-6)
res = nloptr(x0 = x0,
             eval_f = eval_f_list,
             lb = Lb,
             ub = Ub,
             opts = opts)

# parameter estimated values
sig2 = res$solution[1:k]
theta = res$solution[(k+1):(2*k)]
eta = res$solution[(2*k+1):(3*k)]
nu = res$solution[(3*k+1):(4*k)]
tau2 = res$solution[4*k+1]

# cov matrix and mean
Phi = kern(res$solution)
mu = as.vector(1/(t(rep(1,k+N-1))%*%solve(Phi)%*%rep(1,k+N-1))
*t(rep(1,k+N-1))%*%solve(Phi)%*%y)

# EI function
fw = function(w){
  m = sweep(1/(1+exp(-sweep(sweep(T_int,2,nu), 2,eta,'*'))),
```

```
2,1/(1+exp(-eta*(w[(k+1):(2*k)]-nu))))
n = exp(-sweep((sweep(X_int,2,w[1:k]))**2,2,theta,'*'))
gama = apply(sweep(n,2,sig2,'*')*exp(-m**2),1,sum)

# prediction mean
y_hat = as.vector(mu + t(gama)%*%solve(Phi)%*%(y-mu*rep(1,
(k+N-1))))

# covariance
sw = sum(sig2)+tau2-t(gama)%*%solve(Phi)%*%gama+(1-t(rep(1,
(k+N-1)))%*%solve(Phi)%*%gama)**2/(t(rep(1,(k+N-1)))%*%solve(Phi)
%*%rep(1,(k+N-1)))
sw = sqrt(as.vector(sw))

# EI value
EI = (min(y)-y_hat)*pnorm((min(y)-y_hat)/sw,mean=0,sd=1)+
sw*dnorm((min(y)-y_hat)/sw,mean=0,sd=1)

#chosen components
if (length(which(w[1:k]>0.01))==k){
  j = 1
  while (j < k){
    l = j+1
    while (l < (k+1)){
      if ((abs(w[j+k]-w[l+k]))<3){EI=EI-1000}
      l = l+1
    }
    j = j+1
  }
} else{
  c0 = which(w[1:k]<0.01)
  j = 1
  while (j < (k+1)){
    if (j==c0) {j = j+1}
    l = 1
    while (l < (k+1)){
      if ((l==c0) | (l==j)){l = l+1}
      if ((abs(w[j+k]-w[l+k]))<3){EI=EI-1000}
      l = l+1
    }
    j = j+1
  }
```

```
  }
  return(EI)
}

# EI gradient
fw_grad = function(w){
  m = sweep(1/(1+exp(-sweep(sweep(T_int,2,nu), 2,eta,
  '*'))),2,1/(1+exp(-eta*(w[(k+1):(2*k)]-nu))))
  n = exp(-sweep((sweep(X_int,2,w[1:k]))**2,2,theta,'*'))
  gama = apply(sweep(n,2,sig2,'*')*exp(-m**2),1,sum)

  # prediction mean
  y_hat = as.vector(mu + t(gama)%*%solve(Phi)%*%(y-mu*
  rep(1,(k+N-1))))

  # covariance
  sw = sum(sig2)+tau2-t(gama)%*%solve(Phi)%*%gama+(1-
  t(rep(1,(k+N-1)))%*%solve(Phi)%*%gama)**2/(t(rep(1,
  (k+N-1)))%*%solve(Phi)%*%rep(1,(k+N-1)))
  sw = sqrt(as.vector(sw))

  grad_w = rep(0,2*k)
  grad_sw = rep(0,2*k)
  p_wx = t(2*sweep(sweep(X_int,2,w[1:k]),2,theta*sig2)
  *n*exp(-m**2))
  r = ((1/(1+exp(-eta*(w[(k+1):(2*k)]-nu))))**2)*exp(-eta*
  (w[(k+1):(2*k)]-nu))
  p_wt = t(2*sweep(m,2,(sig2*eta*r),'*')*n*exp(-m**2))

  h = 1
  while (h < (k+1)){
    # part of X gradient
    grad_sw[h] = -(1/sw)*(p_wx[h,]%*%solve(Phi)%*%gama+
    (1-t(rep(1,(k+N-1)))%*%solve(Phi)%*%gama)*(p_wx[h,]
    %*%solve(Phi)%*%rep(1,(k+N-1)))/(t(rep(1,(k+N-1)))%*%
    solve(Phi)%*%rep(1,(k+N-1))))

    # part of T gradient
    grad_sw[h+k] = -(1/sw)*(p_wt[h,]%*%solve(Phi)%*%gama+
    (1-t(rep(1,(k+N-1)))%*%solve(Phi)%*%gama)*(p_wt[h,]%*%
    solve(Phi)%*%rep(1,(k+N-1)))/(t(rep(1,(k+N-1)))%*%
    solve(Phi)%*%rep(1,(k+N-1))))
```

```
    # gradient of X
    grad_w[h] = -p_wx[h,]%*%Phi%*%(y-mu*rep(1,(k+N-1)))*
    pnorm((min(y)-y_hat)/sw,mean=0,sd=1) + grad_sw[h]*dnorm
    ((min(y)-y_hat)/sw,mean=0,sd=1)

    # gradient of T
    grad_w[h+k] = -p_wt[h,]%*%Phi%*%(y-mu*rep(1,(k+N-1)))*
    pnorm((min(y)-y_hat)/sw,mean=0,sd=1) + grad_sw[h+k]*
    dnorm((min(y)-y_hat)/sw,mean=0,sd=1)
    h = h+1
  }
  return (grad_w)
}

# optim EI
library(rgenoud)
Lb = c(rep(0.001,k),rep(0.001,k))
Ub = c(rep(1,k),rep(24,k))
dom = cbind(Lb, Ub)
res= genoud(fw, nvars=2*k, Domains = dom, boundary.enforcement=1,
pop.size=1000, max.generations=100,max=TRUE,solution.tolerance=
0.01,gr=fw_grad)

# EI value and w matrix
w_opt = res$par
if (N==1){w = matrix(w_opt,c(1,2*k))} else {w = rbind(w,w_opt)}
EI[N] = res$value

# X matrix and T matrix
X_int = rbind(X_int, w_opt[1:k])
T_int = rbind(T_int,w_opt[(k+1):(2*k)])

# new response value
y2 = 0
u = (1:k)
v = t(combn(u,2))
l = 1
while (l < nrow(v)+1){
  y2 = y2 + s[v[l,1],v[l,2]]*(1/(1+abs(w_opt[(k+1):(2*k)][v[l,1]]
  -w_opt[(k+1):(2*k)][v[l,2]])/24))*(w_opt[v[l,1]]**2)*
  (w_opt[v[l,2]]**2)
```

```
    l = l+1
  }
  wt_opt = (30-w_opt[(k+1):(2*k)])/24
  y1 = sum(g*log(1+w_opt[1:k])*(2.4*wt_opt-2*wt_opt**2))
  y_new = -y1 - y2
  y = c(y,y_new)

  # stopping criterion
  if (N >= 3){
    ymax = max(-y[1:(N+k-2)])
    if ((EI[N]<0.015*ymax)&(EI[N-1]<0.015*ymax)&(EI[N-2]
    <0.015*ymax)){ break }
  }
}
```

Case 3: step 2

```
# initial design T
k = 4
T1 = matrix(runif(k**2,0,24), nrow=k)

# neighbor matrix
neighbour1 = function(xc){
  xn = xc
  ra = sample(k,1)
  r = sample.int(k,2)
  xn[ra,r[1]]=xc[ra,r[2]]
  xn[ra,r[2]]=xc[ra,r[1]]
  return(xn)
}

## vp function
p = 15
rho1 = 0.2
rho2 = 0.8
OF1 = function(xx){
  t = matrix(rep(1,k**2), nrow=k)
  i = 1
  while (i < k+1){
    j = 1
    while (j < k+1){
      l = 1
      while (l < k+1){
        if (xx[l,i]<xx[l,j]){t[i,j]=t[i,j]+1}
        l = l+1
      }
      j = j+1
    }
    i = i+1
  }
  diag(t)=NA
  sum1 = sum(apply(1/t**p, 1, sum, na.rm=TRUE))
  sum2 = 0
  i = 2
  while (i < k+1){
    j = 1
    while (j < i){
```

```
      sum2 = sum2 + 1/(sqrt(sum((xx[i,]-xx[j,])^2))+1)**p
      j = j+1
    }
    i = i+1
  }
  vp = (rho1*sum1+rho2*sum2)**(1/p)
  return(vp)
}

algo1 = list(nS = 3000L,
             vT = 0,
             neighbour = neighbour1,
             x0 = T1,
             printBar = FALSE)

library(NMOF)
sol1 = TAopt(OF1, algo = algo1)
T_int = sol1$xbest


# initial design X
library(LHD)
X0 = FastMmLHD(4,4)

## cp function
p = 15
rho3 = 0.5
rho4 = 0.5
OF2 = function(xx){
  sum3 = 0
  i = 2
  while (i < k+1){
    j = 1
    while (j < i){
      sum3 = sum3 + 1/(rho3*sqrt(sum((xx[i,]-
      xx[j,])^2))+rho4*sqrt(sum((T_int[i,]-T_int[j,])^2))+1)**p
      j = j+1
    }
    i = i+1
  }
  cp = sum3**(1/p)
  return(cp)
```

```
}

# neighbor matrix
neighbour2 = function(xc){
  xn = xc
  r = sample.int(k,2)
  xn[r[1],]=xc[r[2],]
  xn[r[2],]=xc[r[1],]
  return(xn)
}

algo2 = list(nS = 3000L,
             vT = 0,
             neighbour = neighbour2,
             x0 = X0,
             printBar = FALSE)

sol2 = TAopt(OF2, algo = algo2)

# rescale to [0,1]
fn = function(x){x/3}
X_int = apply(sol2$xbest,2,fn)


# coefficient and interaction
g = c(23.2,32.4,30,15)
s1 = rep(0,4)
s2 = c(8.1,0,0,0)
s3 = c(-16.8,-28.4,0)
s4 = c(-13.2,7.2,-11.6,0)
s = cbind(s1,s2,s3,s4)
y1 = rep(0,k)
y2 = rep(0,k)

# initial response
y1 = (log(1+X_int)*(2.4*(30-T_int)/24-2*((30-T_int)/24)**2))%*%g
u = (1:k)
v = t(combn(u,2))
l = 1
while (l < (nrow(v)+1)){
  y2 = y2 + s[v[l,1],v[l,2]]*(1/(1+abs(T_int[,v[l,1]]-
  T_int[,v[l,2]])/24))*(X_int[,v[l,1]]**2)*(X_int[,v[l,2]]**2)
```

```
  l = l+1
}
y = -y1 - y2

# GP model estimation
EI = NA
for (N in (1:50)){

  library(stats)
  x0 = c(rep(50,k),rep(10,k),rep(5,k),rep(10,k),10)
  Lb = c(rep(0.1,k),rep(0.1,k),rep(0.1,k),rep(0,k),0.1)
  Ub = c(rep(150,k),rep(100,k),rep(10,k),rep(24,k),100)

  # upper matrix index
  rcoord <- cbind(rep(seq_len(k+N-2), times = rev(
  seq_len(k+N-2))), unlist(lapply(X = rev(seq_len(k+N-2)),
  FUN = function(nn, nm) seq_len(nn) + nm - nn, nm = k+N-1)))

  # covariance function
  library('nloptr')
  kern =function(xx){
    Phi = matrix(rep(0,(k+N-1)^2), nrow=k+N-1)
    sig2 = xx[1:k]
    theta = xx[(k+1):(2*k)]
    eta = xx[(2*k+1):(3*k)]
    nu = xx[(3*k+1):(4*k)]
    tau2 = xx[4*k+1]

    m = 1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),
    2,eta,'*'))) - 1/(1+exp(-sweep(sweep(T_int[rcoord[, 2],],
    2,nu), 2,eta,'*')))

    n = sweep(exp(-sweep((X_int[rcoord[, 1],]-X_int
    [rcoord[, 2],])**2,2,theta,'*')),2,sig2,'*')

    Phi[rcoord] = apply(exp(-m**2)*n, 1, sum)
    Phi = Phi + t(Phi)
    diag(Phi) = sum(sig2) + tau2
    return (Phi)
  }

  # MLE estimation and gradient
```

```
eval_f_list = function(xx) {

  # MLE estimation
  fy1 = t(y)%*%solve(kern(xx))%*%y
  fy2 = 1/(t(rep(1,k+N-1))%*%solve(kern(xx))%*%rep(1,k+N-1))
  *(t(rep(1,k+N-1))%*%solve(kern(xx))%*%y)**2
  fx = log(det(kern(xx))) + fy1 - fy2

  sig2 = xx[1:k]
  theta = xx[(k+1):(2*k)]
  eta = xx[(2*k+1):(3*k)]
  nu = xx[(3*k+1):(4*k)]
  tau2 = xx[4*k+1]

  grad = rep(0,4*k+1)
  p_si = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_th = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_et = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_nu = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)
  p_ta = matrix(rep(0,(k+N-1)**2), nrow=k+N-1)

  h = 1
  while (h < (k+1)){
    m = 1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),
    2,eta,'*'))) - 1/(1+exp(-sweep(sweep(T_int[rcoord[, 2],],
    2,nu), 2,eta,'*')))

    n = (X_int[rcoord[, 1],]-X_int[rcoord[, 2],])**2

    p_si[rcoord] = (exp(-m**2)*exp(-sweep(n,2,theta,'*')))[,h]
    p_si = p_si + t(p_si)
    diag(p_si)=1

    p_th[rcoord] = -(sweep(n,2,sig2,'*')*exp(-sweep(n,2,theta,
    '*'))*exp(-m**2))[,h]
    p_th = p_th + t(p_th)

    # part of gradient calculation
    r = sweep(T_int[rcoord[, 1],],2,nu)*exp(-sweep(sweep(T_int
    [rcoord[, 1],],2,nu), 2,eta,'*'))

    s = sweep(T_int[rcoord[, 2],],2,nu)*exp(-sweep(sweep(T_int
```

```
[rcoord[, 2],],2,nu), 2,eta,'*'))

rr = (1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu), 2,eta,
'*'))))**2*r - (1/(1+exp(-sweep(sweep(T_int[rcoord[, 2],],2,nu),
2,eta,'*'))))**2*s

p_et[rcoord] = (-2*sweep(m,2,sig2,'*')*rr*exp(-sweep(n,2,theta,
'*'))*exp(-m**2))[,h]
p_et = p_et + t(p_et)

ss = (1/(1+exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),2,eta,
'*'))))**2*(exp(-sweep(sweep(T_int[rcoord[, 1],],2,nu),2,eta,
'*'))) - (1/(1+exp(-sweep(sweep(T_int[rcoord[, 2],],2,nu),2,eta,
'*'))))**2*(exp(-sweep(sweep(T_int[rcoord[, 2],],2,nu),2,eta,'*')))
p_nu[rcoord] = (2*sweep(m,2,(sig2*eta),'*')*ss*exp
(-sweep(n,2,theta,'*'))*exp(-m**2))[,h]
p_nu = p_nu + t(p_nu)

# prediction mean
mu_hat = as.vector(1/(t(rep(1,k+N-1))%*%solve(kern(xx))
%*%rep(1,k+N-1))*t(rep(1,k+N-1))%*%solve(kern(xx))%*%y)

# gradient of sig2
grad[h] = sum(diag((solve(kern(xx))%*%p_si)))-t(y-
mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_si%*%solve
(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))

# gradient of theta
grad[h+k] = sum(diag((solve(kern(xx))%*%p_th)))-
t(y-mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_th%*%
solve(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))

# gradient of eta
grad[h+2*k] = sum(diag((solve(kern(xx))%*%p_et)))-t(y-
mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_et%*%solve
(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))

# gradient of nu
grad[h+3*k] = sum(diag((solve(kern(xx))%*%p_nu)))-t(y-
mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_nu%*%
solve(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))
h = h+1
```

```
  }
  diag(p_ta) = 2*tau2

  # gradient of tau2
  grad[4*k+1] = sum(diag((solve(kern(xx))%*%p_ta)))-t(y-
  mu_hat*rep(1,k+N-1))%*%solve(kern(xx))%*%p_ta%*%
  solve(kern(xx))%*%(y-mu_hat*rep(1,k+N-1))

  return( list( "objective" = fx,
                "gradient"  = grad ) )
}

# optim parameters
opts = list("algorithm"="NLOPT_LD_LBFGS",
            "xtol_rel"=1.0e-6)
res = nloptr(x0 = x0,
             eval_f = eval_f_list,
             lb = Lb,
             ub = Ub,
             opts = opts)

# parameter estimated values
sig2 = res$solution[1:k]
theta = res$solution[(k+1):(2*k)]
eta = res$solution[(2*k+1):(3*k)]
nu = res$solution[(3*k+1):(4*k)]
tau2 = res$solution[4*k+1]

# cov matrix and mean
Phi = kern(res$solution)
mu = as.vector(1/(t(rep(1,k+N-1))%*%solve(Phi)%*%rep(1,k+N-1))
*t(rep(1,k+N-1))%*%solve(Phi)%*%y)

# EI function
fw = function(w){
  m = sweep(1/(1+exp(-sweep(sweep(T_int,2,nu), 2,eta,'*'))),
  2,1/(1+exp(-eta*(w[(k+1):(2*k)]-nu))))
  n = exp(-sweep((sweep(X_int,2,w[1:k]))**2,2,theta,'*'))
  gama = apply(sweep(n,2,sig2,'*')*exp(-m**2),1,sum)

  # prediction mean
  y_hat = as.vector(mu + t(gama)%*%solve(Phi)%*%(y-mu*rep(1,
```

```
  (k+N-1))))

  # covariance
  sw = sum(sig2)+tau2-t(gama)%*%solve(Phi)%*%gama+(1-t(rep(1,
  (k+N-1)))%*%solve(Phi)%*%gama)**2/(t(rep(1,(k+N-1)))%*%solve(Phi)
  %*%rep(1,(k+N-1)))
  sw = sqrt(as.vector(sw))

  # EI value
  EI = (min(y)-y_hat)*pnorm((min(y)-y_hat)/sw,mean=0,sd=1)+
  sw*dnorm((min(y)-y_hat)/sw,mean=0,sd=1)

  #chosen components
  if (length(which(w[1:k]>0.01))==k){
    j = 1
    while (j < k){
      l = j+1
      while (l < (k+1)){
        if ((abs(w[j+k]-w[l+k]))<3){EI=EI-1000}
        l = l+1
      }
      j = j+1
    }
  } else{
    c0 = which(w[1:k]<0.01)
    j = 1
    while (j < (k+1)){
      if (j==c0) {j = j+1}
      l = 1
      while (l < (k+1)){
        if ((l==c0) | (l==j)){l = l+1}
        if ((abs(w[j+k]-w[l+k]))<3){EI=EI-1000}
        l = l+1
      }
      j = j+1
    }
  }
  return(EI)
}

# EI gradient
fw_grad = function(w){
```

```
m = sweep(1/(1+exp(-sweep(sweep(T_int,2,nu), 2,eta,
'*'))),2,1/(1+exp(-eta*(w[(k+1):(2*k)]-nu))))
n = exp(-sweep((sweep(X_int,2,w[1:k]))**2,2,theta,'*'))
gama = apply(sweep(n,2,sig2,'*')*exp(-m**2),1,sum)

# prediction mean
y_hat = as.vector(mu + t(gama)%*%solve(Phi)%*%(y-mu*
rep(1,(k+N-1))))

# covariance
sw = sum(sig2)+tau2-t(gama)%*%solve(Phi)%*%gama+(1-
t(rep(1,(k+N-1)))%*%solve(Phi)%*%gama)**2/(t(rep(1,
(k+N-1)))%*%solve(Phi)%*%rep(1,(k+N-1)))
sw = sqrt(as.vector(sw))

grad_w = rep(0,2*k)
grad_sw = rep(0,2*k)
p_wx = t(2*sweep(sweep(X_int,2,w[1:k]),2,theta*sig2)
*n*exp(-m**2))
r = ((1/(1+exp(-eta*(w[(k+1):(2*k)]-nu))))**2)*exp(-eta*
(w[(k+1):(2*k)]-nu))
p_wt = t(2*sweep(m,2,(sig2*eta*r),'*')*n*exp(-m**2))

h = 1
while (h < (k+1)){
  # part of X gradient
  grad_sw[h] = -(1/sw)*(p_wx[h,]%*%solve(Phi)%*%gama+
  (1-t(rep(1,(k+N-1)))%*%solve(Phi)%*%gama)*(p_wx[h,]
  %*%solve(Phi)%*%rep(1,(k+N-1)))/(t(rep(1,(k+N-1)))%*%
  solve(Phi)%*%rep(1,(k+N-1))))

  # part of T gradient
  grad_sw[h+k] = -(1/sw)*(p_wt[h,]%*%solve(Phi)%*%gama+
  (1-t(rep(1,(k+N-1)))%*%solve(Phi)%*%gama)*(p_wt[h,]%*%
  solve(Phi)%*%rep(1,(k+N-1)))/(t(rep(1,(k+N-1)))%*%
  solve(Phi)%*%rep(1,(k+N-1))))

  # gradient of X
  grad_w[h] = -p_wx[h,]%*%Phi%*%(y-mu*rep(1,(k+N-1)))*
  pnorm((min(y)-y_hat)/sw,mean=0,sd=1) + grad_sw[h]*dnorm
  ((min(y)-y_hat)/sw,mean=0,sd=1)
```

```
    # gradient of T
    grad_w[h+k] = -p_wt[h,]%*%Phi%*%(y-mu*rep(1,(k+N-1)))*
    pnorm((min(y)-y_hat)/sw,mean=0,sd=1) + grad_sw[h+k]*
    dnorm((min(y)-y_hat)/sw,mean=0,sd=1)
    h = h+1
  }
  return (grad_w)
}

# optim EI
library(rgenoud)
Lb = c(rep(0.001,k),rep(0.001,k))
Ub = c(rep(1,k),rep(24,k))
dom = cbind(Lb, Ub)
res= genoud(fw, nvars=2*k, Domains = dom, boundary.enforcement=1,
pop.size=1000, max.generations=100,max=TRUE,solution.tolerance=
0.01,gr=fw_grad)

# EI value and w matrix
w_opt = res$par
if (N==1){w = matrix(w_opt,c(1,2*k))} else {w = rbind(w,w_opt)}
EI[N] = res$value

# X matrix and T matrix
X_int = rbind(X_int, w_opt[1:k])
T_int = rbind(T_int,w_opt[(k+1):(2*k)])

# new response value
y2 = 0
u = (1:k)
v = t(combn(u,2))
l = 1
while (l < nrow(v)+1){
  y2 = y2 + s[v[l,1],v[l,2]]*(1/(1+abs(w_opt[(k+1):(2*k)][v[l,1]]
  -w_opt[(k+1):(2*k)][v[l,2]])/24))*(w_opt[v[l,1]]**2)*
  (w_opt[v[l,2]]**2)
  l = l+1
}
wt_opt = (30-w_opt[(k+1):(2*k)])/24
y1 = sum(g*log(1+w_opt[1:k])*(2.4*wt_opt-2*wt_opt**2))
y_new = -y1 - y2
y = c(y,y_new)
```

```
  # stopping criterion
  if (N >= 3){
    ymax = max(-y[1:(N+k-2)])
    if ((EI[N]<0.015*ymax)&(EI[N-1]<0.015*ymax)&(EI[N-2]
    <0.015*ymax)){ break }
  }
}
```

Bibliography

Ba, S., Myers, W. R., & Brenneman, W. A. (2015). Optimal sliced latin hypercube designs. *Technometrics*, *57*(4), 479–487.

Berenbaum, M. C. (1977). Synergy, additivism and antagonism in immunosuppression. a critical review. *Clinical and experimental immunology*, *28*(1), 1.

Chou, T.-C. (2006). Theoretical basis, experimental design, and computerized simulation of synergism and antagonism in drug combination studies. *Pharmacological reviews*, *58*(3), 621–681.

Deng, X., Lin, C. D., Liu, K.-W., & Rowe, R. (2017). Additive gaussian process for computer models with qualitative and quantitative factors. *Technometrics*, *59*(3), 283–292.

Ding, X., Matsuo, K., Xu, L., Yang, J., & Zheng, L. (2015). Optimized combinations of bortezomib, camptothecin, and doxorubicin show increased efficacy and reduced toxicity in treating oral cancer. *Anti-cancer drugs*, *26*(5), 547–554.

Dueck, G., & Scheuer, T. (1990). Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of computational physics*, *90*(1), 161–175.

Holford, N. H., & Sheiner, L. B. (1981). Understanding the dose-effect relationship. *Clinical pharmacokinetics*, *6*(6), 429–453.

Iadevaia, S., Lu, Y., Morales, F. C., Mills, G. B., & Ram, P. T. (2010). Identification of optimal drug combinations targeting cellular networks: Integrating phospho-proteomics and computational network analysisintegrative identification of optimal drug combinations. *Cancer research*, *70*(17), 6704–6714.

Janus, N., Thariat, J., Boulanger, H., Deray, G., & Launay-Vacher, V. (2010). Proposal for dosage adjustment and timing of chemotherapy in hemodialyzed patients. *Annals of oncology*, *21*(7), 1395–1403.

Jaynes, J., Ding, X., Xu, H., Wong, W. K., & Ho, C.-M. (2013). Application of fractional factorial designs to study drug combinations. *Statistics in medicine*, *32*(2), 307–318.

Johnson, M. E., Moore, L. M., & Ylvisaker, D. (1990). Minimax and maximin distance designs. *Journal of statistical planning and inference*, *26*(2), 131–148.

Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, *13*(4), 455–492.

Joseph, V. R. (2016). Space-filling designs for computer experiments: A review. *Quality Engineering*, *28*(1), 28–35.

Kim, J., Taylor, D., Agrawal, N., Wang, H., Kim, H., Han, A., Rege, K., & Jayaraman, A. (2012). A programmable microfluidic cell array for combinatorial drug screening. *Lab on a Chip*, *12*(10), 1813–1822.

Kumar, S., & Jadon, P. (2014). A novel hybrid algorithm for permutation flow shop scheduling. *arXiv preprint arXiv:1407.5931*.

Lee, B.-Y., Clemens, D. L., Silva, A., Dillon, B. J., Masleša-Galić, S., Nava, S., Ding, X., Ho, C.-M., & Horwitz, M. A. (2017). Drug regimens identified and optimized by output-driven platform markedly reduce tuberculosis treatment time. *Nature communications*, *8*(1), 14183.

Luna, J., Jaynes, J., Xu, H., & Wong, W. K. (2022). Orthogonal array composite designs for drug combination experiments with applications for tuberculosis. *Statistics in Medicine*.

McKay, M. D., Beckman, R. J., & Conover, W. J. (2000). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, *42*(1), 55–61.

Mebane Jr, W. R. (2019). R version of genetic optimization using derivatives.

Mee, R. W. (2020). Order-of-addition modeling. *Statistica Sinica*, *30*(3), 1543–1559.

Morris, M. D., & Mitchell, T. J. (1995). Exploratory designs for computational experiments. *Journal of statistical planning and inference*, *43*(3), 381–402.

Mumenthaler, S. M., Foo, J., Leder, K., Choi, N. C., Agus, D. B., Pao, W., Mallick, P., & Michor, F. (2011). Evolutionary modeling of combination treatment strategies to overcome resistance to tyrosine kinase inhibitors in non-small cell lung cancer. *Molecular pharmaceutics*, *8*(6), 2069–2079.

Piepho, H.-P., & Williams, E. R. (2021). Regression models for order-of-addition experiments. *Biometrical Journal*, *63*(8), 1673–1687.

Rego, C., Gamboa, D., Glover, F., & Osterman, C. (2011). Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research*, *211*(3), 427–441.

Shah, M. A., & Schwartz, G. K. (2000). The relevance of drug sequence in combination chemotherapy. *Drug Resistance Updates*, *3*(6), 335–356.

Sobol', I. M. (1990). On sensitivity estimation for nonlinear mathematical models. *Matematicheskoe modelirovanie*, *2*(1), 112–118.

Van Nostrand, R. (1995). Design of experiments where the order of addition is important. *ASA proceedings of the section on physical and engineering sciences*, *155*, 160.

Voelkel, J. G. (2019). The design of order-of-addition experiments. *Journal of Quality Technology*, *51*(3), 230–241.

Wang, A., Xu, H., & Ding, X. (2020). Simultaneous optimization of drug combination dose-ratio sequence with innovative design and active learning. *Advanced Therapeutics*, *3*(4), 1900135.

Wang, B., Warden, A. R., & Ding, X. (2021). The optimization of combinatorial drug therapies: Strategies and laboratorial platforms. *Drug Discovery Today*, *26*(11), 2646–2659.

Wang, H., Xiao, Q., & Mandal, A. (2020). Musings about constructions of efficient latin hypercube designs with flexible run-sizes. *arXiv preprint arXiv:2010.09154*.

Xiao, Q., Joseph, V. R., & Ray, D. M. (2022). Maximum one-factor-at-a-time designs for screening in computer experiments. *Technometrics*, 1–11.

Xiao, Q., Wang, L., & Xu, H. (2019). Application of kriging models for a drug combination experiment on lung cancer. *Statistics in medicine*, *38*(2), 236–246.

Xiao, Q., Wang, Y., Mandal, A., & Deng, X. (2022). Modeling and active learning for experiments with quantitative-sequence factors. *Journal of the American Statistical Association*, (just-accepted), 1–40.

Xiao, Q., & Xu, H. (2018). Construction of maximin distance designs via level permutation and expansion. *Statistica Sinica*, *28*(3), 1395–1414.

Xiao, Q., & Xu, H. (2021). A mapping-based universal kriging model for order-of-addition experiments in drug combination studies. *Computational Statistics & Data Analysis*, *157*, 107155.

Yang, J.-F., Sun, F., & Xu, H. (2021). A component-position model, analysis and design for order-of-addition experiments. *Technometrics*, *63*(2), 212–224.

Zhang, Y., Tao, S., Chen, W., & Apley, D. W. (2020). A latent variable approach to gaussian process modeling with qualitative and quantitative factors. *Technometrics*, *62*(3), 291–302.

Zimmer, A., Katzir, I., Dekel, E., Mayo, A. E., & Alon, U. (2016). Prediction of multidimensional drug dose responses based on measurements of drug

pairs. *Proceedings of the National Academy of Sciences*, *113*(37), 10442–10447.